The University of New Mexico
Department of Computer Science
8th Student Conference

April 24, 2012

Preface

The Computer Science department at the University of New Mexico is an ABET-accredited program offering Bachelor of Science (BS), Master of Science (MS) and Doctor of Philosophy (PhD) degrees. The department was founded in 1979 and currently has 17 faculty members, 1 lecturers, 2 post-doctoral researchers, 120 graduate students, and 116 undergraduate students. Research strengths and interdisciplinary collaborations continue to expand. The annual student conference highlights cutting edge research currently in development by the graduate community under the leadership of the faculty. The conference is designed to promote active research in the department, foster excellence in presenting results, and encourage new collaborations.

Expertise and active research areas include:

- Automated reasoning and formal methods (PROFESSOR KAPUR)
- Artifical intelligence and Robotics (PROFESSORS TAPIA AND LUGER)
- Artifical life and evolutionary computing (PROFESSORS ACKLEY AND WILLIAMS)
- Biologically inspired computing (PROFESSORS ACKLEY, FORREST, AND MOSES)
- Biomedical engineering (PROFESSOR LUAN)
- Graphics, imaging (PROFESSOR KNISS)
- High performance computing and large scale distributed systems (PROFESSORS ARNOLD AND BRIDGES)
- Machine learning (PROFESSOR LANE)
- Molecular computing (PROFESSOR STEFANOVIC)
- Quantum computing (PROFESSOR MOORE)
- Programming languages (PROFESSORS KAPUR, STEFANOVIC, AND WILLIAMS)
- Security (PROFESSORS ACKLEY, CRANDALL, AND FORREST)
- Scientific visualization (PROFESSOR KNISS)
- Theory and algorithms (PROFESSORS HAYES, LUAN, MOORE, AND SAIA)

Table of Contents

Work-in-progress: Automated Named Entity Extraction for Tracking Censorship of Current Events

Antonio M. Espinoza
Computer Science Department
University of New Mexico
amajest@cs.unm.edu

Jedidiah R. Crandall
Computer Science Department
University of New Mexico
crandall@cs.unm.edu

Abstract

Tracking Internet censorship is challenging because what content the censors target can change daily, even hourly, with current events. The process must be automated because of the large amount of data that needs to be processed. Our focus in this paper is on automated probing of keyword-based Internet censorship, where natural language processing techniques are used to generate keywords to probe for censorship with. In this paper we present a named entity extraction framework that can extract the names of people, places, and organizations from text such as a news story. Previous efforts to automate the study of keyword-based Internet censorship have been based on semantic analysis of existing bodies of text, such as Wikipedia, and so could not extract meaningful keywords from the news to probe with.

We have used a maximum entropy approach for named entity extraction, because of its flexibility. Our preliminary results suggest that this approach gives good results with only a rudimentary understanding of the target language. This means that the approach is very flexible, and while our current implementation is for Chinese we anticipate that extending the framework to other languages such as Arabic, Farsi, and Spanish will be straightforward because of the maximum entropy approach. In this paper we present some testing results as well as some preliminary results from probing China's GET request censorship and search engine filtering using this framework.

1 Introduction

There are many open questions about Internet censorship, including how effective it is, what makes it effective, what kinds of targeted activities it is effective (or is not effective) at stopping, and so forth. A first step toward answering any of these questions is to collect enough data to understand how censorship is applied and what kinds of activities are targeted by the censors. This implies automated probing that is broad and carried out over a long period of time, because censorship within a single country can vary from province to province, company to company, and technology to technology and what content is targeted can change daily, even hourly.

1.1 Related work

Our focus in this paper is on keyword-based Internet censorship, and for the preliminary results we present we are interested specifically in China. Keyword-based Internet censorship in China has been studied by several groups of researchers, but is not well understood. An anonymous government official writing as "Mr. Tao", in a report published by Reporters Without Borders [7], described three types of keywords: masked words, sensitive words, and taboo words. According to Mr. Tao, a keyword list is produced and updated by the Information Office of the State Council. He adds, "each site adds key-words to its own filters in order not to run the risk of being criticised, punished or, worse still, closed down."

One of the more thoroughly studied forms of keyword-based Internet censorship is GET request filtering at the router level, where GET request packets containing blacklisted keywords cause routers in the backbone of China's Internet to forge reset packets and attempt to reset the TCP connection between the offending client and server. In contrast to HTML response filtering, which appears to have not been effective and may have been discontinued [8], GET request filtering is very effective in terms of the ratio of offending connections that are reset and is still pervasive on China's Internet today. The methods of China's HTTP keyword filtering were first published by the Global Internet Freedom Consortium [4]. Clayton *et al.* [2] published a more detailed study of this mechanism. The ConceptDoppler project [3] found that HTTP keyword filtering in China is not peremptory and is not strictly implemented at the bor-

der of the Chinese Internet, with a significant amount of filtering occurring in the backbone. The ConcpetDoppler project also used latent semantic analysis [6] to cluster words from the Chinese-language version of Wikipedia around sensitive concepts and then probe with these potentially sensitive words to see if they are censored by the GET request router-based mechanism. ConceptDoppler initially produced a list of 122 words, and has produced two more lists since.

Software that runs on servers in China, such as blogging software, also implements keyword-based censorship. One snapshot of a blacklist from a blog site is available in a Human Rights Watch Report [5, Appendix II] from 2006, for example. Client-side programs such as chat clients also implement keyword censorship. A blacklist for QQChat is available in the same report [5, Appendix I], and Villeneuve [9] gives a high-level analysis of topics censored by the chat client that is part of TOM-Skype.

Note that all of these lists are one-time-only snapshots. Some of the lists are very different from others, suggesting they come from different sources. Furthermore, our preliminary results indicate that the HTTP GET request blacklists that are used by routers in the backbone of China's Internet do not change on a daily, weekly, or even monthly basis. Existing systems that can continuously probe, such as ConceptDoppler, are based on document summary techniques that cluster words based on concept and therefore are not suitable for finding the named entities that are relevant to current events. Such document summary techniques can only compare documents and terms to an existing corpus of text based on the semantics that are latent in term and document frequencies, while named entity extraction gives additional semantic information about what a document is about based on its use of named entities. Because of the lack of data about Internet censorship and appropriate methods for gathering the data broadly and over a long period of time we have developed a named entity extraction framework, which we present in this paper.

1.2 Structure of the rest of the paper

We discuss the implementation of our framework in Section 2. Then we explain our experimental methodology for our preliminary results in Section 3 followed by the results in Section 4 and some concluding remarks.

2 Implementation

We implemented a *named entity extraction* (NEE) framework by means of *maximum entropy* (ME) machine learning. Borthwick *et al.* [1] demonstrated that an ME approach to NEE allows for flexibility in the choice of features to train on, since the interactions among features are not as important as they would be in other approaches such as Hidden Markov Models or Maximum Likelihood. We focused on three types of named entities: names of people, names of places, and names of organizations.

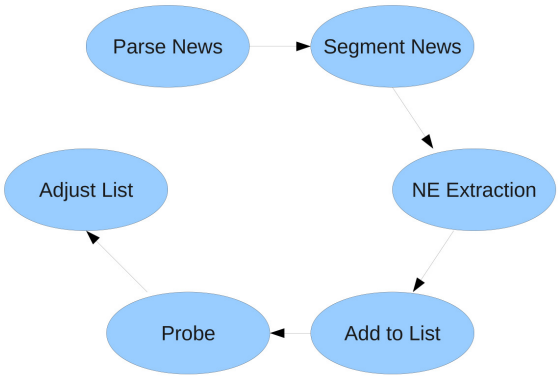


Figure 1: **The high-level workflow of our implementation.**

Our NEE framework requires a training corpus that has existing *labels*. That is, every word in the training corpus should be labeled with one of four labels: as a name of a person, name of a place, name of an organization, or not any of these types of named entities. The first three groups are then subdivided into complete, beginning, middle or end of the type of label. This is done so that it is possible for a named entity to span multiple segments after segmentation. We used the Chinese-language version of Wikipedia as our training corpus. When people, places, or organizations appear in Wikipedia, the reference is often a link to that person, place, or organization. In addition to the labeled data the ME framework also requires a feature vector for each word in order to build a model of *features* that correlate with particular labels. A feature is a property of the labeled word. One example feature is whether the word contains any characters that are common Chinese surnames. Another example feature is if the word is followed by a possessive such as Chinese 的 (de). The following table is a complete list of features used.

Feature	Test
Is place?	Does the word translate to a known place?
Has a name character?	Does the word contain a common name character?
Has punctuation?	Does the word contain any punctuation?
What punctuation (if any)?	What punctuation does the word contain?
Is month?	Does the word contain the character 月?
Has capital letters?	Does the translated word contain roman characters that are capitalized?
Has number?	Does the word consist of only roman numerals?
Has a Chinese number character?	Does the word contain a Chinese number character (一, 二, 三...十).
Has de?	Does the word contain the symbol 的?
Is a dictionary term?	Is the word in a Chinese dictionary?
Parts of Speech	All the parts of speech the translated word has.
Number characters	The number of characters in the word.

We used several heuristics to treat the Wikipedia corpus as a labeled data set. We assumed the link was a label for a name of a person if the document that was linked to had 年出生 (year of birth), 年逝世 (year of death), or 年逝世人物列表 (year of death person list) among its categories. We assumed the link was a label for a name of a place if the document that was linked to had GPS coordinates associated with it or contained one of the following infoboxes: country, city, cncity (cn=Chinese), prc provence (prc=Peoples Republic of China), or university. Lastly, we assumed the link was a label for an organization if it linked to an article that contained a company or organization infobox.

In all of the experiments in this paper, we trained on one third of the Wikipedia corpus using the above labeling scheme, and then tested on a different third of the corpus. For both training and testing, we applied Chinese text segmentation to the entire corpus to divide it into words (because the Chinese written language does not use spaces to divide sentences into words). Then we assigned a feature vector to each word based on the word itself, as well as the word that precedes and follows it. We trained and tested each of the three types of named entities separately.

Using an ME toolkit, we then assigned conditional probabilities to each word for each sublabel conditioned on its feature vector. Because the probabilites given were for a word being the begining 'label', middle 'label', end 'label', complete 'label', or not a 'label' (where 'label' is person, place, or organization), we had to find the highest probable legal path through the output. In order to be a legal path sublabels must be in correct order, for example end 'label' cannot precede middle 'label' legally. Similary beginning, middle, and end 'label' cannot be surrounded by not 'label' on both sides. In order to accomplish this we used the fact the output of the ME tookit can implicitly be thought of as a directed acyclic graph. Therefore we were able to preform a topological sort to find the highest probable legal path.

For testing or for the actual probing experiments, we take an unlabeled corpus of text (or a test set where the labels are withheld), and then assign a label to each word based on the ME model of the training set. We scale the conditional probabilities in the model linearly so that we get a desirable fraction of labeled words.

See Figure 1 for a high-level workflow of our implementation. For probing, we have written parsers for seven Chinese-language news websites. Our framework downloads the news from these websites every day and performs named entity extraction based on the model that was created using Wikipedia. For any word that is labeled as a named entity, we include that word in our list of keywords to probe with on that day. Our probing infrastructure performs two kinds of probes, it tests twelve servers for HTTP GET request filtering based on forged RSTs, and it tests two search engines to see if the word elicits a legal message in Chinese stating that entries have been removed from results for the search query. Our probing infrastructure has multiple priority levels, with levels with lower numbers being higher in priority for testing. If a word is ever interpreted to be blacklisted, it is placed in priority 0 so that it will be tested every 12 hours for the remainder of the probing. Words enter the probing infrastructure at level 1. Every 12 hours level 0 words are probed, followed by level 1 words, then level 2, and so on. If a word does not appear to be blacklisted, it is moved down one priority, except if it is in priority 0 in which case it remains in priority 0. There are 15 priorities, with the lowest being 14. After a word has been probed 14 times and does not appear to be censored, it falls off the bottom of the list.

In order to get search engine results that are independent of GET request censorship, we divide GET requests for the two search engines we test against into separate packets that will be reconstructed by the server but will evade GET request filtering. For testing for forged RSTs, we wait at least 100 seconds between each query for any pair of IP addresses. As a special consideration, the search engine results do not affect the priorities of keywords, because we found this to cause many words that

were not actually targets of censorship to be in priority 0. We record a traceroute to each server every hour, so that any major changes in the keyword censorship that might be due to changes in routing can be explained.

3 Expiremental Methodology

For the preliminary results we present in this paper, there are two experiments that we performed. One experiment is to measure the specificity and recall of the NEE framework on a different third of Wikipedia than the training set. This gives us baseline numbers to see how well the NEE framework is performing. The other experiment for which we have preliminary results is a test run of approximately two months (with some downtime) in which we ran the entire NEE and probing framework and obtained some results that are censored topics from the news.

For the first experiment, we focus on specificity instead of precision because of the context of probing with keywords. Precision is the probability that a word labeled as a named entity actually is a named entity. Since there are no human consumers of the output of our NEE framework, precision is not as relevant. Any word that is not a named entity but is labeled as such will be probed with, perhaps unnecessarily, but this is relatively acceptable compared to missing named entities. What we wish to consider instead is how much extra probing we have to do to ensure that we label a good fraction of the named entities as named entities. Thus, recall and specificity are better indicators of performance in our context than recall and precision. Recall is the probability that an actual named entity is labeled as a named entity. Specificity is the proportion of words that are not named entities that are not labeled as named entities. Thus, as long as the specificity remains high enough that NEE is saving us about an order of magnitude of probing compared to just probing with every word, we can trade off precision for recall and achieve a high recall while greatly reducing the amount of necessary probing.

For the second experiment, our initial two months of running the entire infrastructure includes downloading and parsing the news from seven sources every day, labeling the named entities, and probing for both GET request and search engine censorship. This data has various issues such as downtime and the need to remove some polluted data manually, but gives promising anecdotal evidence that censorship of current events can be detected using NEE. We provide a summary of the types of words we found to evoke censorship and how the different forms of censorship seem to vary with the news, with the caveat that these are preliminary results and no certain conclusions can be drawn from them at this time.

4 Results

In this section we present both sets of results: results from testing for the specificity and recall by withholding labels from the Wikipedia dataset, and preliminary results from an initial two-month run of the entire infrastructure.

4.1 Specificity and recall

For labeling the names of people, we obtained the following results:

- Specificity: 83.44%
- Recall: 89.63%
- Precision: 0.42%

A precision of 0.42% is usually not considered to be very good for a named entity extractor, but remember that our context is different. One way to interpret these results is that we can label only 16.6% of the words in our dataset as names of people (thus reducing the amount of probing necessary by nearly an order of magnitude), and include 89.63% of the actual names of people in our probing by doing so.

The results for names of places are as follows:

- Specificity: 69.80%
- Recall: 96.3%
- Precision: 0.77%

And, finally, the results for names of organizations are as follows:

- Specificity: 88.40%
- Recall: 87.56%
- Precision: 0.28%

4.2 Initial two-month run

One of the more surprising results from our initial two months of data is that the HTTP GET request blacklist appears to be fairly static. That is, words do not seem to be added to or removed from this particular censorship blacklist on a daily, weekly, or even monthly basis. We remind the reader that these are preliminary results and our data has some downtime and other issues. However, our data was taken during a time of many reports of arrests and censorship related to the Jasmine Revolution protests in China in 2011, and despite many of these current events being censored in search engines we probed

with these keywords for HTTP GET request censorship and saw none that was related to any current event. Nor did we see any evidence of any keyword being added or removed in our preliminary results.

We did notice that current events evoked censorship in search engines, however. Specifically, certain words caused the search engine to return a warning that results had been removed due to local laws. Note that this probably means that a website was removed that contained the word we probed with and was highly ranked, it does not mean that the word itself is on any keyword blacklist. This is an important distinction. We determined that this is probably the case with the following experiment. We searched for both "fuck" and "fuck you" in both search engines that we used for probing. The word "fuck" causes the message saying results have been removed to appear, while "fuck you" does not cause the message to appear. This suggests that this form of censorship is more topical and not based solely on a certain byte string appearing in the query. However this does not preclude a blacklist for search engine censorship.

We witnessed search engine censorship of certain words from the news that we assert is definitely censorship based on current events because of the words themselves. Some of the words are:

- 茉莉花 (Jasmine Flower): related to the Jasmine Revolution protests.
- 诺贝尔 (Nobel Prize), 刘先生 (Mr. Liu), LIU, Xiao, Liu, and 挪威 (Norway): these are all related to Liu Xiaobo winning the Nobel Prize while imprisoned by China.
- 七十七, 77, 七七 (all mean the number 77): these elicited censorship at a time when China's President was being criticized by many Chinese citizens. He had visited a woman at her home during a live newscast, and asked her how much she pays for rent. She replied that she paid 77 RMB, or approximately 12 dollars, per month. It is likely that she is on a government program and this is the small portion of the actual rent that she pays, but many felt that she had been coerced by the government to claim that her rent was very low to make the government look good.
- 王府井 (Wangfujing): this is an area in Beijing where some of the Jasmine Revolution protests happened.

5 Concluding Remarks

In conclusion, our preliminary results are promising in terms of building an infrastructure that can probe censorship with words from current events. Our NEE algorithm gives a good specificity and recall, and we demonstrated that this infrastructure can produce instances of censorship that are related to current events.

6 Acknowledgments

We would like to thank the anonymous FOCI reviewers for their insightful comments. We would also like to thank the many people who helped us improve our translations and gave feedback on other aspects of the paper. Fletcher Hazlehurst, Veronika Strnadova, Leif Guillermo, and Ronald Garduno helped with various aspects of implementation or understanding of the maximum entropy framework and features. This material is based upon work supported by the National Science Foundation under Grant Nos. #0844880 and #1025442. Antonio Espinoza and Jedidiah Crandall are also supported by the DARPA CRASH program.

References

[1] BORTHWICK, A., STERLING, J., AGICHTEIN, E., AND GRISHMAN, R. Exploiting diverse knowledge sources via maximum entropy in named entity recognition. In *In the Proceedings of the Sixth Workshop on Very Large Corpora* (1998), pp. 152–160.

[2] CLAYTON, R., MURDOCH, S. J., AND WATSON, R. N. M. Ignoring the Great Firewall of China. *I/S: A Journal of Law and Policy for the Information Society 3*, 2 (2007), 70–77.

[3] CRANDALL, J. R., ZINN, D., BYRD, M., BARR, E., AND EAST, R. ConceptDoppler: a weather tracker for Internet censorship. In *Proc. of 14th ACM Conference on Computer and Communications Security (CCS)* (2007).

[4] The Great Firewall Revealed. Whitepaper released by the Global Internet Freedom Consortium in December of 2002.

[5] "Race to the Bottom": Corporate Complicity in Chinese Internet Censorship. In *Human Rights Watch* (August 2006). `http://www.hrw.org/reports/2006/china0806`.

[6] LANDAUER, T. K., FOLTZ, P. W., AND LAHAM, D. Introduction to latent semantic analysis. *Discourse Processes 25* (1998), 259–284.

[7] MR. TAO. China: Journey to the heart of Internet censorship. Investigative report sponsored by Reporters Without Borders For Freedom and Chinese Human Rights Defenders, Oct 2007.

[8] PARK, J. C., AND CRANDALL, J. R. Empirical study of a national-scale distributed intrusion detection system: Backbone-level filtering of html responses in china. In *Proceedings of the 2010 IEEE 30th International Conference on Distributed Computing Systems* (Washington, DC, USA, 2010), ICDCS '10, IEEE Computer Society, pp. 315–326.

[9] VILLENEUVE, N. Breaching trust: An analysis of surveillance and security practices on China's TOM-Skype platform. Available at `http://www.infowar-monitor.net/breachingtrust/`.

Internet Topology over Time

Benjamin Edwards
University of New Mexico
bedwards@cs.unm.edu

Steven Hofmeyr
LBNL
shofmeyr@lbl.gov

George Stelle
University of New Mexico
stelleg@cs.unm.edu

Stephanie Forrest
University of New Mexico
forrest@cs.unm.edu

***Abstract*—There are few studies that look closely at how the topology of the Internet evolves over time; most focus on snapshots taken at a particular point in time. In this paper, we investigate the evolution of the topology of the Autonomous Systems graph of the Internet, examining how eight commonly-used topological measures change from January 2002 to January 2010. We find that the distributions of most of the measures remain unchanged, except for average path length and clustering coefficient. The average path length has slowly and steadily increased since 2005 and the average clustering coefficient has steadily declined. We hypothesize that these changes are due to changes in peering policies as the Internet evolves. We also investigate a surprising feature, namely that the maximum degree has changed little, an aspect that cannot be captured without modeling link deletion. Our results suggest that evaluating models of the Internet graph by comparing steady-state generated topologies to snapshots of the real data is reasonable for many measures. However, accurately matching time-variant properties is more difficult, as we demonstrate by evaluating ten well-known models against the 2010 data.**

I. Introduction

The Internet is growing rapidly, having more than tripled in size in the last decade, from 10,000 Autonomous Systems (ASes) in 2002 to 34,000 in 2010. However, few studies have looked carefully at the time evolution of the Internet topology at the AS-level. Most studies consider a snapshot of the AS-level topology of the Internet, derived from the latest data available at the time of the research, e.g. [1], [2], [3], [4].

In this paper, we are interested in how the topology of the AS-level Internet changes over time. We take the common approach of regarding the Internet as a graph, where the vertices are ASes and the edges are routing links between them. There are many properties of the Internet graph that can be investigated, from the simple degree distribution to more complex measures such as betweenness centrality. We select a set of eight commonly used measures that are relevant to the way the Internet functions. The measures are described in section II.

We investigate how the selected measures change over the period from January 2002 to January 2010, and present the results in section IV. We find that the distributions for most of the measures remain unchanged except average path length and clustering coefficient. Since 2005, the average path length has slowly and steadily increased and the average clustering coefficient has steadily declined. These results may signify changes in peering relationships in the Internet; we discuss this idea in section VI.

Our results imply that Internet topology models can be evaluated using single snapshots of the topology in time for many measures but not all. We can expect that models that matched invariant measures five years ago will still match today. To this end we include a brief summary of 10 well-known models and their performance on the latest data set in subsection IV-C. We find that models that were accurate when originally proposed, often many years ago, still accurately predict many time-invariant AS features (such as centrality), while doing a poorer job on the time-variant measures, such as clustering coefficient. In addition, the unchanging maximum degree of the Internet is often poorly predicted.

II. Topological properties of the Internet

We selected a set of eight measures for our analysis of Internet topology evolution. Although many more measures are available, e.g., [5], [6], [7], [8], we chose those most commonly used in the past to evaluate both generative models and data of the actual AS topology [9]. Further, we selected measures that seem most relevant to understanding the functioning of the Internet. Table I summarizes the measures and our rationales for choosing them. Given space limitations, we restrict our attention to graph-based measures, ignoring network operation constraints, traffic flow analysis, and distributions of AS relationship types.

The measures are divided into four categories: *Node Centrality*, *Path Length*, *Community Structure*, and *Scale Free Structures*. We also track simple properties such as the maximum and average degree of ASes.

A. Node Centrality

Node centrality measures are related to the prominence of ASes, which is important when evaluating the implications of ISP regulation [10] or the robustness of the Internet [11]. We use three measures of centrality: the node *degree*, the *betweenness*, which is the fraction of all shortest paths that pass through a node [12], and the *page rank*, which is the number of times that a node will be visited on a sufficiently long random walk on the graph [13]. The betweenness centrality is inversely related to the robustness of the graph to removal of nodes, because the more paths that pass through a node, the more damage will be done when that node is removed.

Measure	Rationale
Degree Centrality	Simplest way to measure AS prominence
Betweenness Centrality	AS prominence under best (shortest-path) routing. Inversely related to robustness to node deletion
Page Rank Centrality	AS prominence under average (random) routing.
Path Length	Related to routing efficiency (hops between source and destination)
Clustering Coefficient	Related to the peering structure of the Internet, and routing resilience (number of alternative routes)
K-Cores Decomposition	Related to tier structure of the AS Graph
Assortativity	Relevant to peering relations
S-Metric	Distinguishes among scale-free graphs, alternate measure of assortativity

TABLE I
SUMMARY OF MEASURES.

B. Path Length

The *average shortest path length* from a node to all other nodes in the graph (the geodesic distance)[1] is important because it relates to the number of routing hops between ASes. Not all packets travel along the shortest paths because of business agreements (such as the valley-free rule), but to a first approximation, routing distances (hops on the AS graph) are largely determined by the shortest paths.

Alternative path length measurements include diameter, which is the longest of the shortest paths between all pairs of nodes, and the effective diameter, which is the path length that defines the 90th percentile of all paths [15]. In the Internet, the distribution of path lengths has small variance, so diameter and effective diameter are only slightly larger than the shortest average path length and highly correlated with it. Hence we use only the average shortest path length.

C. Community Structure

Community structure measures how groups of nodes form substructures within the graph and is relevant to understanding various aspects of the Internet, such as the tiered structure and resilience to node deletions. Although there are many community structure measures [16], [17], we chose three that reveal important features of the AS graph. The first measure is the *local clustering coefficient*, which is the number of edges among the neighbors of a node as compared to the maximum possible number [18].[2] The clustering coefficient is related to the resilience of the routing infrastructure, because it reflects the number of alternative routes between pairs of nodes (for example, a tree has a coefficient of 0 and the removal of any edges will partition the graph). The second community structure measure is *degree assortativity*, which measures whether nodes tend to connect to others of similar degree [19]. The final measure is *k-cores decomposition*, which measures successive maximally connected subgraphs [20]. We report two measures for k-cores: the distribution of k-cores values, and the size of the maximum core, k-max.

[1]This is sometimes referred to as *closeness centrality*, though there are other definitions for closeness centrality [14].

[2]We do not use transitivity, which is an alternative definition of the clustering coefficient, because it tends to be highly correlated with the average degree and so does not yield additional useful information.

D. Scale-free Structures

The power-law degree distribution of the AS graph is a scale-free property often cited as a distinguishing feature of the Internet, e.g., [3]. If the AS graph can be described as scale-free it may share properties with other scale-free networks, for example, the tendency to be 'robust yet fragile' or the preferential attachment growth dynamic. However, Li et al [21] showed that it is possible to construct graphs and general data sets that have similar scale-free properties but very different structures. To address this issue, Li et al. propose the *s-metric*—the sum over edges of the product of the degree of the two nodes an edge connects. This computation yields a single value that measures the extent to which a graph is actually scale-free.

III. DATA SETS

To investigate how the Internet changes over time, we collected a set of AS graphs covering the period from January 2002 to January 2010 by parsing monthly snapshots of BGP routing table dumps from Oregon Route Views and RIPE.[3] Although BGP routing tables are dumped every few hours, monthly snapshots were of sufficient temporal resolution given the long time scale of the analysis. The monthly snapshots were compiled by parsing all of the dumps from the first day of each month, taking every adjacent pair in the ASPATH and adding them to the graph for that month. We did not filter out self loops, private Autonomous Systems Networks (ASNs), or any other potential spurious or inaccurate results from the dumps, as it is generally assumed that the number of false positives of this type are small [22].

One potential problem with the BGP data is that nodes and edges disappear and reappear due to the way the data are sampled. While there are other ways of dealing with disappearing edges and nodes [15], we assumed that nodes and edges that temporarily disappear from the BGP tables actually exist throughout from first appearance to last. Our data set is available[4] in networkx[5] format.

To validate our results, we ran identical experiments using data collected from the Cooperative Association of Internet Data Analysis (CAIDA) [23], and obtained essentially identical results. The collectors of the CAIDA data sets went to great

[3]www.routeviews.org andwww.ripe.net

[4]https://ftg.lbl.gov/projects/asim/data-2/

[5]http://networkx.lanl.gov/

length to deal with various false-positive errors and the issue of nodes and edges that temporarily disappear, so the close agreement between the two data sets indicates that relatively simple preprocessing of the data is adequate for the purposes of our study.

Although false positives in the data are likely rare, false negatives (missing links) are likely common because the BGP dumps do not capture peering links between smaller ASes on the edge of the graph.[6] Although several studies have attempted to quantify the number of missing links (e.g. [22], [24], [25]), it is difficult to determine how exactly those hidden links could affect the structure of the AS-graph. Consequently, we focus on the *visible* Internet, in which we see subtle topological changes (see section IV) that we speculate could be caused by an increase in missing links.

IV. Results

Most of the measures yield a distribution rather than a single value. Although we can plot the distributions together, year by year, it is also useful to have a single value for determining the changes over time. A common approach to this problem aggregates distributions, using measures of central tendency, extent, or spread [9]. However, studying the distributions as a whole before aggregating allows us to discover changes to the shape of the distribution (e.g. a transition from an exponential distribution to a power-law) that might not be revealed under aggregation. Consequently, we test whether the distributions between years differ using the Cramér–von Mises Criterion (CMC) [26]. The CMC tests the hypothesis that two samples of data are drawn from the same distribution. Although many alternative tests and measures exist [27], [28], [29], the CMC gives accurate comparisons and captures intuitive similarities between plots that can be seen visually.

We used the CMC to identify year by year changes in all of the measures that have distributions. Table II shows the changes from one year to the next from 2002 to 2010. For each year, we applied the measures to the AS-graph for June; varying the month of data collection does not vary the results. For measures that do not have distributions (k-max, assortativity and the s-metric), Table II reports the absolute values, rather than the year by year differences.

A. Unchanging Features

Table II shows that the node centrality measures (degree, betweenness and page rank) stay constant over time. Figure 1 illustrates this point, showing that the power-law degree distribution is virtually identical over time. We obtain similar results for betweenness and page rank—the distributions are stable over time (data not shown).

In Figure 1 not only is the slope of the distribution unchanging, but the *extent* (maximum degree) is nearly constant. Only three ASes have had the maximum degree in the years 2002 to 2010: MCI Inc., Level 3 Communications, and Cogent Communications (see Figure 2). MCI, which held the top

[6]Roughan et al [22] estimate that an AS graph extracted from public BGP views is likely to miss 27% of links overall, and 70% of peer-to-peer links.

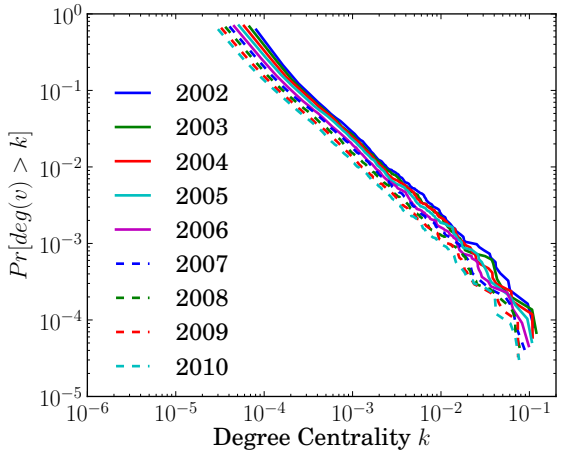


Fig. 1. Complement cumulative distribution of the degree of ASes between 2002-2010

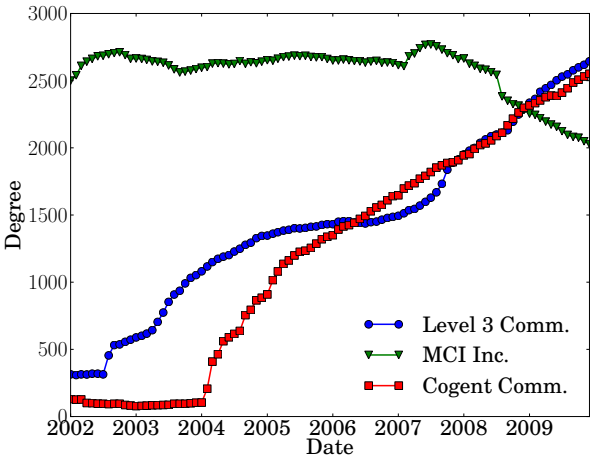


Fig. 2. Degree changes in the three largest degree ASes.

position until late 2008, maintained a nearly constant degree, while Cogent and Level 3 had a nearly monotonic increase in degree over the same time period. Growth is not guaranteed for an AS: 61% of all ASes experienced a month to month decline in degree at least once between 2002 and 2010 and fully 84% of ASes that have existed for five or more years experienced at least one monthly decline.

Other clearly unchanging features in Table II are the k-cores and the degree assortativity. The s-metric appears to gradually increase over the entire period, but this could be a meaningless change: According to Li et al. [21], graphs with low s-metric values (below 0.1) are likely "scale-rich" and difficult to differentiate from each other using the s-metric.

B. Changing Features

Two measures exhibit distinct changes over time: the average path length and the clustering coefficient. The changes in the distribution of path length, although statistically significant, are not easily visible on a plot. The clustering coefficient, on the other hand, has changes that are clearly visible, as can

Year	Nodes	Edges	Degree	Betweenness	Page Rank	Path Length	Clustering	K-Cores	K-max	Assort.	S-Metric
2002	13172	26695							16	-0.189	.0114
2003	15446	32089	NS	NS	NS	****	NS	NS	18	-0.188	.0121
2004	17722	39654	NS	NS	NS	****	NS	NS	22	-0.188	.0159
2005	20174	45505	NS	NS	NS	****	NS	NS	24	-0.196	.0176
2006	22708	50796	NS	NS	NS	****	*	NS	24	-0.189	.0183
2007	25691	58200	NS	NS	NS	****	NS	NS	26	-0.189	.0199
2008	28640	64111	NS	NS	NS	****	**	NS	25	-0.192	.0203
2009	31645	69938	NS	NS	NS	****	*	NS	25	-0.195	.0210
2010	34055	71544	NS	NS	NS	****	****	NS	21	-0.191	.0179

TABLE II

ANNUAL CHANGE OF MEASURES. WE REPORT COMPUTED VALUES FOR K-MAX, ASSORTATIVITY AND S-METRIC. FOR OTHER MEASURES, WE REPORT THE STATISTICAL SIGNIFICANCE OF THE CHANGE BETWEEN YEARS (NS: NOT SIGNIFICANT, *: $p < .1$, **: $p < .05$, ***: $p < .01$, **** : $p < .001$).

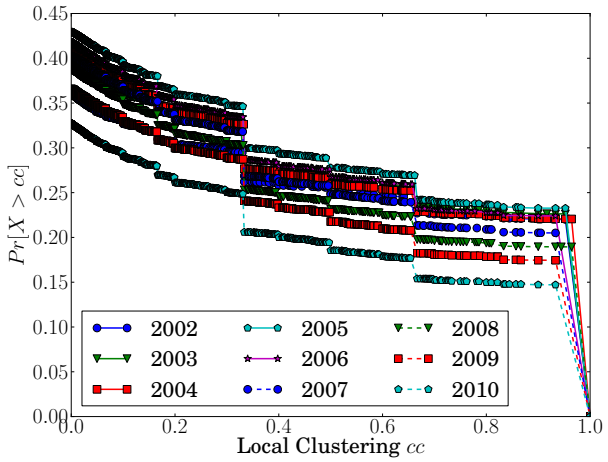


Fig. 3. Complement cumulative distribution of the local clustering coefficient of the AS graph

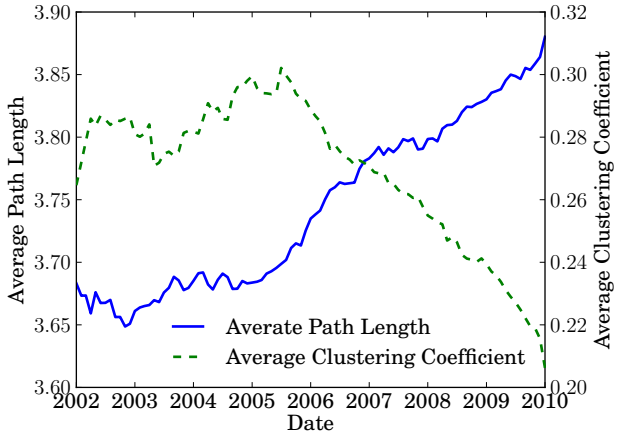


Fig. 4. The change in the average path length and the average clustering coefficient, plotted monthly. Both clustering coefficient and path length experience a nearly constant variance of .16 and .3 respectively, and so we do not report error bars in this figure.

be seen in Figure 3. Although the distribution changes little in shape, there is a distinct downward trend over time.

The trends in path length and clustering coefficient are evident in Figure 4, which plot averages over time. We can see that around 2005, the structure of the visible AS graph began to change. The average path length and average clustering coefficient do not exhibit any clear trend until mid 2005, when they began slowly increasing and decreasing respectively. In addition, the k-max value was steadily increasing up to 2005, after which it leveled off (see Table II). From these changes we can infer that both the efficiency of the *visible* Internet (as measured by number of hops on the shortest paths between locations) and resilience to router failure, have been decreasing since 2005. We discuss possible reasons for these changes in section VI.

The shift in topology beginning in 2005 indicates that the Internet has not reached steady-state. This observation contradicts many models of the AS graph, in which measures converge to a steady-state distribution. The next section studies how well different models of Internet topology match the AS graph on the different measures.

C. Topological Models of the Internet

The relatively slow changes we see in the Internet imply that generative models that give accurate results at one size should do so at most sizes. To check this we generated topologies from 10 different models and compared them to the AS graph from June 2010. Table III lists the models we considered. In each case, we generated a topology of 34,055 nodes, the same number as our latest snapshot of the AS graph.

The comparisons of the different models to the AS graph are shown in Figure 5. As expected, later models are more accurate than earlier models (the models are ordered from oldest at the top to most recent at the bottom). All models generate close matches to the degree distribution, which makes sense because this is typically the first measure authors use to evaluate a model, and it has not changed over time. Further, most models match the other measures of centrality quite well, presumably because they are also time-invariant.

It is notable that all the models perform worst on the measures that change the most, namely path length and clustering. In general, models match time-invariant measures better than time-varying measures, with the exception of maximum degree, which no model captures well.

Name	Abbr.	Description	Reference
Barábasi-Albert	BA	Original preferential attachment model	[1]
Heuristically Optimized Trade-offs	FKP	Local optimization model	[30]
Generalized Linear Preference	GLP	Modified preferential attachment model	[31]
Univariate Heuristically Optimized Trade-offs	UFKP	Modified local optimization model	[32]
Bivariate Heuristically Optimized Trade-offs	BFKP	Modified local optimization model	[32]
Multivariate Heuristically Optimized Trade-offs	MFKP	Modified local optimization model	[32]
Interactive Growth	IG	Modified preferential attachment model	[33]
Positive Feedback Preference (1)	PFP1	Modified IG model	[33]
Positive Feedback Preference (2)	PFP1	Modified PFP1 model	[33]
ASIM	ASIM	Agent based topology generator	[10]

TABLE III
SUMMARY OF MODELS EVALUATED.

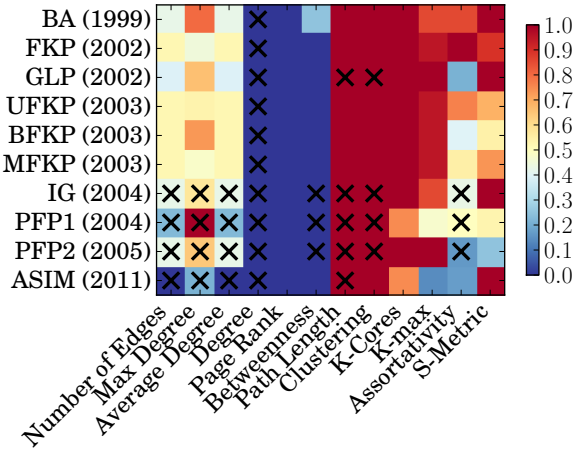


Fig. 5. Evaluating models. The color of each square corresponds to the statistical significance level for the Cramér-von Mises criterion when comparing the model to the January 2010 AS graph for a given measure. The number of stars (0 to 4) is scaled between 0 and 1 for consistency with the other measures. For k-cores, degree assortativity and the s-metric, the color corresponds to the relative error. Hence lower values mean more similar distributions. X's indicate that the measure was reported in the original publication of the model.

It is not surprising that most generative models we studied performed well on the time-invariant measures the models were originally tested on. However, most models perform much worse on measures they were not originally tested against. This indicates the importance of evaluating models against a wide variety of measures, because many different graphs can be similar in one measure, but it is much harder for different graphs to be similar in multiple measures.

V. Related Work

There are few studies that look at changes in topological characteristics beyond the number of nodes and edges. Most of those that do focus on inter-AS relationships, for example, Chang et al [34] study the changes in customer-provider relationships and find that the number of providers is increasing over time. Another approach was taken by Oliveira et al [35], in which they investigate the changing relationship over time between stub ASes and transit providers. They find that the net growth of rewirings for transit providers levels off at the end of 2005, around the same time our results show subtle changes in the Internet. Gill et al [36] look more closely at the evolution of peering relationships, and find that over time large content providers are relying less on Tier-1 ISPs, and more on peering with lower tiers. This finding is supported by Labovitz et al [37], who report a rapid increase in the traffic flow over peer links over time, resulting in a less hierarchical Internet topology. These observations could potentially explain some of our results, as we discuss in section VI.

In addition to studying business relationships, Dhamdhere el al [38] reported on changes in average degree and average path length over time. Their results on path length agree with ours, although their study included only data up to 2007, so the trends are less clear. Because the degree distribution does not change, it is likely that the shift they see in average degree is a result of a steadily increasing sample size. Another study [39] used spectral analysis to investigate clustering on the AS graph, and study coreness and changing path diversity. This analysis, however, covers short time spans (at most two and a half years), and only considers data before 2004.

The work of Zhang et al [40] is perhaps closest to ours. They study changes in several topological measures over the time period from 2001 to 2006. Because of this time period, their results do not capture the trends we report post-2005. However, the changes they document agree with what we observed in the earlier period: They find the assortativity and k-cores are stable over time and from 2004/2005 onwards, the k-max value changes little. Further, they find the average clustering coefficient starts declining around 2005, and the average path length starts increasing gradually.

VI. Discussion

We have reported a distinct shift in the topology of the *visible* Internet since 2005: the average path length is increasing, and the average clustering coefficient is decreasing (Figure 4). On the surface, it would appear that the Internet is getting both less efficient and less resilient. But this may not actually be the case, because the the shift is likely caused by changes in peering policies that affect the hidden Internet and cannot be measured with public BGP dumps. As mentioned in section V, there are several studies showing that content providers are routing more traffic over hidden peer-to-peer links, and relying less on the more publicly visible Internet

infrastructure. Consequently they have less need to establish new customer-provider links, and a decreasing number of new customer-provider links increases the *observed* average path length of the graph.

Most models match topological measures that are invariant over time in the AS graph, particularly centrality. However performance degrades when examining time-variant measures such as average path length and clustering coefficient. Future modeling efforts will need to focus on incorporating mechanisms that can cope with such changing dynamics. For example, few existing models allow for the loss of links in the AS graph, a common feature according to our data. Agent-based models such as ASIM are potentially a promising direction for future AS topology modeling efforts because they can naturally model economic pressures that lead to link deletion. Further, robust statistical techniques such as the CMC will be needed to verify topological results.

In conclusion, it is surprising that so few of the common measures of Internet topology have changed over the past eight years, even though the number of ASes has tripled during this time period. Those measures that do change point to the increasing importance of understanding the role of policy and economics in determining Internet topology. Going forward, it will be increasingly important to find ways to reveal hidden links and evolving peering relationships.

VII. Acknowledgments

The authors gratefully acknowledge the support of DOE grant DE-AC02-05CH11231. Stephanie Forrest acknowledges partial support of DARPA (P-1070-113237), NSF (EF1038682,SHF0905236), and AFOSR (Fa9550-07-1-0532).

References

[1] A. Barabási and R. Albert, "Emergence of Scaling in Random Networks," *Science*, vol. 286, no. 5439, 1999.

[2] S. Zhou and R. J. Mondragón, "The rich-club phenomenon in the internet topology," *CoRR*, vol. cs.NI/0308036, 2003.

[3] A. Clauset, C. Rohilla Shalizi, and M. E. J. Newman, "Power-law distributions in empirical data," *ArXiv e-prints*, Jun. 2007.

[4] P. Holme, J. Karlin, and S. Forrest, "An integrated model of traffic, geography and economy in the internet," *CoRR*, vol. abs/0802.3283, 2008.

[5] M. Piraveenan, M. Prokopenko, and A. Y. Zomaya, "Local assortativity and growth of internet," *The European Physical J. B*, vol. 70, 2009.

[6] C. Song, S. Havlin, and H. A. Makse, "Self-similarity of complex networks," *Nature*, vol. 433, Jan. 2005.

[7] L. da F. Costa, F. A. Rodrigues, G. Travieso, and P. R. V. Boas, "Characterization of complex networks: A survey of measurements," *Advances in Physics*, vol. 56, 2007.

[8] P. Mahadevan, D. Krioukov, M. Fomenkov, X. Dimitropoulos, k. c. claffy, and A. Vahdat, "The internet as-level topology: three data sources and one definitive metric," *SIGCOMM Comput. Commun. Rev.*, vol. 36, Jan. 2006.

[9] H. Haddadi, D. Fay, A. Jamakovic, O. Maennel, A. W. Moore, R. Mortier, M. Rio, and S. Uhlig, "Beyond node degree: Evaluating as topology models," *CoRR*, vol. abs/0807.2023, 2008.

[10] S. Hofmeyr, T. Moore, S. Forrest, B. Edwards, and G. Stelle, "Modeling internet-scale policies for cleaning up malware," in *Proc. WEIS*, 2011.

[11] J. C. Doyle, D. L. Alderson, L. Li, S. Low, M. Roughan, S. Shalunov, R. Tanaka, and W. Willinger, "The "robust yet fragile" nature of the Internet," *PNAS*, vol. 102, no. 41, Oct. 2005.

[12] L. C. Freeman, "A set of measures of centrality based on betweenness," *Sociometry*, vol. 40, no. 1, 1977.

[13] S. Brin and L. Page, "The anatomy of a large-scale hypertextual web search engine," in *Seventh International World-Wide Web Conf.*, 1998.

[14] G. Sabidussi, "The centrality index of a graph," *Psychometrika*, vol. 31, no. 4, Dec. 1966.

[15] J. Leskovec, J. Kleinberg, and C. Faloutsos, "Graph evolution: Densification and shrinking diameters," *ACM Trans. Knowl. Discov. Data*, vol. 1, Mar. 2007.

[16] L. C. Freeman, "Finding social groups: A meta-analysis of the southern women data," in *Dynamic Social Network Modeling and Analysis. The National Academies*, 2003.

[17] S. Fortunato, "Community detection in graphs," *Physics Reports*, vol. 486, no. 3-5, 2010.

[18] D. J. Watts and S. H. Strogatz, "Collective dynamics of 'small-world' networks." *Nature*, vol. 393, no. 6684, 1998.

[19] M. E. J. Newman, "Mixing patterns in networks," *Phys. Rev. E*, vol. 67, 2003.

[20] S. B. Seidman, "Network structure and minimum degree," *Social Networks*, vol. 5, no. 3, 1983.

[21] L. Li, D. Alderson, J. Doyle, and W. Willinger, "Towards a theory of scale-free graphs: Definition, properties, and implications," *Internet Mathematics*, vol. 2, no. 4, 2005.

[22] M. Roughan, S. J. Tuke, and O. Maennel, "Bigfoot, sasquatch, the yeti and other missing links: what we don't know about the as graph," in *Proc. ACM SIGCOMM IMC*, 2008.

[23] K. Claffy, "Border gateway protocol (bgp) and traceroute data workshop report," Cooperative Association for Internet Data Analysis (CAIDA), Tech. Rep., Oct 2011.

[24] R. Cohen, "The internet dark matter: on the missing links in the as connectivity map," in *in Proc. IEEE INFOCOM*, 2006.

[25] Y. He, G. Siganos, M. Faloutsos, and S. Krishnamurthy, "Lord of the links: a framework for discovering missing links in the internet topology," *IEEE/ACM Trans. Netw.*, vol. 17, pp. 391–404, April 2009.

[26] T. W. Anderson, "On the distribution of the two sample cramer von mises criterion," *Ann. Math. Stat.*, vol. 33, pp. 1148–1159, 1962.

[27] J. Lin, "Divergence measures based on the shannon entropy," *IEEE Transactions on Information theory*, vol. 37, 1991.

[28] A. Kolmogoroff, "Confidence limits for an unknown distribution function," *The Annals of Mathematical Statistics*, vol. 12, no. 4, 1941.

[29] D. T. L. C. G. Rogers, *Numeric methods in finance*. Cambridge University Press, 1997.

[30] A. Fabrikant, E. Koutsoupias, and C. Papadimitriou, "Heuristically optimized trade-offs: A new paradigm for power laws in the internet," in *Automata, Languages and Programming*, 2002, vol. 2380.

[31] T. Bu and D. Towsley, "On distinguishing between internet power law topology generators," in *Proc. INFOCOM*, vol. 2, 2002.

[32] H. Chang, S. Jamin, and W. Willinger, "Internet connectivity at the AS-level: an optimization driven modeling approach," in *Proc. ACM SIGCOMM MoMeTools Workshop*, 2003.

[33] S. Zhou and R. J. Mondragón, "Accurately modeling the internet topology," *Phys. Rev. E*, vol. 70, no. 6, Dec 2004.

[34] H. Chang, S. Jamin, and W. Willinger, "To peer or not to peer: Modeling the evolution of the Internet's AS-level topology," in *Proc. INFOCOM*, 2006.

[35] R. Oliveira, B. Zhang, and L. Zhang, "Observing the evolution of internet as topology," in *University of Cambridge, Computer Laboratory. His*, 2006.

[36] P. Gill, M. Arlitt, Z. Li, and A. Mahanti, "The flattening internet topology: natural evolution, unsightly barnacles or contrived collapse?" in *Proc. PAM*, 2008.

[37] C. Labovitz, S. Iekel-Johnson, D. McPherson, J. Oberheide, and F. Jahanian, "Internet inter-domain traffic," in *Proceedings of the ACM SIGCOMM 2010 conference on SIGCOMM*, ser. SIGCOMM '10. New York, NY, USA: ACM, 2010, pp. 75–86.

[38] A. Dhamdhere and C. Dovrolis, "Ten years in the evolution of the internet ecosystem," in *Proc. IMC*, 2008.

[39] M. Gaertler, "Dynamic analysis of the autonomous system graph," in *Proc. IPS*, 2004.

[40] G.-Q. Zhang, G.-Q. Zhang, Q.-F. Yang, S.-Q. Cheng, and T. Zhou, "Evolution of the internet and its cores," *New J. of Physics*, vol. 10, no. 12, 2008.

Progress in Spoken Programming

Benjamin M. Gordon
George F. Luger
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.

An introduction to this research was presented at the UNM CS Student Conference in 2011 [8]. In this paper, we describe progress and lessons learned so far.

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. Products such as Apple's Siri have demonstrated the usefulness of systems built around non-keyboard inputs. In this research, we 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 [2,3,5], 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

This research has been previously described in [8] and [9].

In terms of other research, 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 [11].

More recently, Désilets, Fox and Norton created VoiceCode [5] at the National Research Center in Canada. Begel and Graham studied how programmers verbalized code [3]. 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 [3, 4].

Arnold, Mark and Goldthwaite proposed a system called VocalProgramming [2]. 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 [12]. 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 Progress

This research project is made of two components: The programming language itself, and the development environment.

3.1 Spoken Programming Language

The language is a simple imperative language with English-like syntax for the supported constructs. It supports simple loops, conditionals, functions, and variables of a few basic data types. The syntax and supported features are more completely described in [10].

A mostly-complete compiler has been created that takes the textual form of the language as input and produces programs that run on the Java JVM as output. It uses ANTLR [1] as the parser engine and produces Java as an intermediate language. All of the major constructs of the

language are implemented (assignments, loops, function calls, etc), and the compiler is capable of compiling a large number of programs that solve real (though small) classic problems in computer science.

As originally anticipated, the syntax appears extremely verbose compared to traditional programming languages, but experience has shown that the extra verbosity is largely mitigated by the rhythms of comfortable English speech in practice. As an example, here is a tiny, "Hello, World"-style program that prints the number 42 and exits:

```
define function main taking no arguments as
    print 42
    return 0
end function
```

As a slightly larger example, this program prints $n!$ for n ranging from 0 through 9 using a naive recursive factorial implementation:

```
define function factorial taking arguments N as
    if N < 2 then
        return 1
    else
        set X to N - 1
        set Z to the result of calling factorial with X
        return N * Z
    end if
end function

define function main taking no arguments as
    set I to 0
    while I < 10 do
        set Y to the result of calling factorial with I
        print Y
        new line
        set I to I + 1
    end while
    return 0
end function
```

The primary component missing from the proposed compiler features is type inference. Implementation of this feature is underway; currently, the compiler infers that every variable is an integer (notice that the examples above are compilable under this restriction).

3.2 Programming Environment

Unlike a traditional text-based programming language, a spoken programming language isn't much good without an environment to support entry and editing. The environment for this programming language is built as a plugin for the Eclipse IDE [6], using CMU Sphinx [13] as the voice recognition component.

The Eclipse plugin currently allows basic dictation of program text. It does not yet support editing or debugging commands. Additionally, due to the way Sphinx handles grammar-based recognition, the entire program must be spoken without a pause for it to be successfully recognized. The "print 42" program above can be dictated, but the second example is too long for normal humans to get through without needing a breath. Correcting this deficiency is the current primary focus of development in this area.

4 Future Work

The compiler and Eclipse plugin are expected to be finished in summer of 2012. After some initial sanity testing and feedback from early users, the effectiveness of the complete system will be evaluated through a user study. We anticipate beginning this study in fall of 2012.

5 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. The creation of a new programming language and an associated environment for voice input was proposed in spring 2011. Implementation of this idea is proceeding apace and will soon be ready for testing. Upon completion of the editing environment, we 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] ANTLR Parser Generator. `http://www.antlr.org/`, Retrieved April 13, 2012.

[2] 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.

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

[4] 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.

[5] 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.

[6] Eclipse: The eclipse foundation open source community website. `http://www.eclipse.org/`, Retrieved January 10, 2011.

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

[8] Benjamin M. Gordon. Developing a Language for Spoken Programming. In *UNM Computer Science Student Conference*, pages 3–10, `http://www.cs.unm.edu/~csgsa/unm-cs-conf7.pdf`, April 2011.

[9] Benjamin M. Gordon. Developing a Language for Spoken Programming. In *Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence*, pages 1847–1848, August 2011.

[10] Benjamin M. Gordon. Developing a Language for Spoken Programming (Dissertation Proposal). `http://www.cs.unm.edu/~bmgordon/proposal-bmg.pdf`, May 2011.

[11] 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.

[12] 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.

[13] CMU sphinx - speech recognition toolkit. `http://cmusphinx.sourceforge.net/`, Retrieved January 15, 2011.

Enriching Chatter Bots With Semantic Conversation Control

Chayan Chakrabarti
Department of Computer Science, University of New Mexico, Albuquerque, NM 87131
cc@cs.unm.edu

Abstract

Businesses deploy chatter bots to engage in text-based conversations with customers that are intended resolve their issues. However, these chatter bots are only effective in exchanges consisting of question-answer pairs, where the context may switch with every pair. I am designing a semantic architecture that enables chatter bots to hold short conversations, where context is maintained throughout the exchange. I leverage specific ideas from conversation theory, speech acts theory, and knowledge representation. My architecture models a conversation as a stochastic process that flows through a set of states. The main contribution of this work is that it analyses and models the semantics of conversations as entities, instead of lower level grammatical and linguistics forms. I evaluate the performance of the architecture in accordance with Grice's cooperative maxims, which form the central idea in the theory of pragmatics.

Motivation and Problem Statement

Communicating with chatter bots has come a long way, from pioneering AI demonstrations like ELIZA, to modern software like Siri. Many businesses also realize their customer service operations using chatter bots as virtual representatives (www.goarmy.com, www.alaskaair.com). But communication with these chatter bots takes the form of successive question-answer pairs, where the context may switch with every pair. The main goal of this work is to design an integrated architecture that enables the chatter bot go beyond mere question-answer exchanges, to hold a short conversation, where the context is maintained throughout the exchange. The domain is restricted to customer service situations through text-based chat. The chatter bot answers FAQ type questions, resolves customer service issues, spots opportunities during the conversation to disseminate unsolicited information (information about related services and promotions), and evaluates the semantic flow of the conversation. If the flow of the conversation requires the chatter bot to pursue a course of action beyond its programmatic capabilities, it realizes this and transfers the conversation to a human representative.

Conversation theory defines a formal framework for shared construction of knowledge between multiple conversationalists (Pask 1976). A conversation is a process that flows through a set of states (Winograd and Flores 1986). The set of states represents a speech act (GoldKuhls 2003). A conversation can have several semantic states (Ginsburg 2008). The states are proxy indicators of customer satisfaction (Stolcke et al. 1998). Sentiment detection (Pang and Lee 2008) is also another indicator that influences conversations. Hence customer satisfaction can be measured by observing the flow of a conversation through various states (Twitchell et al. 2004).

While the proposed research borrows ideas from other works, it solves a distinct problem. I am not modeling the low level abstractions of sentences, phrases, and words, or linguistic artifacts of grammar, discourse resolution, and parts of speech. I consider conversations to be the unit of analysis. Each conversation segment is a data point. I am analyzing and modeling the conversation itself, and not the lower level grammatical minutia that form individual components of the conversation. Furthermore, I am exploring well-structured conversations in a fixed domain.

Semantic Conversation Architecture

The key aspects of holding a conversation are *what to say*, and *how to say* it, which are handled by the Knowledge Engine (KE), and the Conversation Engine (CE) in the chatter bot architecture. The Chat Interface (CI) directly interfaces with the user, and has modules for pre-processing the raw text input, identifying the Speech Act, Sentiment, and Topic associated with the utterance, and passing this information to the CE and KE. See Figure 1.

The KE contains a Speech Acts Hash Set (SAHS) and a Topic Hash Set (THS). The SAHS is a data structure that

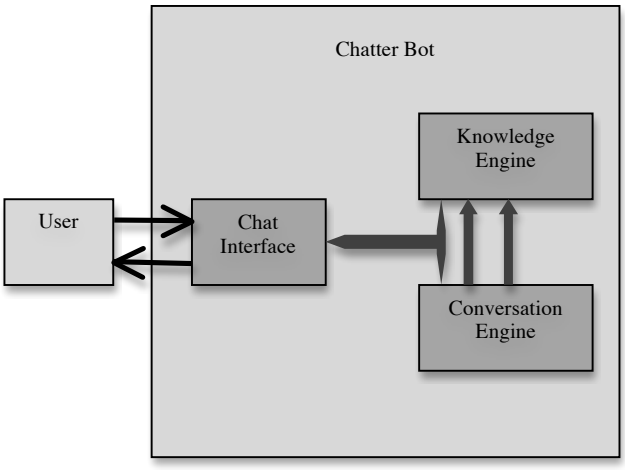


Figure 1: Overview of Chatter Bot Architecture.

stores the Speech Acts in the form of probabilistic finite state automata. The probabilities are learned from a corpus

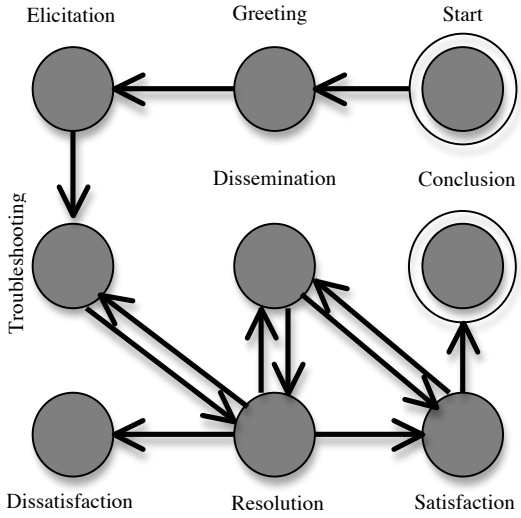


Figure 2: Conversation Speech Act represented by a probabilistic finite state automaton.

of conversations. The THS is a data structure containing specific information about the domain. The information is organized in the form of Goal-Fulfillment Maps (O'Shea et al. 2010) and ontologies.

The CE contains the probabilistic finite state representation of the conversation Speech Act. One such example, shown in Figure 2, consists of 8 states: *Greeting* (includes small talk), *Elicitation* (trying to understand issue), *Troubleshooting (working through steps to understand the issue)*, *Resolution*, *Dissatisfaction*, *Dissemination* (give unsolicited information such as possible promotions), and *Conclusion*. The CE also has a conversation planner (CP) containing a simple decision tree that engineers the conversation towards satisfactory states, guided by the transition matrix of the automaton.

Hypotheses, Datasets, and Validation

There are two specific hypotheses. First, the knowledge representation framework will have sufficient representation power to model domain knowledge. Second, the semantic conversation control algorithm will be able to detect transitions in conversation states, predict outcomes, and use this knowledge to control the conversation.

I am using four corpora of logged chat sessions. Each corpus has several million lines of chat, split in to distinct chat sessions. Two contain chat logs from human-led sessions, and two contain chat logs from chatter bot led sessions in the same domain.

The performance of my chatter bot is evaluated according to Grice's maxims of cooperation (Grice 1989): *quality* (bot gives correct information), *quantity* (bot is informative as needed), *relation* (conversation is relevant to discussion), and *manner* (conversation is unambiguous).

Current and Future Work

I had an internship at NextIT in Spokane, WA, where I had access to real-world corpora for testing my approach. I have implemented a prototype of the CE. Future work includes implementing the KE, the CI, and experimenting with conversation strategies in the CP.

References

Ginzburg, J. 2008. *Semantics for Conversation*. King's College, London. CSLI Publications.

GoldKuhl, G. 2003. *Conversation Analysis as a Theoretical Foundation for Language Action Approaches?* 8th Int'l Working Conf. on the Lang. Action Perspective on Comm. Modeling.

Grice, P. 1989. *Studies in the Way of Words*. Harvard Univ. Press.

O'Shea, K., Bandar, Z., and Crockett, K. 2010. *A Conversational Agent Framework using Semantic Analysis*. Int'l Journal of Intelligent Computing Research. Vol 1, Issue 2, June 2010.

Pang, B., and Lee, L. 2008. *Opinion Mining and Sentiment Analysis*. Foundations and Trends in Information Retrieval. Vol. 2, Nos 12 (2008) 1135.

Pask, G. 1976. *Conversation Theory*. Elsevier Press.

Stolcke, A., Shriberg, E., Bates, R., Coccaro, N., Jurafsky, D., Martin, R., Meteer, M., Ries, K., Taylor, P., and EssDykema, C. 1998. *Dialog Act Modeling for Conversational Speech*. AAAI 98.

Twitchell, D., Adkins, M., Nunmaker, J., and Burgoon, J. 2004. *Using Speech Act Theory to Model Conversations for Automated Classification and Retrieval*. 9th Int'l Working Conf. on the Lang. Action Perspective on Comm. Modeling.

Winograd, T., and Flores, F. 1986. *Understanding Computers and Cognition. A New Foundation for Design*. Norwood, NJ. Ablex Publishing Corp.

On the Viability of Compression for Reducing the Overheads of Checkpoint/Restart-based Fault Tolerance

Dewan Ibtesham, Dorian Arnold, and Patrick G. Bridges
Department Of Computer Science
The University of New Mexico
Albuquerque, NM 87131
{*dewan,darnold,bridges*}*@cs.unm.edu*

Kurt B. Ferreira and Ron Brightwell
Scalable System Software Department
Sandia National Laboratories
Albuquerque, NM 87185–1319
{*kbferre,rbbrigh*}*@sandia.gov*

***Abstract*—The increasing size and complexity of high performance computing (HPC) systems have lead to major concerns over fault frequencies and the mechanisms necessary to tolerate these faults. Previous studies have shown that state-of-the-field checkpoint/restart mechanisms will not scale sufficiently for future generation systems. Therefore, optimizations that reduce checkpoint overheads are necessary to keep checkpoint/restart mechanisms effective. In this work, we demonstrate that checkpoint data compression is a feasible mechanism for reducing checkpoint commit latency and storage overheads. Leveraging a simple model for *checkpoint compression viability*, we show: (1) checkpoint data compression is feasible for many types of scientific applications expected to run on extreme scale systems; (2) checkpoint compression viability scales with checkpoint size; (3) user-level versus system-level checkpoints bears little impact on checkpoint compression viability; and (4) checkpoint compression viability scales with application process count. Lastly, we describe the impact checkpoint compression might have on projected extreme scale systems.**

***Keywords*-Fault tolerance; Checkpoint Compression;**

I. Introduction

High-performance computing (HPC) systems have been increasing dramatically in size, and this trend is expected to continue. On the current Top 500 list [1], 300 (or 60%) of the 500 entries have greater than 8,192 cores, compared to 17 (or 3.4%) just 5 years ago. Also on this list, four of the systems have more than 200K cores; an additional seven have more than 128K cores, and another 10 have more than 64K cores. This year, the Lawrence Livermore National Laboratory schedules to deploy its 1.6 million core system, Sequoia [2], and future extreme scale systems are projected to have on the order of tens to hundreds of millions of cores by 2020 [3].

Future high-end systems are also expected to increase in complexity; for example, heterogeneous systems like CPU/GPU-based systems are expected to become much more prominent. We expect increased system sizes along with this increased complexity to yield extremely low mean times between failures (MTBF). Recent studies indeed show that system failure rates depend on the numbers of processor chips and that system MTBF for the biggest systems on the Top 500 lists are expected to fall below 10 minutes in the next few years [4]

In HPC, checkpoint/restart is perhaps the most commonly employed fault-tolerance mechanism for applications. Yet, as we describe in Section II, increasing checkpoint/restart overheads coupled with higher failure frequencies threaten to make checkpoint/restart infeasible for future systems. Compressing checkpoint data is, perhaps, an obvious strategy for improving checkpoint/restart efficiency, but compression is not viable if its benefits do not outweigh its costs. Alas, the majority of applications and systems that use checkpoint/restart do not compress checkpoint data. In this work, we demonstrate that given the current and increasing gap between processing and data transfer capabilities, checkpoint compression can be a very effective strategy for improving both the time and space efficiency of checkpoint/restart-based fault tolerance. We use a combination of *mini-applications* or *mini apps* [5] and a representative scientific application, LAMMPS [6] along with the Berkeley Lab Checkpoint/Restart (BLCR) framework [7] and a set of off-the-shelf compression utilities to study the viability of checkpoint compression. In this paper, we present the result of this study and make the following contributions:

- We offer a viability model for checkpoint data compression that accounts for the cost and benefits of compression for both checkpoint commit and recovery operations;
- We show that checkpoint data compression can be a very effective strategy for reducing checkpoint and restart latencies;

- We show what compression algorithms are best suited for checkpoint data compression;
- We show that application scale, in terms of memory footprint size, process counts, or run time can bear little impact on the effectiveness of checkpoint data compression;
- We show that checkpoint data compression can be effective for both application-level and system-level checkpoints;
- We show that checkpoint data compression can improve significantly an application's makespan, the application's time to solution in the presence of failures; and
- We offer a discussion of checkpoint data compression given current high performance processor and I/O technologies and trends.

The organization of this paper is as follows: in the next section, we give a background of the checkpoint/restart mechanism and a survey of related checkpoint compression work. In Section III, we present our checkpoint compression viability model and describe how we can use it to model both coordinated and uncoordinated distributed checkpointing protocols. In Section IV, we describe the applications, compression algorithms and the checkpoint library that comprise our evaluation framework as well as our experimental results. We conclude with a discussion of the implications of our experimental results for future checkpoint/restart research, development and deployment.

II. Background and Related Work

During normal operation, *checkpoint/restart* (or *rollback recovery*) protocols [8], periodically record process state to stable storage devices, devices that survive tolerated failures. Process state comprises all the state necessary to run a process correctly including its address space or memory footprint and register states. When a process fails, a new incarnation of the failed process is *recovered* from the intermediate state of the failed process' most recent checkpoint – thereby reducing the amount of lost computation. Checkpoint/restart is a well studied, general fault tolerance mechanism. However, recent studies [4], [9], [10] predict poor utilizations (approaching 0%) for applications running on imminent systems and the need for dedicated reliability resources.

A. Checkpoint Optimizations

Focusing on the checkpoint on the *checkpoint commit* problem, saving a checkpoint to stable storage, we can consider two sets of checkpoint optimization strategies. The first set of strategies hide or reduce (perceived) commit latencies without actually reducing the amount of data to commit. These strategies include *concurrent checkpointing* [11], [12], *diskless* or *multi-level checkpointing* [13]–[15], *remote checkpointing* [16], [17] and checkpointing filesystems [18]. The second set of strategies reduce commit latencies by reducing checkpoint sizes. These strategies include *memory exclusion* [19] and *incremental checkpointing* [20]–[22]. In Section V, we discuss the potential interplay between these optimizations and checkpoint compression.

B. Related Compression Research

Li and Fuchs implemented a compiler-based checkpointing approach, which exploited compile time information to compress checkpoints [23]. They concluded from their results that a compression factor of over 100% was necessary to achieve any significant benefit due to high compression latencies. Plank and Li proposed in-memory compression and showed that, for their computational platform, compression was beneficial if a compression factor greater than 19.3% could be achieved [24]. In a related vein, Plank et al also proposed *differential compression* to reduce checkpoint sizes for incremental checkpoints [25]. Moshovos and Kostopoulos used hardware-based compressors to improve checkpoint compression ratios [26]. Finally, in a related but different context, Lee et al study compression for data migration in scientific applications [27].

This work focuses on the use of software-based compressors for checkpoint compression. Given recent advances in processor technologies, we demonstrate that since processing speeds have increased at a faster rate than disk and network bandwidth, data compression can allow us to trade faster CPU workloads for slower disk and network bandwidth.

III. A Checkpoint Compression Viability Model

Intuitively, checkpoint compression is a viable technique for improving the performance of checkpoint/restart protocols when the benefits of checkpoint data reduction outweigh the costs of reducing the checkpoint data. Our viability model is inspired by the concept offered by Plank et al [24]. Plank et al focused solely on the impact of compression for the checkpoint commit phase. Our model addresses the cost and benefits of compression for both checkpoint and recovery phases. In Section IV-E, we use the results from this model to determine the overall impact of checkpoint compression on application performance.

We assume that individual processes of a distributed application are checkpointed in a coordinated fashion: all processes coordinate at the end of each checkpoint

interval to checkpoint a globally consistent application state comprised of one checkpoint per process. This is the commonly employed strategy in the HPC domain. We also assume that there are an equal number of checkpoint and recovery operations. Our justification for this latter assumption follows: optimally, an application takes a single checkpoint before each failure – upon failure, only the most recent checkpoint is used, therefore, other checkpoints are not useful. The application only needs to recover once per failure. Therefore, in the optimal case, the number of checkpoints equals the number of failures, which equals the number of recoveries. There are various works that define optimal checkpoint intervals [28], [29]. Finally, we assume that checkpoint commit is synchronous; that is, the primary application process is paused during the commit operation and is not resumed until checkpoint commit is complete.

Checkpoint compression is viable when the time to compress and write or commit a checkpoint and the time to read and decompress that checkpoint is less than the time to commit and read the uncompressed checkpoint. Assuming the times to read and write are the same:

$$t_{comp} + 2t_{cc} + t_{decomp} < 2t_{uc}$$

where t_{comp} is *compression latency*, t_{decomp} is *decompression latency*, t_{cc} is the *time to read or write the compressed checkpoint* and t_{uc} is the *time to read or write the uncompressed checkpoint*. This expression can be rewritten as:

$$\frac{c}{r_{comp}} + (2 \times \frac{(1-\alpha) \times c}{r_{commit}}) + \frac{c}{r_{decomp}} < 2 \times \frac{c}{r_{commit}}$$

where c is the size of the original checkpoint, *compression factor* α is the percentage reduction due to data compression, r_{comp} is *compression speed* or the rate of data compression, r_{decomp} is *decompression speed*, and $rcommit$ is *commit speed* or the rate of checkpoint commit or reading (including all associated overheads). The last equation can be reduced to:

$$\frac{2\alpha \times r_{comp} \times r_{decomp}}{r_{comp} + r_{decomp}} < r_{commit} \quad (1)$$

Equation 1 defines the minimal ratio between checkpoint commit rate and compression rate, decompression rate and compression factor in order for the overall time savings of checkpoint compression to outweigh its costs. Of course, checkpoint compression has the additional benefit of saving storage space, but we do not factor that into our model.

IV. An Evaluation of Checkpoint Compression

In this study, we seek to answer several fundamental questions regarding checkpoint data compression:

- *Can benefit of compressed checkpoints outweigh the additional latencies necessary to compress and decompress the checkpoint?*
- *Do the time or space scales of an application impact checkpoint data compression?*
- *Does the viability of checkpoint data compression change for application-level versus system-level checkpoints?*; and ultimately
- *What real impact can checkpoint compression have on the execution time of an application?*

We now describe the applications, tools and experiments we use to answer these questions and discuss the conclusions we have made based on our experimental and modeling results.

For all but our scaling experiments, we used a 64-bit, four core Intel Xeon processor with a 2.33 GHz clock cycle rate and 2 GB of memory. For our scaling study, we collected checkpoints from application runs on a Cray XT5 series machine. However, for uniformity and ease of access to the compression utilities, these checkpoints also were compressed/decompressed on our four core workstation.

A. Evaluation Tool Chain

We used a range of applications, libraries and utilities in this study. In this section, we describe these various components.

1) The Mini Applications: We chose four *mini-applications* or *mini apps* from the Mantevo Project [5], namely HPCCG version 0.5, miniFE version 1.0, pHPCCG version 0.4 and phdMesh version 0.1. The first three are implicit finite element mini apps and phdMesh is an explicit finite element mini app. HPCCG is a conjugate gradient benchmark code for a 3D chimney domain that can run on an arbitrary number of processors. This code generates a 27-point finite difference matrix with a user-prescribed sub-block size on each processor. miniFE mimics the finite element generation assembly and solution for an unstructured grid problem. pHPCCG is related to HPCCG, but has features for arbitrary scalar and integer data types, as well as different sparse matrix data structures. PhdMesh is a full-featured, parallel, heterogeneous, dynamic, unstructured mesh library for evaluating the performance of operations like dynamic load balancing, geometric proximity search or parallel synchronization for element-by-element operations.

Mini apps are small, self-contained programs that embody essential performance characteristics of key applications. While the Mantevo mini apps are not (yet) as popular as other HPC benchmarks, like the NAS Parallel Benchmarks or the HPC Challenge Benchmark, we believe the mini apps are much better suited for this study. HPC benchmarks generally target the evaluation of computer system performance. On the other hand, the mini apps are meant to be lightweight application proxies for the heavyweight counterparts. In other words, the mini apps are intended to mimic real application characteristics including the memory footprint properties relevant to this checkpoint compression study.

2) A Full Application: LAAMPS: We use LAMMPS (the Large-scale Atomic/Molecular Massively Parallel Simulator) to evaluate checkpoint compression on a full featured scientific application. LAMMPS [6], [30] is a classical molecular dynamics code developed at Sandia National Laboratories. LAMMPS is a key simulation workload for the U.S. Department of Energy and is representative of many other molecular dynamics code. In addition, LAMMPS has built-in checkpointing support that allows us to compare generic, system-based mechanisms with an application specific mechanism. For our experiments, we used the embedded atom method (EAM) metallic solid input script, which is used by the Sequoia benchmark suite.

3) Compression Utilities: For this study, we focused on the popular compression algorithms investigated in Morse's comparison of compression tools [31]. We do not present results from some algorithms that did not perform well. Additionally, some algorithms can be parameterized to trade between execution time for compression factor. We only present the parameter set that represents the best trade-off.

- **zip**: `zip` is an implementation of Deflate [32], a lossless data compression algorithm that uses the LZ77 [33] compression algorithm and Huffman coding. It is highly optimized in terms of both speed and compression efficiency. The `zip` algorithm treats all types of data as a continuous stream of bytes. Within this stream, duplicate strings are matched and replaced with pointers followed by replacing symbols with new, weighted symbols based on frequency of use.

  `zip` takes an integer parameter that ranges from zero to nine, where zero means fastest compression speed and nine means best compression factor. For our experiments, "`zip(1)`" represents the best trade-off.
- **7zip** [34]: `7zip` is based on the Lempel-Ziv-Markov chain algorithm (LZMA) [35]. It uses a dictionary scheme similar to LZ77.
- **bzip2**: `bzip2` is an implementation of the Burrows-Wheeler transform [36], which utilizes a technique called block-sorting to permute the sequence of bytes to an order that is easier to compress. The algorithm converts frequently-recurring character sequences into strings of identical letters and then applies move to front transform and Huffman coding.

  In bzip2, compression performance varies with block size. `bzip2` takes an integer parameter that ranges from zero to nine, where a smaller value specifies a smaller block size. For our experiments, "`bzip2(1)`" represents the best trade-off.
- **pbzip2** [36]: `pbzip2` is a parallel implementation of `bzip2`. `pbzip2` is multi-threaded and, therefore, can leverage multiple processing cores to improve compression latency. The input file to be compressed is partitioned into multiple files that can be compressed concurrently.

  `pbzip2` takes two parameters. The first parameter is the same block size parameter as in `bzip2`. The second parameter defines the file block size into which the original input file is partitioned. For our experiments, "`pbzip2(1,5)`" represents the best trade-off.
- **rzip**: `rzip` uses a very large buffer to take advantage of redundancies that span very long distances. It finds and encodes large chunk of duplicate data and then uses `bzip2` as a backend to compress the encoding.

  Similar to `zip`, `rzip` takes an integer parameter that ranges from zero to nine, where zero means fastest compression speed and nine means best compression factor. For our experiments, "`rzip(3)`" represents the best trade-off.

4) Checkpoint/Restart Utilities: The Berkeley Lab Checkpoint/Restart library (BLCR) [7], a system-level infrastructure for checkpoint/restart, is an open source checkpoint/restart library and is deployed on several HPC systems. For most of our experiments, excluding some application specific checkpoints taken with LAMMPS, we obtained checkpoints using BLCR. Furthermore, we use the OpenMPI [37] framework, which has integrated BLCR support.

For our scaling study we used a user-level checkpoint library built into LAMMPS. LAMMPS can use application-specific mechanisms to save the minimal state needed to restart its computation. More specifically, it saves each atom location and speed. The largest

data structure in the application, the neighbor structure used to calculate forces, is not saved in the checkpoint and is recalculated upon restart. This scheme reduces per-process checkpoint files to about one eighth of the applications memory footprint.

B. Evaluating Checkpoint Compression Effectiveness

We compressed and decompressed many checkpoints collected from our application suite using the different compression utilities. For each experiment, we measured the performance metrics the performance metrics necessary to determine checkpoint viability using Equation 1 from Section III, namely compression factor, compression speed and decompression speed.

For our baseline experiments, we were not concerned about scaling along either the time our space dimentions. We chose problem sizes that allowed each application to run long enough to generate 5 checkpoints. The three implicit finite element mini apps, HPCCG, pHPCCG and miniFE were given a 100x100x100 problem size. phdMesh and LAMMPS were given a 5x5x5 problem size. Each application was run using 2–3 MPI processes, except for phdMesh, which was run without MPI support. Checkpoint intervals for miniFE, pHPCCG, HPCCG and LAMMPS were 3, 3, 5 and 60 seconds, respectively. For phdMesh the 5 checkpoints were taken at simulation timestep boundaries. BLCR was used to collect all checkpoints, which ranged in size from 311 MB to 393 MB for the mini apps to about 700 MB for LAMMPS.

Figure 1 shows how effective the various algorithms are at compressing checkpoint data. We can see that all the algorithms achieve a very high *compression factor* of about 70% or higher for the *mini apps* and about 57-65% for LAMMPS, where compression factor is computed as: $1 - \frac{compressed\ size}{uncompressed\ size}$. This means, then that the primary distinguishing factor becomes the compression speed, that is, how quickly the algorithms can compress the checkpoint data.

Figures 2(a) and 2(b) show compress and decompression speeds, respectively. In general, and not surprisingly, the parallel implementation of `bzip2`, `pbzip2`, generally outperforms all the other algorithms. Decompression is a much faster operation than compression, since during the compression phase, we must search for compression opportunities, while during decompression, we simply are using a dictionary or lookup table to expand compressed items.

Based on the above results and Equation 1

$$\frac{2\alpha \times r_{comp} \times r_{decomp}}{r_{comp} + r_{decomp}} < r_{commit},$$

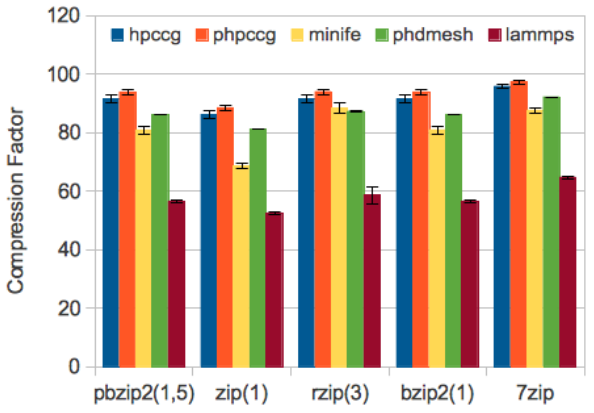


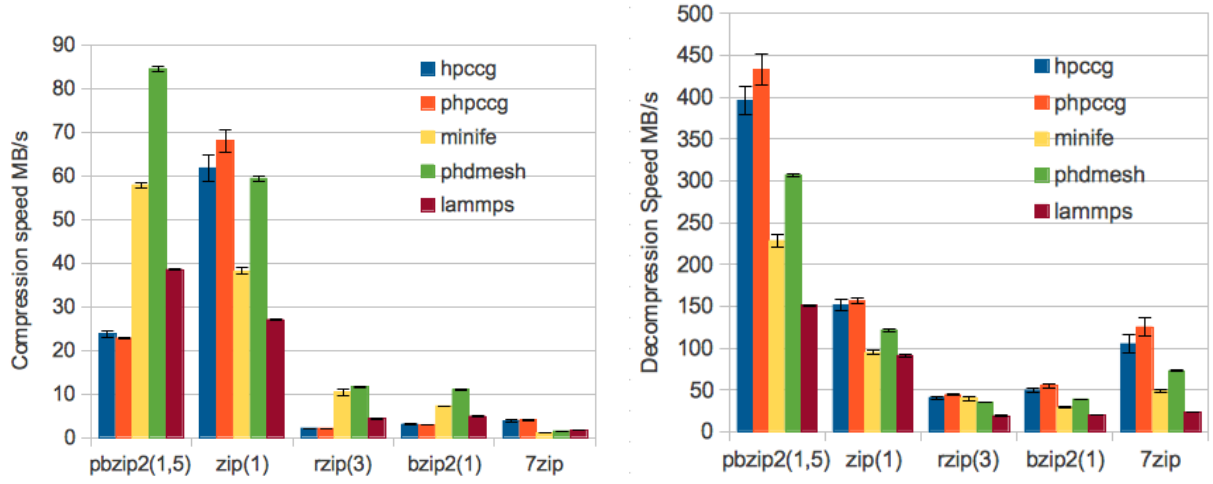
Figure 1. Checkpoint compression factors for the various algorithms and applications. Higher is better: a factor of 90% means that file size was reduced by 90%.

which represents our viability model, Figure 3 demonstrates the checkpoint read/write bandwidths that make compression viable. For each application, the highest bar of all the compression algorithms represents its worse case scenario. For the worse case application, LAMMPS, checkpoint compression is viable unless a system can sustain a per process checkpoint read/write bandwidth of greater than about three GB/s. In the best case, phdMesh, the necessary per process checkpoint read/write bandwidth raises to greater than 11 GB/s. In Section V, we describe the impact of these results in the context of extreme scale systems. The executive summary is that checkpoint compression is a very viable solution for current and projected HPC systems. (Since pbzip2 and zip performance dominate those of the other compression utilities, for the remainder of this paper, we only present results for these two algorithms.)

C. Evaluating the Impact of User versus System Level Checkpoints

Next, we examine the compression effectiveness of system-level checkpoints versus that of application specific checkpoints. We use LAMMPS for this testing due to its optimized, application specific checkpointing mechanism described in the previous section. For these tests we compare application generated restart files with those generated by BLCR. In each case, we take 5 checkpoints equally spaced throughout the application run.

System-level checkpointing saves a snapshot of the application context such that it can be restarted where it left off. Application specific checkpointing only needs to save the data needed to resume operation. As a



(a) Compression

(b) Decompression

Figure 2. Checkpoint Compression and Decompression Speeds.

result, for a fixed problem, system level checkpoints are typically much larger in size. In our tests, LAMMPS' application specific checkpoints were 170MB in size compared to about 700MB BLCR generated checkpoints. However, based on our results in Table I, we observe that checkpoint compression is viable for both application specific and system level checkpoints.

There is, however, a qualitatitve difference in the break-even points for checkpoint compression. Our data reveals that the major reason is that, system level checkpoints compressed better than user level checkpoints (for example, pbzip2 compression factors are 56.5% compared to 43.3%). Additionally, the average compression and decompression speeds were higher for system level checkpoints than for user level checkpoints (again for pbzip2, 94.8 MB/s compared to 87 MB/s).

	pbzip(1,5)	**zip(1)**
System Checkpoint	3.38 GB/s	2.13 GB/s
Application Checkpoint	2.79 GB/s	1.77 GB/s

Table I
COMPRESSION BREAK-EVEN POINTS FOR SYSTEM LEVEL AND APPLICATION SPECIFIC CHECKPOINTS.

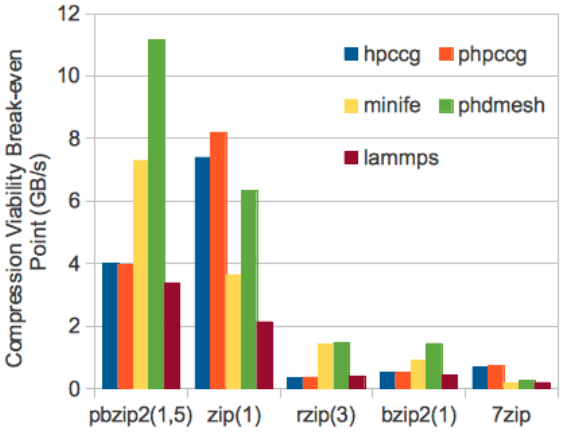
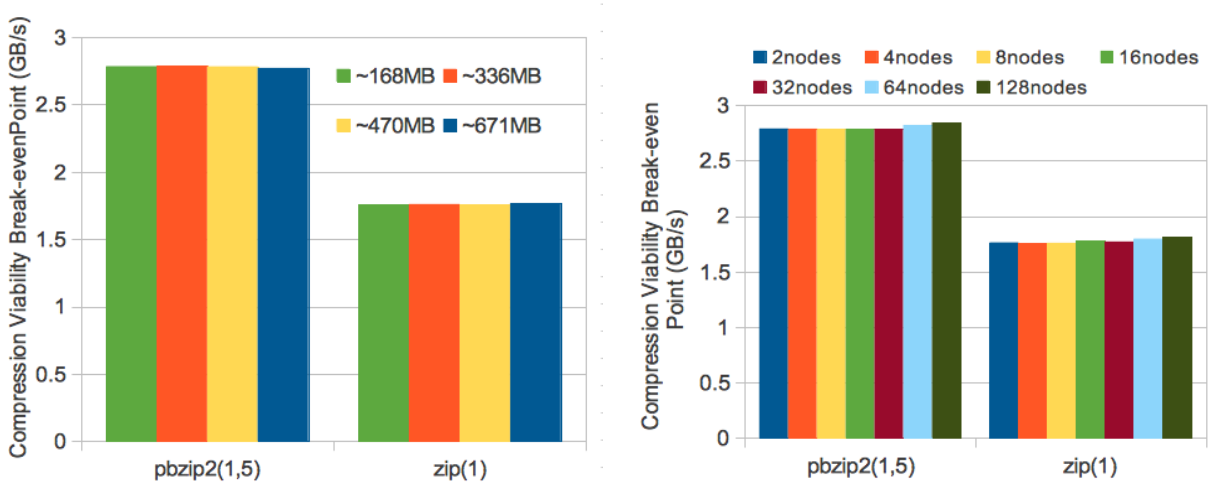


Figure 3. Checkpoint Compression Viability: Unless, checkpoint read/write bandwidth exceeds our viability factor (y-axis), checkpoint compression should be used.

D. Evaluating the Impact of Scale

For our scaling experiments, we use the LAMMPS and its its built-in checkpoint mechanism. We observe how checkpoint viability scales with (1) memory size; (2) time (between checkpoints); and (3) process counts.

In our first set of scaling experiments, we evaluate the first two scaling dimensions, checkpoint size and time between checkpoints. We progressively increased the LAMMPS problem size while keeping the number of applicaiton processes fixed at two. In this manner, memory footprint and checkpoint sizes increases. This also means that the application runs for a longer time, since the per process workload has been increased. For each LAMMPS process, five checkpoints were taken uniformly throughout the application run. The checkpoints we collected from these tests averaged about 168MB, 336MB, 470MB and 671MB for the various problem sizes.



(a) Scaling Checkpoint Sizes and Application Runtime.

(b) Scaling Process Counts.

Figure 4. Results from our Scaling Experiments.

Figure 4(a) shows the viability results from these experiments. We readily observe that in no case did checkpoint size show any impact on the viability of checkpoint compression for LAMMPS.

For the study of scaling in terms of process count, we compare the compression ratios for a weak scaling LAMMPS EAM simulation for between 2 and 128 MPI processes. In each test, the per-process restart file size is over 170 MB. In these runs we take 5 equally spaced checkpoints. Figure 4(b) shows once again that application process counts did not bear an impact on checkpoint compression viability. We have no reason to believe these results will be different for larger process count runs.

E. Performance Impact of Compression

To outline the impact of checkpoint compression on application time to solution, we created a performance model for expected time to solution for an application with checkpoint/restart. This model is based on Daly's higher order model [28], which assumes node failures are independent and exponentially distributed. The model takes as input the mean time between failures (MTBF) for the system, the checkpoint commit time, the checkpoint restart time, the number of nodes used in the application and the time the application would take to complete in a failure-free environment.

We modified this model to integrate checkpoint compression and decompression. For the checkpoint commit time we included the time to compress the checkpoint image as well as the time to write this compressed image to stable storage on the parallel system. Similarly, on restart we included the time to read the compressed checkpoint image and perform the decompression step.

In Figure 5, we show the result of this model. In this figures we show the efficiency of an application computation. This efficiency metric is defined as the time to solution in the failure environment divided by the time to solution in a failure-free environment. For this figure we use the best compression ratio and rates for each application described previously in the paper. In addition, this model assumes each node uses 2GB of memory and that $\frac{1}{3}$ of that memory is written on each checkpoint. These values are representative of what we have observed at the Sandia National Laboratory for our capability workloads. Finally, we assume a five year node MTBF as has been measured in current studies [38].

Regarding file I/O rates to stable storage, we use a report based on a study of I/O performance on Argonne National Laboratories 557 TFlop Blue Gene/P system (Intrepid) [39] to select I/O rates for our model. This work executes an I/O scaling study majoring maximum achieved throughput for carefully selected read and write patterns. From this report, the best observable per process I/O bandwidths 1 MB/s for both reading and writing. This performance scales to about 32,768 processes and then decreases. For example, at 131,072 processes, per process read bandwidth is 385 KB/s and per process write bandwidth is 328 KB/s. At any rate, for our study, we optimistically choose the best observed per process I/O bandwidth of 1 MB/s.

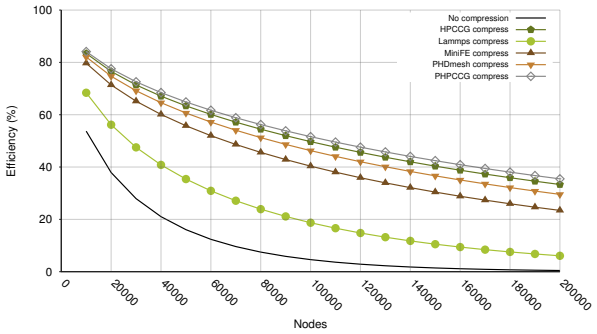


Figure 5. Impact of Checkpoint Compression on Application Efficiency.

From this figure we see that each of the compression rates measured in this work have a dramatic and positive influence on the performance of traditional checkpoint/restart at scales seen in today's systems as well as the expected scales of future systems.

V. Discussion

A. Compression versus Checkpoint I/O Bandwidth

The relationship between compression performance (compression factor and compression and decompression speeds) checkpoint I/O bandwidth is the key factor of the viability of checkpoint compression. As Figure 3 shows, for our worse case application, LAMMPS with pbzip2 compression, compression is viable if per-process checkpoint bandwidths are less than 3 GB/s. In the best case, phdMesh with pbzip2 compression, per process checkpoint bandwidths must exceed 11 GB/s. In Section IV-E, we described the best observed per process I/O bandwidths that we found in the literature, 1 MB/s. For comparison, the Oak Ridge Cray XT5 Jaguar petascale system has peak per-node and per-core checkpoint bandwidths of 5.3 MB/s and 1 MB/s, respectively, three orders of magnitude less than needed. Similarly, the Lawrence Livermore Dawn IBM BG/P system has a peak per-node checkpoint bandwidth of about 2 MB/s [1] As a result, aggressive use of checkpoint compression appears to be viable and indeed desirable on current large-scale platforms.

The performance impact of checkpoint compression on future systems depends highly on future computer storage architecture developments. One report suggested using multiple disks per processor to provide sufficient storage bandwidth for high-speed checkpointing (5 GB/sec per process) [3]. Other researchers have suggested using similar approaches that combine non-volatile memories with spinning storage to reduce the potential costs of the checkpoint file systems [42], though anticipated storage system costs are still $60M. These aggressive bandwidths are at the boundary of where checkpoint compression is viable, and so it is unclear whether or not checkpoint compression would be useful if such high-bandwidth (and expensive) I/O systems were adopted.

Checkpoint compression also reduces the bandwidth and storage pressures on checkpointing file systems. If energy consumption and cost of such storage systems are important design limiters compared to the CPU power and costs, as is expected [3], checkpoint compression could be an important technology in reducing the demands on exascale storage systems.

B. Compression versus Other Checkpoint Optimizations

We believe that checkpoint compression is completely complementary to other known checkpoint optimizations such as the ones listed in Section II. Latency saving techniques diskless, multi-level and remote checkpointing still require the transfer of data from one node to another. As such, checkpoint compression can still significantly decrease the time it takes to transfer a checkpoint. Additionally, techniques that store checkpoints in remote DRAM memories to avoid disk latencies benefit from reduced checkpoint sizes. Particularly for capability class applications, which are resource intensive and often memory-bound, checkpoint compression would reduce storage pressures on the memory system.

Optimization strategies that also aim at reducing checkpoint sizes, like memory exclusion and incremental checkpointing, can benefit further from data compression. For example, if applications employing these optimizations have similar "memory footprint features" as the applications in this study, the portions of the application processes' address space that still need to be checkpointed would demonstrate the same compression viability features as in this study.

C. Compression Viability Model Assumptions

In this study, we assumed coordinated distributed checkpoints, In actuality, for our viability model, it does not really matter how many processes are checkpointing simultaneously. Our model specifies the minimum per process (or per checkpoint) bandwidth render compression useless. This is the case whether all processes are actively checkpointing simultaneously or some subset thereof (including singleton sets).

[1] Oak Ridge's Spider Lustre-based file system provides 240 GB/sec of aggregate bandwidth [40], while Dawn's Lustre file system is listed as providing 70 GB/sec of peak bandwidth on LLNLL reference pages [41].

We also assumed an equal number of checkpoints and recoveries. The equations in Section III can easily be modified to accommodate a disproportionate number of checkpoints to recoveries. Lastly, we assumed a synchronous checkpoint commit mechanism in which the target process is preempted until the checkpoint has been written to stable storage. As such, we can safely assume that the application is interupted completely for the entire checkpointing operation. If checkpoints could be committed asynchronously, our model would have to account for the reduced **perceived** latency and, therefore, reduced impact of checkpoint commit on application performance.

D. Future Enhancements

Our results show that different compression algorithms exhibit different performance on different application checkpoints. We would like to understand what aspects and features of the checkpoint data impact compression algorithm performance. This would help us predict the optimal compression algorithm for a particular application as well as give us insights into how we might improve an algorithms performance.

The positive results from standard, off-the-shelf compression utilities suggests that we might be able to yield even better results with some customizations. As suggested above, a detailed study what makes a good or bad compression algorithm for checkpoint compression could lead to optimization opportunies. Another promising avenue, which we are now exploring, is the use of GPUs for accelerating compression speeds.

Acknowledgments

This work was supported in part by Sandia National Laboratories subcontract 438290. Sandia National Laboratories is a multiprogram laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000. The authors are grateful to the members of the Scalable Systems Laboratory at the University of New Mexico and the Scalable System Software Group at the Sandia National Laboratory for helpful feedback on portions of this study.

REFERENCES

[1] "Top 500 Supercomputer Sites," http://www.top500.org/ (visited March 2012). [Online]. Available: http://www.top500.org/

[2] "ASC Sequoia," https://asc.llnl.gov/computing_resources/sequoia (visited May 2011). [Online]. Available: https://asc.llnl.gov/computing_resources/sequoia/

[3] K. Bergman *et al.*, "ExaScale Computing Study: Technology Challenges in Achieving Exascale Systems," Defense Advanced Research Projects Agency Information Processing Techniques Office (DARPA IPTO), Tech. Rep., September 2008.

[4] B. Schroeder and G. A. Gibson, "A large-scale study of failures in high-performance computing systems," in *Dependable Systems and Networks (DSN 2006)*, Philadelphia, PA, June 2006.

[5] M. A. Heroux, D. W. Doerfler, P. S. Crozier, J. M. Willenbring, H. C. Edwards, A. Williams, M. Rajan, E. R. Keiter, H. K. Thornquist, and R. W. Numrich, "Improving performance via mini-applications," Sandia National Laboratory, Tech. Rep. SAND2009-5574, 2009.

[6] S. J. Plimpton, "Fast parallel algorithms for short-range molecular dynamics," *Journal Computation Physics*, vol. 117, pp. 1–19, 1995.

[7] P. H. Hargrove and J. C. Duell, "Berkeley lab checkpoint/restart (blcr) for linux clusters," *Journal of Physics: Conference Series*, vol. 46, no. 1, 2006.

[8] E. N. Elnozahy, L. Alvisi, Y.-M. Wang, and D. B. Johnson, "A survey of rollback-recovery protocols in message-passing systems," *ACM Computing Surveys*, vol. 34, no. 3, pp. 375–408, 2002.

[9] E. N. Elnozahy and J. S. Plank, "Checkpointing for peta-scale systems: A look into the future of practical rollback-recovery," *IEEE Transactions on Dependable and Secure Computing*, vol. 1, no. 2, pp. 97–108, April-June 2004.

[10] K. Ferreira, R. Riesen, P. Bridges, D. Arnold, J. Stearley, J. H. L. III, R. Oldfield, K. Pedretti, and R. Brightwell, "Evaluating the viability of process replication reliability for exascale systems," in *SC*, S. Lathrop, J. Costa, and W. Kramer, Eds. ACM, Nov. 2011.

[11] D. Z. Pan and M. A. Linton, "Supporting reverse execution for parallel programs," in *1988 ACM SIGPLAN and SIGOPS Workshop on Parallel and Distributed Debugging (PADD '88)*. Madison, WI: ACM Press, 1988, pp. 124–129.

[12] K. Li, J. F. Naughton, and J. S. Plank, "Real-time, concurrent checkpoint for parallel programs," in *2nd ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPOPP '90)*. Seattle, Washington: ACM, 1990, pp. 79–88.

[13] N. H. Vaidya, "A case for two-level distributed recovery schemes," in *ACM SIGMETRICS Joint International Conference on Measurement and Modeling of Computer Systems*, ser. SIGMETRICS '95/PERFORMANCE '95. New York, NY, USA: ACM, 1995, pp. 64–73. [Online]. Available: http://doi.acm.org/10.1145/223587.223596

[14] J. Plank, K. Li, and M. Puening, "Diskless checkpointing," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 9, no. 10, pp. 972–986, oct 1998.

[15] A. Moody, G. Bronevetsky, K. Mohror, and B. R. de Supinski, "Design, modeling, and evaluation of a scalable multi-level checkpointing system," in *ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis (SC '10)*, 2010, pp. 1–11. [Online]. Available: http://dx.doi.org/10.1109/SC.2010.18

[16] G. Stellner, "Cocheck: Checkpointing and process migration for MPI," in *International Parallel Processing Symposium*. Honolulu, HI: IEEE Computer Society, April 1996, pp. 526–531.

[17] V. C. Zandy, B. P. Miller, and M. Livny, "Process hijacking," in *8th International Symposium on High Performance Distributed Computing (HPDC '99)*, Redondo Beach, CA, August 1999, pp. 177–184.

[18] J. Bent, G. Gibson, G. Grider, B. McClelland, P. Nowoczynski, J. Nunez, M. Polte, and M. Wingate, "Plfs: a checkpoint filesystem for parallel applications," in *Conference on High Performance Computing Networking, Storage and Analysis (SC '09)*, 2009, pp. 21:1–21:12. [Online]. Available: http://doi.acm.org/10.1145/1654059.1654081

[19] J. S. Plank, Y. Chen, K. Li, M. Beck, and G. Kingsley, "Memory exclusion: Optimizing the performance of checkpointing systems," *Software – Practice & Experience*, vol. 29, no. 2, pp. 125–142, 1999.

[20] E. N. Elnozahy, D. B. Johnson, and W. Zwaenpoel, "The performance of consistent checkpointing," in *11th IEEE Symposium on Reliable Distributed Systems*, Houston, TX, 1992. [Online]. Available: citeseer.ist.psu.edu/elnozahy92performance.html

[21] G. Bronevetsky, D. Marques, K. Pingali, S. McKee, and R. Rugina, "Compiler-enhanced incremental checkpointing for openmp applications," in *IEEE International Symposium on Parallel&Distributed Processing*, 2009, pp. 1–12. [Online]. Available: http://portal.acm.org/citation.cfm?id=1586640.1587642

[22] J. S. Plank, M. Beck, G. Kingsley, and K. Li, "Libckpt: Transparent checkpointing under unix," in *USENIX Winter 1995 Technical Conference*, New Orleans, LA, January 1995, pp. 213–224.

[23] C.-C. Li and W. Fuchs, "Catch-compiler-assisted techniques for checkpointing," in *Fault-Tolerant Computing, 1990. FTCS-20. Digest of Papers., 20th International Symposium*, jun 1990, pp. 74–81.

[24] J. S. Plank and K. Li, "ickp: A consistent checkpointer for multicomputers," *Parallel & Distributed Technology: Systems & Applications, IEEE*, vol. 2, no. 2, pp. 62–67, 1994.

[25] J. S. Plank, J. Xu, and R. H. B. Netzer, "Compressed differences: An algorithm for fast incremental checkpointing," University of Tennessee, Tech. Rep. CS-95-302, August 1995. [Online]. Available: http://web.eecs.utk.edu/~plank/plank/papers/CS-95-302.html

[26] A. Moshovos and A. Kostopoulos, "Cost-effective, high-performance giga-scale checkpoint/restore," University of Toronto, Tech. Rep., November 2004.

[27] J. Lee, M. Winslett, X. Ma, and S. Yu, "Enhancing data migration performance via parallel data compression," in *International Parallel and Distributed Processing Symposium*, 2002, pp. 444–451.

[28] J. T. Daly, "A higher order estimate of the optimum checkpoint interval for restart dumps," *Future Gener. Comput. Syst.*, vol. 22, no. 3, pp. 303–312, 2006.

[29] M. Bougeret, H. Casanova, M. Rabie, Y. Robert, and F. Vivien, "Checkpointing strategies for parallel jobs," in *SC*, S. Lathrop, J. Costa, and W. Kramer, Eds. ACM, 2011, p. 33.

[30] Sandia National Laboratories. (2010, April) The LAMMPS molecular dynamics simulator. [Online]. Available: http://lammps.sandia.gov

[31] K. G. M. Jr., "Compression tools compared," *Linux Journal*, no. 137, September 2005.

[32] P. Deutsch, "Deflate compressed data format specification." [Online]. Available: ftp://ftp.uu.net/pub/archiving/zip/doc

[33] J. Ziv and A. Lempel, "A universal algorithm for sequential data compression," *Information Theory, IEEE Transactions on*, vol. 23, no. 3, pp. 337–343, May 1977.

[34] "7zip project official home page," http://www.7-zip.org.

[35] I. Pavlov, "Lzma sdk (software development kit)," 2007. [Online]. Available: http://www.7-zip.org/sdk.html

[36] J. G. Elytra, "Parallel data compression with bzip2."

[37] E. Gabriel *et al.*, "Open MPI: Goals, concept, and design of a next generation MPI implementation," in *Recent Advances in Parallel Virtual Machine and Message Passing Interface*, ser. Lecture Notes in Computer Science, D. Kranzlmüller, P. Kacsuk, and J. Dongarra, Eds. Springer Berlin / Heidelberg, 2004, vol. 3241, pp. 353–377, 10.1007/978-3-540-30218-6_19. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-30218-6_19

[38] B. Schroeder and G. A. Gibson, "Understanding failures in petascale computers," *Journal of Physics Conference Series*, vol. 78, no. 1, 2007.

[39] S. Lang, P. Carns, R. Latham, R. Ross, K. Harms, and W. Allcock, "I/o performance challenges at leadership scale," in *Conference on High Performance Computing Networking, Storage and Analysis (SC '09)*, 2009, pp. 40:1–40:12. [Online]. Available: http://doi.acm.org/10.1145/1654059.1654100

[40] G. Shipman, D. Dillow, S. Oral, and F. Wang, "The Spider center wide file system: From concept to reality," in *Proceedings of the 2009 Cray User Group (CUG) Conference*, Atlanta, GA, May 2009.

[41] B. Barney. (2011, August) Introduction to livermore computing resources. [Online]. Available: http://computing.llnl.gov/tutorials/lc_resources

[42] G. Grider, "Exa-scale FSIO: Can we get there? can we afford to?" in *Proceedings of the 7th IEEE Workshop on Storage Network Architecture and Parallel I/Os*, 2011.

[43] S. Lathrop, J. Costa, and W. Kramer, Eds., *Conference on High Performance Computing Networking, Storage and Analysis, SC 2011, Seattle, WA, USA, November 12-18, 2011*. ACM, 2011, 2011.

Three Researchers, Five Conjectures: An Empirical Analysis of TOM-Skype Censorship and Surveillance

Jeffrey Knockel, Jedidiah R. Crandall, and Jared Saia
University of New Mexico
Dept. of Computer Science
{jeffk, crandall, saia}@cs.unm.edu

Abstract

We present an empirical analysis of TOM-Skype censorship and surveillance. TOM-Skype is an Internet telephony and chat program that is a joint venture between TOM Online (a mobile Internet company in China) and Skype Limited. TOM-Skype contains both voice-over-IP functionality and a chat client. The censorship and surveillance that we studied for this paper is specific to the chat client and is based on keywords that a user might type into a chat session.

We were able to decrypt keyword lists used for censorship and surveillance. We also tracked the lists for a period of time and witnessed changes. Censored keywords range from obscene references, such as 二女一杯 (two girls one cup, the motivation for our title), to specific passages from 2011 China Jasmine Revolution protest instructions, such as 成都 春熙路麦当劳门前 (McDonald's in front of Chunxi Road in Chengdu). Surveillance keywords are mostly related to demolitions in Beijing, such as 灵境胡同拆迁 (Ling Jing Alley demolition).

Based on this data, we present five conjectures that we believe to be formal enough to be hypotheses that the Internet censorship research community could potentially answer with more data and appropriate computational and analytic techniques.

1 Introduction

How effective is keyword censorship at stifling the spread of ideas? Is constant surveillance necessary for effective Internet censorship? What are the computational, linguistic, political, and social problems faced by both the censors and the people seeking to evade censorship?

A good understanding of how Internet censorship works, how it is applied, and what its impacts are will require both ideas from the social sciences and computational ideas. Consider a relatively simple question such as if keyword-based censorship is effective at stopping protests when censorship keywords target specific advertised protest locations, *e.g.*, 西大直街康宁路路口世纪联华 (Corning West and Da Zhi Street intersection, Century Lianhua gate). Estimating the effectiveness of this entails an understanding of psychology to quantify the effects of perceived surveillance and uncertainty, meme spreading, social networking, content filtering, linguistics to anticipate attempts to evade the censorship, and many other factors.

In this paper, we propose five conjectures about censorship. Our conjectures are based on our recent results in reverse-engineering TOM-Skype censorship and surveillance, combined with past studies of Internet censorship. TOM-Skype is an Internet telephony and chat program that is a joint venture between TOM Online (a mobile Internet company in China) and Skype Limited. TOM-Skype contains both voice-over-IP functionality and a chat client, the former of which implements keyword-based censorship and surveillance that we have reverse-engineered.

We present these conjectures in a formal way, in an attempt to propose them as testable hypotheses on which future research can focus. We do not expect all of our conjectures to be true. However, all of them have the properties that: 1) they can in principle be empirically tested; and 2) determining whether they are true or false will advance our understanding of Internet censorship. We contend that the enumeration of such testable conjectures is critical in order for the study of Internet censorship to continue as a viable area of scientific research.

1.1 TOM-Skype results

In this paper, we give preliminary results from reverse-engineering different versions of TOM-Skype. Our results include the cryptography algorithms used for both censorship and surveillance, differences between TOM-Skype versions, fully decrypted lists of keywords with translations, changes to the lists over time, and a rough

categorization of three of the lists.

Recently, Nart Villeneuve demonstrated that the chat functionality of TOM-Skype triggers on certain keywords, preventing their communication and uploading messages to a server in China [16]. He provided some high-level analysis of what is censored and how this mechanism works. In this paper, we provide a more detailed analysis of TOM-Skype, including the algorithms for protecting the keywords that trigger censorship and surveillance. All versions of TOM-Skype have at least one of two separate lists: one that triggers both censorship and surveillance and one that only triggers surveillance. We have decrypted all lists for all versions of TOM-Skype that we analyzed, and we have translated the most recent version's lists and tracked changes in the lists. The encryption for protecting the keyword lists in earlier versions is based on a simple algorithm involving additions and exclusive-or operations on each byte, whereas the encryption for later versions is AES-based. The encryption for uploading conversations that trigger surveillance is DES-based. By overcoming the anti-debugging functionalities built into both Skype and TOM-Skype, we also have a detailed understanding of how the censorship and surveillance is implemented. Based on our analysis, we propose a set of five conjectures.

1.2 Related work

The Open Net Initiative is an excellent source of information about censorship in a variety of countries [6]. However, the descriptions of what is filtered are relatively high-level. The report by Zittrain and Edelman [17] is a good overview of some of China's censorship implementations.

The methods of China's HTTP keyword filtering were first published by the Global Internet Freedom Consortium [10]. Clayton *et al.* [3] published a more detailed study of this mechanism. The ConceptDoppler project [4] studied multiple routes and also used Latent Semantic Analysis [12] to reverse-engineer 122 blacklisted keywords by clustering around sensitive concepts and then probing. ConceptDoppler has generated two more recent lists as well.

Human Rights Watch [11], Reporters Without Borders [14], and others [13, 1] have released reports describing China's censorship regime. These reports often include insider information about what is censored [14] or perhaps full leaked blacklists, such as the list of keywords blocked in the QQChat chat program [11, Appendix I] or by a particular blog site [11, Appendix II]. In the case of the QQChat list, hackers found the list in the QQ software by doing a simple string dump of QQChat's dynamically linked libraries, *i.e.*, there was no encryption.

There are several factors that make the list we have obtained from TOM-Skype unique among the lists that have been made public thus far. The first is that, not only is the TOM-Skype list more up-to-date, but in the three weeks that we have been monitoring it we have recorded many updates to it. We plan to record daily updates for a long period of time. Also, we believe our list is the first to provide more than anecdotal evidence of some of the shorter-term applications of censorship, such as the censorship of specific intersections where protesters planned to meet or events in the news. Our list is not only complete, but there is a clear separation between censorship keywords and surveillance keywords.

The prior work that is closest to ours is the aforementioned study of TOM-Skype by Villeneuve [16]. That analysis was based on obtaining the uploaded conversations, which were available for download on the server that TOM-Skype uploaded them to at the time, and performing clustering and other aggregate analyses of this data. In contrast, the analysis we present in this paper is based on reverse-engineering of the TOM-Skype implementation of censorship and of the cryptography for protecting the keyword blacklist. Thus, we are able to present exactly what the keywords are and make a clear distinction between keywords that evoke censorship and surveillance and those that only evoke surveillance.

There has been some amount of historical, political, economic, and legal discussion of the potential effectiveness and applications of Internet censorship in China and elsewhere [1, 13, 7, 2, 5]. Our goal in this paper is to present data that can help the research community move toward formalizing conjectures about Internet censorship that can be tested computationally. We propose five such conjectures after presenting our data.

This paper is structured as follows. In Section 2 we present our findings from reverse-engineering TOM-Skype. Section 3 discusses the two keyword blacklists for the most recent version of TOM-Skype, which we have translated and analyzed in detail. Then we propose five conjectures and some recommendations for future work in Section 4.

2 Empirical analysis of TOM-Skype

In this section we describe the basic empirical results from our efforts to reverse-engineer TOM-Skype.

2.1 Censorship mechanisms

Each version of TOM-Skype that we analyzed features shared censorship mechanisms. Each binary contains a built-in, encrypted list of keywords to censor, and, via HTTP, each client downloads at least one additional encrypted list of censored keywords called a "keyfile" from TOM's servers. Each client uses at least one of these lists to censor incoming and/or outgoing chat messages at any

time.

However, we also found stark differences in their censorship implementations. Different versions use different encryption algorithms, different built-in keyword lists, and download keyfiles from different locations. Moreover, implementations vary in whether they censor incoming and/or outgoing chat messages.

2.1.1 TOM-Skype 3.6 and 3.8

We first analyzed TOM-Skype 3.6.4.316 and 3.8.4.44. We found that these clients censor both incoming and outgoing chat messages by failing to render them in one's chat window and by failing to record them in one's chat history. Censored outgoing messages are additionally never sent.

When the client starts up, words are initially censored according to the keyword list built into the binary. After a keyfile is downloaded from TOM's servers, the downloaded keyfile *substitutes* the built-in keyword list. By using a packet sniffer, we found that these clients download keyfiles from the following URL:

skypetools.tom.com/agent/newkeyfile/keyfile

To decrypt this file, we redirected `skypetools.tom.com` DNS queries to our own HTTP server, allowing us to force TOM-Skype to load keyfiles of our choosing. Then, through binary search, we were able to locate the ciphertext entry for the keyword "fuck" in the keyfile, which is the keyword that Villeneuve [16] used. We started with a known-plaintext analysis of this keyword. We then employed a chosen ciphertext attack by adding initially single-character words to the list to see which words were filtered by TOM-Skype. As we recognized patterns and became more familiar with the decryption algorithm, we were soon able to censor entire words. The decryption algorithm that we discovered follows:

Algorithm 1 Decrypting TOM-Skype 3.6 keyfiles

1: **procedure** DECRYPT($C_{0..n}, P_{1..n}$)
2: **for** $i \leftarrow 1, n$ **do**
3: $P_i = (C_i \oplus 0\text{x}68) - C_{i-1} \pmod{0\text{xff}}$
4: **end for**
5: **end procedure**

The ciphertext always has one more byte than the plaintext, since the ciphertext's first byte serves as an initialization vector. We found that this algorithm also decrypts the keyword lists built into the binaries.

2.1.2 TOM-Skype 4.0 and 4.2

Next we analyzed TOM-Skype 4.0.4.226 and 4.2.4.104. These versions implemented censorship similarly as our tested 3.6 and 3.8 versions, except they download a keyfile from the following URL:

`a`*n*`.`skype.tom.com/installer/agent/keyfile

where n is a pseudorandom, uniformly-distributed integer between 1 and 8, inclusive. This file can be decrypted using the previous algorithm, as can these clients' built-in keyword lists.

2.1.3 TOM-Skype 5.0 and 5.1

Finally, we analyzed TOM-Skype 5.0.4.14 and 5.1.4.10. These versions delegate censorship to a separate out-of-process binary `ContentFilter.exe`. We found these versions to only perform censorship on incoming messages.

We found that `ContentFilter.exe` downloads keyfiles from the following URL:

skypetools.tom.com/agent/keyfile

which, as before, substitutes the list built into `ContentFilter.exe`'s binary. Moreover, we found that only TOM-Skype 5.1.4.10's `ContentFilter.exe` additionally downloads a keyfile from the following URL:

skypetools.tom.com/agent/keyfile_u

The words in the latter keyfile are not used for censorship but only for surveillance, detailed in the next section.

We found that both of these keyfiles and the keyword lists built into `ContentFilter.exe` are encrypted with a 256-bit AES key in ECB mode. This UTF-16LE-encoded key is originally known to have been used to encrypt the downloaded keyfile in TOM-Skype 2.5 [9]:

$\texttt{CENSOR\_KEY}_{5.0}$ `= "0sr TM#RWFD,a43 "`

In UTF-16LE encoding, this key is 256 bits, although half of the bytes are null.

2.2 Surveillance mechanisms

We found all versions of TOM-Skype analyzed in the previous section to perform surveillance except 5.0.4.14. Each of the other versions, whenever it performs censorship, reports back encrypted text in a query string to the following URL:

a*n*.skype.tom.com/installer/tomad/ContentFilterMsg.php

where again n is a pseudorandom, uniformly-distributed integer between 1 and 8, inclusive.

As reported in the previous section, TOM-Skype 5.1.4.10 has an additional downloaded keyfile containing words only used for surveillance of but not the censorship of incoming messages.

By reverse engineering TOM-Skype 3.8.4.44's `Skype.exe` and TOM-Skype 5.1.4.10's `ContentFilter.exe` binaries, we discovered

two DES keys used to encrypt surveillance text, where which is used depends on the version. Being outside of the main Skype binary, we first targeted `ContentFilter.exe`, as `Skype.exe` is known to contain anti-debugging measures that cause the program to crash when attached with a debugger [8]. In `ContentFilter.exe`, we discovered that before surveillance text is encrypted, each sequence of six bytes of the text is used as the first six bytes of each eight-byte DES block to be encrypted. The remaining two bytes are pseudorandom, uniformly-distributed between 0x27 and 0x73, inclusive. DES encryption is performed on all blocks in ECB mode using the following 64-bit key ASCII-encoded:

$$\texttt{SURVEIL\_KEY}_{4.0} = \texttt{"X7sRUjL\0"}$$

which we also found to be used by TOM-Skype 4.0.4.226 and 4.2.4.106. Note that the 8th byte of $\texttt{SURVEIL\_KEY}_{4.0}$ is a null byte. Although we express this null byte for clarity, in TOM-Skype's implementation, this byte is the null-terminating byte of the string. This string, ASCII-encoded, also appears as a literal in the binary.

To discover the other DES key and circumvent the anti-debugging measures in TOM-Skype 3.8.4.44's `Skype.exe` executable, we used DLL injection, a technique where we cause TOM-Skype's calls to library functions to instead call code that we have written. We previously observed that, when stuffing each eight-byte DES block with the two random bytes, `ContentFilter.exe` reseeds the random number generator with a hardware time that it retrieves via a library call. We similarly found and then exploited this behavior in `Skype.exe` by causing each of these library calls for the time to call our code and sleep for ten seconds, allowing us to attach with a debugger while TOM-Skype slept. After attaching, we suspended all other threads not sleeping in our code. Then we observed the encryption process in the debugger before TOM-Skype's anti-debugging measures activated. We found the eight bytes of the DES key embedded in instructions in eight cases of a compiled switch statement. When ASCII-encoded, the following 64-bit DES key is used in ECB mode to encrypt surveillance text in TOM-Skype 3.6.4.316 and 3.8.4.44:

$$\texttt{SURVEIL\_KEY}_{3.6} = \texttt{"32bnx23l"}$$

After decrypting the surveillance text, we found that less information was reported in Skype 5.1.4.10 versus older versions. Here is example surveillance plaintext for a censored outgoing message for tested versions before 5.*x*:

```
jdoe falungong 4/24/2011 2:25:53 AM 0
```

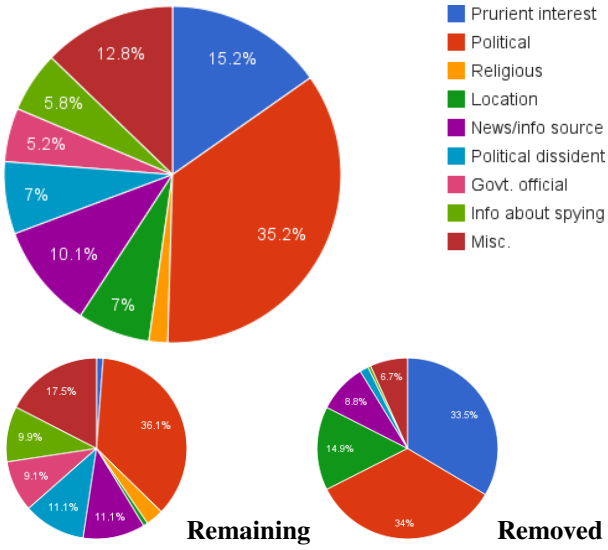


Figure 1: **Distribution of keywords on the keyword list that evokes both censorship and surveillance.**

Here "jdoe" is the sender of the outgoing message, "falungong" is the offending message in its entirety, followed by the date, time, and a "0" to indicate that the censored message was outgoing. When an incoming message is censored, then that message's sender is reported instead and the trailing "0" becomes a "1."

In contrast to versions before 5.*x*, here is example surveillance for version 5.1.4.10:

```
falungong 4/24/2011 2:29:57 AM 1
```

No username appears to be reported. Since all surveillance-related text in version 5.1.4.10 is incoming, a "1" will always trail the message.

3 Keyword analysis

We found that the built-in keyword lists in Tom-Skype 3.6.4.316, 3.8.4.44, 4.0.4.226, and 4.2.4.104 were identical and contained 108 censored words in either English or Chinese. Moreover, although they retrieve keyfiles from different URL's, as of 4/29/2011, each retrieved keyfile was identical and contained 442 words in either English or Chinese. These keyfiles have not been modified since 4/22/2011, and, if we believe their HTTP last-modified headers, they were last modified on 3/11/2011.

Since we began downloading Tom-Skype 5.0.4.14 and 5.1.4.10 keyfiles on 4/22/2011, we have noticed substantial changes to both the censorship keyfile and the 5.1-specific surveillance-only keyfile. We do not know the reason for the changes, but one possible reason is the human rights talks between China and the United States that were scheduled for 4/27/2011 and 4/28/2011 [15]. We focus on these lists and their changes in this section.

Before 4/22/2011, the keyword list that evokes both censorship and surviellance contained **Prurient interests**, *e.g.*, 两女一杯 (Two girls one cup), 二男一马 (Two men one horse), and 操烂 (Fuck rotten); **Political** terms, *e.g.*, 六四 (Liu Si, in reference to the Tiananmen Square protests that occurred on June 4th, 1989—this is literally the numbers "64"), 陆肆 (Lu Si, a homophonic way of writing 六四), and 河蟹社会 (River Crab Society, a corruption of "和谐社会", which means "harmonious society"); **Religious** terms, *e.g.*, 法轮 (Falun) and 观音法门 (Quan Yin Method, a Buddhist meditation method); **Locations** of planned events such as protests, *e.g.*, 广州 天河体育中心正门 (The main entrance of the Guangzhou Tianhe Sports Center in Guangzhou) and 杭州 湖滨路 凯悦酒店前至音乐喷泉旁一带 (Hyatt Regency Hubin Road, next to the area in front of the musical fountain in Hangzhou); **News/information sources**, *e.g.*, 维基百科 (Wikipedia) and 加拿大广播公司 (Canadian Broadcasting Corporation); **Political dissidents**, *e.g.*, 刘晓波 (Liu Xiaobo) and 吾尔开希 (Wu'er Kaixi, a student leader from the Tiananmen Square protests of 1989); **Government officials**, *e.g.*, 刘延东 (Liu Yandong, she is the highest ranking female in the communist party and a member of the politburo—she is caught up in a scandal involving her son-in-law) and 影帝温 (Oscar best actor winner, a nickname for Wen Jiabao after he appeared to cry insincerely on television); **Information about spying**, *e.g.*, 手机窃听软件免费下载 (Phone tapping software free download) and 三利普 (Three gain universal, part of a product name at sunlips.com, 三利普加强版二代橡皮, that appears to be a remote microphone for spying), and other **Miscellaneous** keywords. The contents of the original keyword list (before 4/22/2011) is shown in Figure 1. Figure 1 also shows the distribution of the words that were taken away (right) and that remained (left) on 4/22/2011.

Figure 2 shows the distribution of the 158 words for the list that evokes only surveillance. Most of this list is specific demolition sites or other references to the demolitions in Beijing, where people have reportedly been forced from their homes and their houses demolished to make room for future construction. The only words on this list that are not related to these demolitions are five keywords related to the Shouwang church, a Christian church in Beijing that illegally holds congregations outdoors in public places, and two references that appear to be names of companies or parts of a company name: 西屋国际 (Westinghouse International) and 大恒 (Daheng).

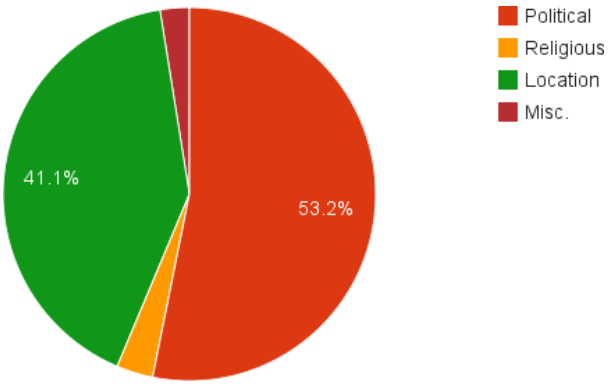


Figure 2: **Distribution of keywords on the keyword list that evokes only surveillance.**

Another interesting aspect of the Skype lists we analyzed, specifically the one that evokes both censorship and surveillance, is that there are many phrases that appear to be exact phrases taken from online instructions for protesters or calls for sit-ins. For example, one document suggests that protesters should take symbolic actions that are ambiguous so that they will not be arrested by the police, *e.g.*, 拿着麦克风表示自由 (Hold a microphone to indicate liberty—a passage where the document suggests that if people want to signify that they need liberty, they should put a picture of a microphone on their clothes or bag and gesture as if speaking into a microphone when they speak). Another document calls for a sit-in in response to the demolitions in Beijing, and another instructs people on how to make an origami jasmine flower and pleads protesters to not get arrested because this is an early phase of the protests.

4 Conjectures

In this section we present five conjectures that are based on the data we presented in this paper as well as previously available data on Internet censorship. Our aim is for these conjectures to be testable hypotheses so that the research community can confirm or refute them given the right data and appropriate computational and analytical techniques. All of the following conjectures are limited to content and traffic within a country where the censoring occurs. It is unlikely that all of these conjectures are true. However, we believe that each of these conjectures have the properties that: 1) they can in principle be empirically tested; 2) determining whether they are true or false will advance our understanding of Internet censorship. The five conjectures are:

1. **Effectiveness Conjecture:** "Censorship is effective, despite attempts to evade it." More formally, censoring a keyword reduces the number of accesses to content that either contains that keyword or contains related keywords. This may simply be because the quality-of-service for accessing content that is the target of censorship goes down whenever viewers or publishers must change their behavior in

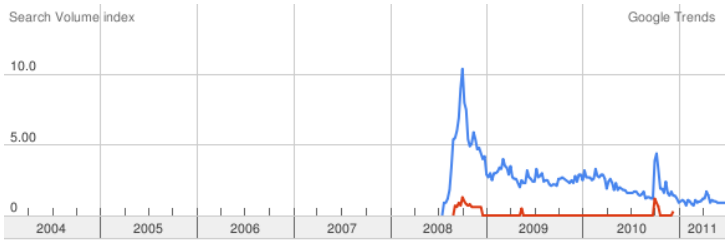


Figure 3: **Google Trends data for the Chinese-language searches for the "2 girls 1 cup" meme. The higher-volume data line is for 两女一杯 and the lower-volume data line is for 二女一杯.**

Figure 4: **Google Trends data for the English-language searches for the "2 girls 1 cup" meme.**

some specific way to access or disseminate the content.

The motivation for including this conjecture is that the censorship keywords we found in TOM-Skype that were phrases from specific documents were from documents that are not prevalent on the Web. These documents were presumably important enough that they were targeted by censorship, but there are very few instances of these documents online which suggests that they were not as widely disseminated as the authors of the documents had intended. For example, 拿着麦克风表示自由 (Hold a microphone to indicate liberty) is a phrase from instructions for Jasmine revolution protests in China in 2011. As of July 2011, searching for this exact phrase in quotation marks in the United States version of Google at www.google.com, which is known not to exclude results in response to China's Internet censorship, returns only nine results. The document appears in other places in addition to these nine results online, but with blacklisted phrases divided with dashes or paraphrased.

2. **Spread Skew Conjecture:** "Censored memes spread differently than uncensored memes." More formally, censoring a keyword qualitatively changes the time *vs.* number of accesses plot of the keyword. In particular, the distribution is not simply scaled downwards by a fixed amount, but may also be more or less spread out over time and have a different distribution.

   Because most of the censored keywords in our data do not show significant traffic volume in Google Trends, we cannot support this conjecture with our data. However, the conjecture was inspired by the Internet meme "2 girls 1 cup", which in Chinese is targeted by four keywords in our data (两女一杯, 二女一杯, 俩女一杯, and 两女吃一杯, only the first two of which have enough search volume to appear in Google Trends results). Figures 3 and 4 show the Google Trends results for Chinese and English for this meme, respectively. While the search volume of the Chinese-language versions of the meme is too small to make direct comparisons or extrapolate too much information about the distribution of the meme, the fact that the Chinese-language version of the meme has a lower peak and a taller tail is what inspired the spread skew conjecture. For each distribution, if we consider the 150 weeks after we first have data for that distribution, the English distribution has skewness 2.9511 but the Chinese distribution has skewness 2.0506. This may be a result of censorship effectively removing a portion of the right tail of the distribution over time. Note that Google Trends data is normalized in these graphs and that the English version of the meme has 32.0 times the total traffic volume as the most popular Chinese version.

3. **Interactions of Secrecy and Surveillance Conjecture:** "Keyword based censorship is more effective when the censored keywords are unknown and online activity is, or is believed to be, under constant

surveillance." More formally, for a given word that is perceived to be sensitive, accesses to the content related to that word will be fewer if it is unknown whether the word is censored or not. Further, accesses to related content will be fewer if Internet users believe that their online activities are being recorded and monitored.

This conjecture is inspired by the fact that the entity censoring Tom-Skype has made efforts to keep the list of censored and surveilled keywords and the surveillance traffic private.

4. **Peer-to-peer *vs.* Client-Server Conjecture:** "The types of keywords censored in peer-to-peer communications are fundamentally different than the types of keywords censored in client-server communications."

   For example, the censored keyword list for TOM-Skype, a peer-to-peer application, contains a higher fraction of proper nouns than censored keyword lists for client-server applications (see [4, 11] in addition to our list for examples of both types of lists). In particular, we noticed a high number of names of people and places on the censorship blacklist for TOM-Skype.

5. **Neologism Conjecture:** "Neologisms are an effective technique in evading keyword based censorship, but censors frequently learn of their existence." More formally, if a neologism is used in place of a censored keyword, the content will spread relatively freely until the neologism itself is censored. Phenomena such as "reblogging" and "retweeting" are impacted by this.

   We included this conjecture because of the large number of neologisms present in our data. Examples include 陆肆 (Lu Si, which sounds like 六四, or 64, in reference to the June 4th Tienanmen Square incident) and 影帝温 (Oscar best actor winner, a nickname for Wen Jiabao). Note that some keywords have a large number of possible neologisms, so that this conjecture may not be true for a large number of keywords. For example, we have seen instances in online Web forums of 六四, or 64, being referred to as "32 + 32" or "8 squared". Uncertainty about what keywords are being blacklisted and the possibility of surveillance are also factors in the effectiveness of neologisms, however.

5 Conclusion

In conclusion, we have presented new data about Internet censorship in China based on our efforts to reverse-engineer TOM-Skype and proposed five conjectures based on this data. For future work, our hope is that the research community will test these and other conjectures with more data and appropriate computational and analytic techniques.

Note: Complete lists with translations of the censorship and surveillance keywords for TOM-Skype are available at http://cs.unm.edu/~jeffk/tom-skype/.

Acknowledgments

We would like to thank the anonymous FOCI reviewers for their insightful comments. We would also like to thank the many people who helped us improve our translations and gave feedback on other aspects of the paper. This material is based upon work supported by the National Science Foundation under Grant Nos. CCR #0313160, CAREER #0644058, CAREER #0844880, and TC-M #090517.

References

[1] CHASE, M. S., AND MULVENON, J. C. *You've Got Dissent! Chinese Dissident Use of the Internet and Beijing's Counter-Strategies*. RAND Corporation, 2002.

[2] CLAYTON, R. Failures in a hybrid content blocking system. In *Privacy Enhancing Technologies* (2005), pp. 78–92.

[3] CLAYTON, R., MURDOCH, S. J., AND WATSON, R. N. M. Ignoring the great firewall of china. *I/S: A Journal of Law and Policy for the Information Society 3*, 2 (2007), 70–77.

[4] CRANDALL, J. R., ZINN, D., BYRD, M., BARR, E., AND EAST, R. ConceptDoppler: a weather tracker for Internet censorship. In *Proc. of 14th ACM Conference on Computer and Communications Security (CCS)* (2007).

[5] DANEZIS, G., AND ANDERSON, R. The economics of resisting censorship. *IEEE Security and Privacy 3*, 1 (2005), 45–50.

[6] DEIBERT, R. J., PALFREY, J. G., ROHOZINSKI, R., AND ZITTRAIN, J. Access denied: The practice and policy of global internet filtering. *The MIT Press* (2007).

[7] DORNSEIF, M. Government mandated blocking of foreign web content. In *Security, E-Learning, E-Services: Proceedings of the 17. DFN-Arbeitstagung über Kommunikationsnetze* (2003), J. von Knop, W. Haverkamp, and E. Jessen, Eds., Lecture Notes in Informatics, pp. 617–648.

[8] FABRICE, D., AND KORTCHINSKY, K. Vanilla skype part 1. Available at http://recon.cx/en/f/vskype-part1.pdf.

[9] FABRICE, D., AND KORTCHINSKY, K. Vanilla skype part 2. Available at http://recon.cx/en/f/vskype-part2.pdf.

[10] The Great Firewall Revealed. Whitepaper released by the Global Internet Freedom Consortium in December of 2002.

[11] "Race to the Bottom": Corporate Complicity in Chinese Internet Censorship. In *Human Rights Watch* (August 2006). http://www.hrw.org/reports/2006/china0806.

[12] LANDAUER, T. K., FOLTZ, P. W., AND LAHAM, D. Introduction to latent semantic analysis. *Discourse Processes 25* (1998), 259–284.

[13] LIANG, C. Red light, green light: has China achieved its goals through the 2000 Internet regulations? *Vanderbilt Journal of Transnational Law 345* (2001).

[14] MR. TAO. China: Journey to the heart of Internet censorship. Investigative report sponsored by Reporters Without Borders and Chinese Human Rights Defenders, Oct 2007.

[15] PERALTA, E. China, United States to Begin Human Rights Talks. Blog post 26 April 2011, URL: http://www.npr.org/blogs/thetwo-way/2011/04/26/135745130/china-united-states-to-begin-human-rights-talks, accessed 8 May 2011.

[16] VILLENEUVE, N. Breaching trust: An analysis of surveillance and security practices on China's TOM-Skype platform. Available at http://www.infowar-monitor.net/breachingtrust/.

[17] ZITTRAIN, J., AND EDELMAN, B. Internet filtering in China. *IEEE Internet Computing 7*, 2 (2003), 70–77.

Formica ex Machina: Ant Swarm Foraging From Physical to Virtual and Back Again

Joshua P. Hecker, Kenneth Letendre, Karl Stolleis, Daniel Washington, and Melanie E. Moses

University of New Mexico,
Albuquerque, NM 87111
{jhecker,mmoses}@cs.unm.edu
{kletendr}@unm.edu
http://cs.unm.edu/

Abstract. Ants use individual memory and pheromone communication to achieve effective collective foraging. We implement these strategies as distributed search algorithms in robotic swarms. Swarms of simple robots are robust, scalable and capable of exploring for resources in unmapped environments. We test the ability of individual robots and teams of three robots to collect tags distributed at random and in clustered distributions. Teams of three robots that forage based on individual memory without communication collect RFID tags from all three distributions approximately twice as fast as a single robot using the same strategy. Adding pheromone-like communication in the teams of three robots improves foraging success. Our simulation system mimics the foraging behaviors of the robots and replicates our results, with slight improvements in the three robot teams. Simulated swarms of 30 and 100 robots collect tags 8 and 22 times faster than teams of three robots. This work demonstrates the feasibility of programming large robotic swarms for collective tasks such as retrieval of dispersed resources, mapping and environmental monitoring. It also lays a foundation for evolving collective search algorithms *in silico* and then implementing those algorithms *in machina* in robust and scalable robotic swarms.

Keywords: swarm intelligence, robots, agent-based models, social insect foraging, genetic algorithms

1 Introduction

One goal of swarm robotics is to engineer groups of simple, low-cost robots that can cooperate as a cohesive unit to accomplish collection and exploration tasks such as mapping, monitoring, search and rescue, and foraging for resources in unmapped environments [4, 5, 8]. Ideally, robotic swarms are capable of exploring unknown environments without the benefit of prior knowledge to guide them. Individuals must adapt to sensor error and motor drift, and the swarm must function given variation, errors and failures in individual robots.

Biology often provides inspiration for approaches to achieve these design goals [4, 8, 21]. Biologically-inspired decentralized approaches in particular have enhanced scalability and robustness by removing single points of failure from communication bottlenecks and rigid control structures. Thus far such approaches have not yet reached the level of emergent coordination observed in natural systems [28].

Our contribution is inspired by colonies of seed harvester ants who forage for seeds in a desert environment using a combination of individual memory and information sharing through pheromone trails. Our robots are equipped with a sensor suite which mimics the real ants: time-based odometry approximates physical location analogous to the ants' stride integration [33], and ultrasound ranging measures distance to objects and corrects for drift similar to an ant's landmark-based navigation [16]. Like ants, the robots use individual memory and communication of previously successful search locations to improve search performance. Our robots search for radio-frequency identification (RFID) tags, and upon finding them, return to a central nest.

The search algorithm utilized by individual members of the swarm is derived from our previous work that used an agent-based model (ABM) guided by genetic algorithms (GA) to replicate foraging behaviors of seed harvester ants [11, 18]. We duplicate parameters from the ant model in the robots. For example the robots movement during uninformed search replicates the correlated random walk of virtual ants that was evolved by the GA to produce colonies that find seeds quickly. We modified the ABM to replicate the constraints of the robot hardware, and to model the behavior and environment of the robots in their search for RFID tags. This parallel physical and virtual implementation allows us to compare results from identical experiments *in machina* as implemented in physical robots and *in silico* in the ABM (as in [7, 19]. We conduct additional experiments with the ABM in which we scale up the size of the swarm, the number of tags, and the size of the area in which the virtual robots search. Because we see similar foraging success in simulated and robotic swarms with 1 and 3 individuals, these trials suggest future capabilities of swarms of 30 and 100 robots.

2 Background

2.1 Swarm Robotics

Swarm robotics is necessitated by problems that are inherently too complex or difficult for a single robot, and by the need to develop systems that are cheaper, more adaptive, and robust to failures, errors and dynamic environments [5, 8]. Like ant colonies and other complex biological systems, robotic swarms have potential to utilize efficient, robust, distributed approaches to physical tasks. Effective algorithms for swarm robotics must extend beyond simulation to intelligently deal with the complexities of navigating in real environments [19, 20, 7]: sensors are imperfect and may fail, collisions with obstacles (including other robots) are common, and real environments are dynamic, changing in response to external factors and the activities of the robots themselves. Further, approaches must balance the benefit of centralized information exchange with the scalability of decentralized approaches [24, 2, 27]. Even highly decentralized robot interactions show diminishing returns in which interference between robots can make swarm efficiency decrease as the swarm size grows [17].

Recent work has demonstrated the feasibility of swarms in which collectively intelligent behaviors emerge from distributed interactions among robots. Simultaneous localization and mapping (SLAM) enables robots to infer knowledge about unknown environments [10, 1, 22]. Localization space-trails (LOST) facilitate a shared world view between robots without a global coordinate system through the use of local landmarks and waypoints [32].

Simple low-cost platforms have been designed specifically to form robot swarms, e.g., [21, 7, 6, 30], but a great challenge exists in transforming a set of simple mobile components into a functional swarm. Robotic swarms have not yet approached the emergent intelligence of biological swarms [28], but a promising approach is to use evolutionary algorithms to determine the parameters of individual behavior that result in effective collective action [26, 9, 31, 12].

2.2 Biological Ants

Our algorithms are largely inspired by foraging in desert seed-harvester ants of the genus *Pogonomyrmex* [13]. These foragers typically leave their colony's single nest, travel in a relatively straight line to some location on their territory, and then switch to a searching behavior. The forager searches by moving in a correlated random walk, where the probability of turning is dependent on whether the forager expects to find seeds in the area (informed by pheromone trails or previous foraging success) or not. An informed ant has an initially high tendency to turn, keeping the ant in a small area. Over time, if a seed is not found, degree of turning decreases, which straightens out the search path, and the ant tends to wander farther from its initial search location [12].

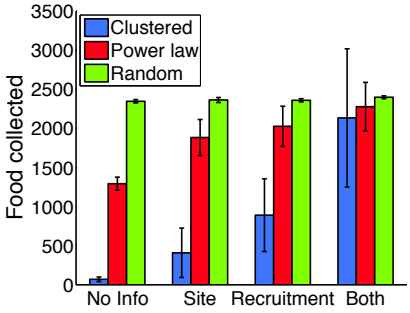
An ant with no prior expectations of finding seeds will use a smaller degree of turning and therefore explore a relatively larger area. When a forager finds a seed, it brings it directly back to the nest. Foragers often return to the location where they previously found a seed, in a process called site fidelity[23, 3, 13]. Seeds are hard to find, so the duration of a foraging trip, which includes travel time and search time, is dominated by the time spent searching for a seed [23, 3]. Effective search strategies for foragers will minimize the time spent searching for seeds (which minimizes the risk of foraging in the hot, dry desert) while maximizing the number of seeds collected. It is unclear exactly how often these ants lay and follow pheromone trails [16, 15, 25], but our recent work indicates laying and following pheromone trails to dense piles of food may be an effective component of these ants' foraging strategies [18, 12].

2.3 Agent-based Model

We have used Genetic Algorithms (GAs) to find the optimal balance of site fidelity and pheromone communication in simulated ant colonies [18]. We simulated ant foraging using a set of agent-based models (ABMs) of foragers on a grid, with parameters optimized by a GA to specify how ants travel from the nest, search, and use site fidelity and pheromone communication. GAs are an optimization technique that simulates the process of evolution by natural selection [14], just as biological ants undergo evolutionary pressure to maximize foraging success (among other goals and constraints). Therefore GAs were an appealing method for selecting parameters for our ant foraging model [29, 26]. The foraging success of virtual ants evolved by the GA is shown in Figure 1.

The ant foraging ABM was modified to model our swarm robots and our experimental setup. The simulation provides both a theoretical benchmark and a basic architecture for using GAs to optimize real world parameters. All *in machina* experiments have been duplicated *in silico*, and results are presented side by side to allow comparison.

Fig. 1: Bars represent number of seeds collected during simulated foragin trials by colonies of 100 foragers. Colonies forage on clustered, random, and power law distributed, after optimization by GA to mazimize food collection rate on those distributions. Simulations using site fidelity, pheromone recruitment, both methods together, or neither (no information use) are used as the fitness function in a GA that selects parameters governing travel from the nest, turning during the foragers' search behavior, and use of site fidelity and/or recruitment.



3 Methods

3.1 Hardware

While our algorithms and architecture are intended to be used with a variety of platforms, swarm performance will depend on the particular specifications of the hardware on which those algorithms are implemented. Our robots are built using easily obtained off-the-shelf components (Table 1) at a total cost of $450 per robot. System architecture is based on the Arduino open-source hardware platform, allowing for straightforward programming in a C++-style language based on Wiring. Sensor error is described in Table 2.

3.2 Search Algorithm

The search behavior used by the robots to locate RFID tags is shown in Fig. 2.

1. Set Search Site Location: The robot begins at the nest in the center and selects an initial search site location, encoded as a direction, d, and heading, h. This location is initially chosen at random, but may be influenced by memory or communication in subsequent foraging trips.
2. Travel to Search Site(yellow path) Travelling robots iterate through behaviors to avoid collisions with other robots, correct for motor drift, and communicate events with the coordination server.
3. Search for Tag (blue path): The robot moves in a correlated random walk with direction at time t drawn from a normal distribution centered around direction θ_{t-1} and standard deviation $SD = \omega + \gamma/t_s^{\delta}$, where ω determines the degree of turning during an uninformed search (i.e. at a random location), and γ/t_s^{δ} determines an additional degree of turning at the beginning of an informed search, and which decreases over time spent searching. Equation 1 results tight turns in an initially small area that expand to explore a larger area over time.
4. Travel to Nest (pink path): The robot leaves the location of the found tag, stepping toward the known nest location. The robot lays a pheromone on its return trip if count C of other tags detected in the 8-cell neighborhood of the collected tag is > 1. Pheromone evaporates exponentially with time.

Table 1: Robots components

Component	**Description**
Chassis	The Open Source Robotics OSbase chassis is a four-motor, treaded differential drive platform powered by a 7.4V LiPo rechargeable battery
Microprocessor	The Arduino Uno is an open-source, low-cost development board for the Atmel ATmega328 microprocessor. The ATmega328 is an 8-bit, 16 Mhz processor with 32K of onboard memory for program storage, and +5V logic and onboard power regulation
Motor Shield	The SparkFun Electronics L298 H-Bridge motor driver board controls the four onboard motors. The shield attaches via stacking headers onto the Arduino Uno
Wireless Shield	The SparkFun Electronics WiFly Shield provides wireless communication via standard 802.11b/g TCP protocol through the Roving Networks RN-131C module. Control is via the Uno SPI bus
Compass	The SparkFun Electronics HMC6352 digital compass board uses the Honeywell 6352 compass chip to report magnetic headings with a published accuracy of 2.5°. Module communication is via Arduino Uno TWI bus
GPS	The US GlobalSat EM-406a GPS receiver has a ceramic chip antenna and reports data via NEMA-formatted strings over the Arduino Uno hardware serial port with a 1Hz update rate. The GPS is not used in these experiments
Ultrasound	The Devantech SRF-05 ultrasonic rangefinder provides distance measurements up to 4 meters and communicates via two standard Arduino Uno digital pins
RFID Reader/Writer	The Parallax RFID module reads and writes data using standard 125kHz RFID tags and communicates over Arduino Uno pins using serial port emulation via the Arduino library

Table 2: Sensor error: Compass data show errors in degrees at a fixed heading. GPS data measures error from true location across 5000 data points collected in each location. Odometry data are from 20 trials for each location and measures ability to navigate to a point 5 meters away using programmed motor turning rate, time and compass heading. Ultrasound errors are measured from a concrete barrier 0.5 and 1.5 meters away.

	Location 1	**Location 2**
Compass (°)	2.5 ($\sigma = 1.7$)	3.0 ($\sigma = 0.78$)
GPS (m)	6.6 ($\sigma = 3.6$)	13 ($\sigma = 5.7$)
Odometry (cm)	20 ($\sigma = 7.7$)	22 ($\sigma = 5.5$)
Ultrasound (cm)	1.3 ($\sigma = 0.38$)	4.1 ($\sigma = 4.1$)

5. Set Next Search Location: On subsequent trips, d and h are determined by either returning to the previously found tag location if $C > 0$, or following a pheromone to a location identified by another robot.

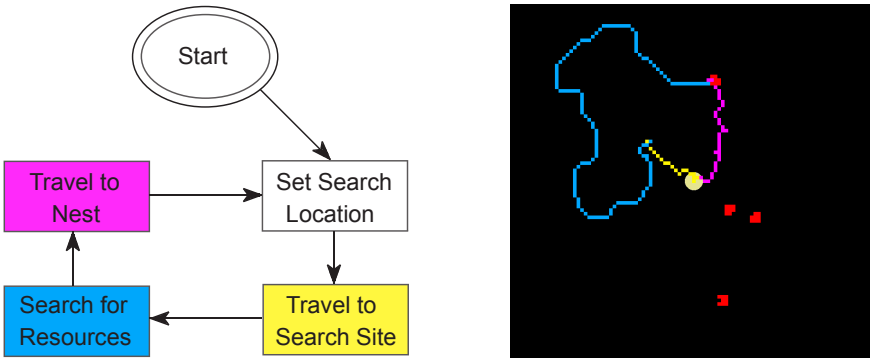


Fig. 2: A robot begins its search at a globally shared central nest site (center circle) and **sets a search location**. The robot then **travels to the search site** (yellow line). Upon reaching the search location, the robot **searches for tags** (blue line) until tags (red squares) are found or a probabilistic timeout occurs. After searching, the robot **travels to the nest** (purple line).

3.3 Experimental Design

We conducted experiments on outdoor concrete surfaces. Each trial runs for a maximum of one hour. A cardboard cylinder marks the center point and represents a home or 'nest' to which the robots return once they have located a tag. This center point is used for localizing and error correction by the robots' ultrasonic sensors. All robots involved in a trial are initially placed near the cylinder to minimize dead reckoning error. We program each robot to stay within a 3m radius 'virtual fence' to deter drift outside of the experimental area.

In every experiment, 32 RFID tags are arranged in one of three different patterns: random, clustered, or power law. The random layout has tags scattered throughout a ring between 50 cm and 200 cm in a uniform distribution (Figure 3(a)). The clustered layout has four piles of eight tags placed at 90° intervals at 50, 100, 150, and 200 cm in relation to the central nest (Figure 3(b)). The power law layout uses piles of varying size and number: one large pile of eight tags at 125 cm, two medium piles of four tags at 75 and 175 cm, four small piles of two tags at 50, 100, 150, and 200 cm, and eight randomly placed tags (Figure 3(c)). Experiments are replicated under identical conditions for individual robots and for groups of three bots.

Robot locations are continually transmitted over WiFi to a central server and logged for analysis. When a tag is found, its unique identification number is transmitted back to the server, providing us with a detailed record of tag discovery. Note that tags can only be read once, simulating seed retrieval. The

central server also acts as a coordinator for virtual pheromone trails. Locations deemed important enough to require a pheromone value (i.e. those with two or more tags discovered by the robot) are added to a list data structure. Each location's associated pheromone value is decayed over time by the server; when a location's pheromone value has dropped below a threshold of 0.001, it is removed from the list. As each robot returns to the nest, the server randomly selects a location from the list (if available) and transmits it to the robot.

Our simulations are design to replicate the behavior of the robots and their experimental area. We measured the physical dimensions of the robots, their speed while traveling and searching, and the range over which their RFID reader can detect an RFID tag. We built the simulation with spatial dimensions that reproduce the properties of the robot, their 3-m radius experimental area, and the distribution of tags in this area. Like the real robots, simulated robots avoid collisions by turning to the right to move past other robots. We allow the simulated robots to search for tags for an amount of time equivalent to an hour, which we calibrated by the speed of the robots as they search and travel around the experimental area. In addition to simulating the 3-m radius area to which the physical robots were restricted, we also simulated the behavior of the robots in a much larger area in which movement is not restricted to 3 m of the nest, and tags are distributed in the same density but in such large numbers that even large swarms of robots collect only a fraction of the available tags. We simulated 1- and 3-robot swarms, and also scaled up to 30 and 100 robot swarms to observe the scaling properties of the system.

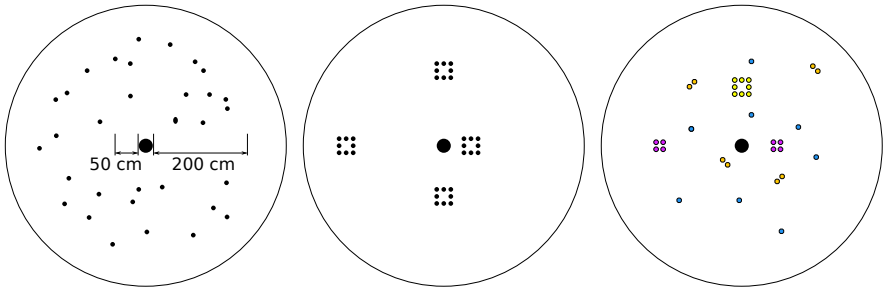
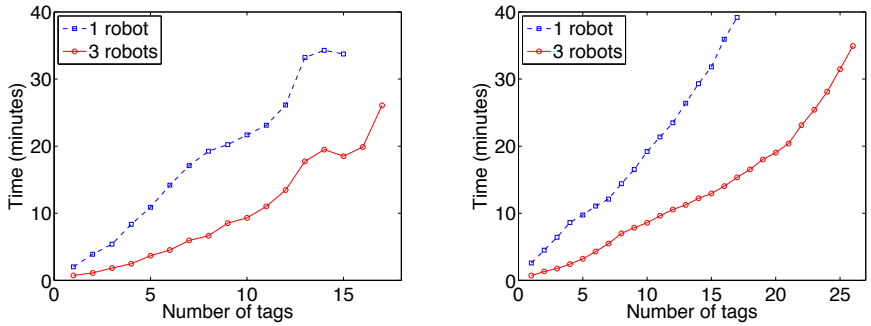


Fig. 3: 32 RFID tags layed out in (a) random, (b) clustered, and (c) power law distributions.

4 Results

We analyze the rates at which robots retrieve tags from each distribution, individually or in teams of three, in real robots and in simulation. Unless otherwise noted, result for each experimental treatment are averaged over five robot experiments and twenty experiments in simulation. Error bars indicate one standard deviation of the mean.

Time to collect 32 randomly distributed tags in 5 physical and 20 virtual experiments is shown in Figure 4. In robots and in simulation, three robots collect tags faster than one robot, however, the speedup varies over the course of the experiments (i.e., the red and blue lines are not parallel). When we average time to collect n tags, where n varies between 1 and the maximum number of tags collected, we find that 3 robots collect tags approximately twice as fast as 1 robot. The simulated experiments show slightly better scaling than the real robots. It is not surprising that simulated teams of 3 robots are faster than real teams of 3 robots because real robots have more difficulty with avoiding each other, physical hardware limitations, imperfect localization and the possibility that real robots confuse each other with the nest.



(a) Physical (b) Virtual

Fig. 4: Time to collect tags in a random distribution for one and three robots in physical (5 replicates) and virtual (20 replicates) experiments.

As more tags are found over the course of each experiment, it becomes increasingly difficult to find new, remaining tags, and the foraging rate sometimes decelerates. This ceiling effect limits our ability to observe differences between 1 and 3 robot teams. Figure 5 shows data from both physical and virtual experiments for one and three robots. We also analyzed time to collect 25% of the tags from the random, clustered, and power law distributions. We observe improved performance with three robots which collect 25% of the tags 2.8 times faster than one robot in the physical experiment and 2.3 times faster in the model.

Figure 6 illustrates the the rate of tag collection per minute of experiment time for physical and virtual swarms. Each bar denotes the collection rate for a swarm size over a particular tag distribution. This provides a normalized comparison between swarm sizes as well as distributions, regardless of overall experiment runtime which may vary between trials. We were not able to distinguish a significant effect of tag distribution on tag collection rate by the robots (General Linear Model [GLM]: $p > 0.1$; $n = 18$); but we did find a significant effect of distribution on tag collection rate using the larger sample size afforded by simulation (GLM: $p < 0.001$; $n = 120$). In the simulations, the greatest rate of tag collection was in the clustered distribution, followed by the power law distribution, followed by random. Note that this the reverse of the pattern with

respect to distribution in Figure 1, a result of greater overall density of tags in the robots' experimental area, and therefore greater ease of discovering piles, relative to the food densities used in our previous modeling work.

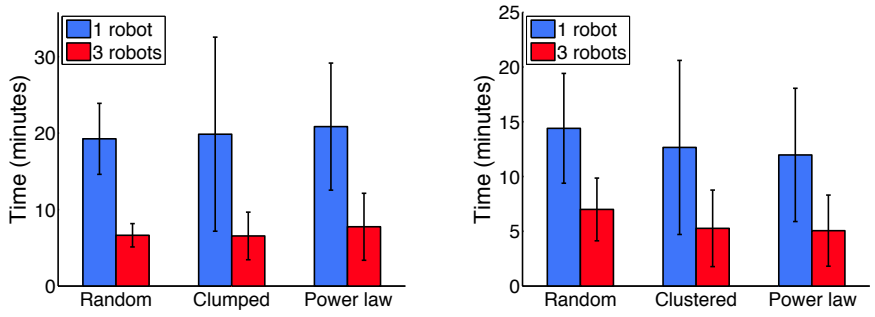
We look at the effects of pheromone trails on tag collection rate in Figure 7. Figure 7(a) compares physical and virtual results for three robots using pheromones while searching for tags in a power law distribution. Results from the physical experiment are averaged over three trials. Figures 7(b) and 7(c) show simulated results for 1, 3, 30, and 100 robots collecting power law distributed tags in an unbounded world.

5 Discussion

We have used Agent Based Models (ABM) and Genetic Algorithms (GA) to translate foraging behaviors of seed harvesting ants into algorithms for robotic swarms searching for RFID tags. We tested two sets of algorithms: one in which robots rely on individual memory of locations of previously found tags (mimicking site fidelity), and one in which robots communicate locations of previously found tags (mimicking pheromones) as waypoints to a central server that acts as the robots' nest'. We tested each approach in single robots and teams of 3 robots, and observed that 3 robots find tags approximately twice as fast as 1 robot when using site fidelity. Pheromone-like communication improves foraging success robots in simulation. We did not observe that pheromones improved foraging in real robots, but in addition to small sample size, we attribute the lack of success primarily to errors that were propagated by miscommunication. Pheromones decrease performance when robots get lost and communicate incorrect locations to other robots. In simulation we found that pheromones improved foraging in 3-robot teams by 10% to 50% (depending on distribution) over site fidelity alone. Additionally, in simulation, the combination of pheromones and site fidelity provided an approach that is scalable to swarms of 100 robots. We suspect that improving the robots' ability to navigate will reduce this problem. The close correspondence between simulation and real robots in smaller swarms make us optimistic that these results could be replicated in large robot swarms.

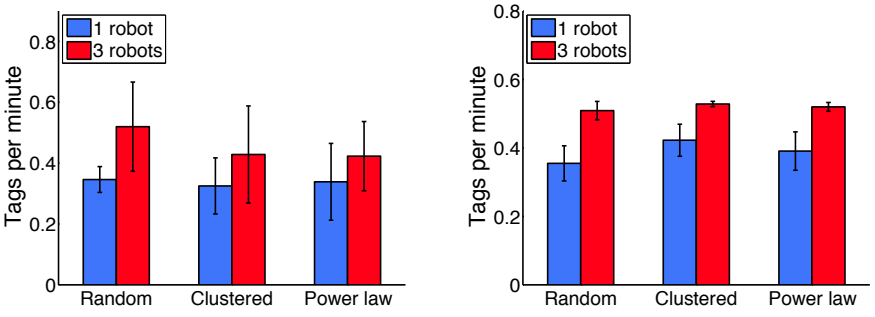
As in the ants, we found that site fidelity is an effective strategy for foraging. This behavior has several benefits. First, it is extremely simple and easily encoded into very simple devices, including devices much simpler than the robots we used here. Second, the approach is highly parallelizable because it requires no communication among robots. Third, it leads to effective and small teams.

Our simulations of ants and our simulations of robots show that adding pheromone communication increases foraging success, particularly on clustered distributions (Figs. 1, 6). We demonstrated that it is possible to implement pheromone communication in robots by having robots report the location where they found a tag to a central server if the robot saw at least 2 additional tags in the vicinity. The server then implements a simple pheromone algorithm and reports those locations to other robots. When we add this pheromone-like behavior to our robots, we observe robots clearing large clusters of tags faster. Simulations show more success with pheromones because simulated ants don't get lost or miscommunicate. Simulations suggest that this approach is highly scalable. When we scale up to 100 robots, each robot is about half as efficient as a single robot, meaning that teams of 100 robots collect resources 50 times faster than a single robot (Fig. 7). This per-robot decline is largely due to the



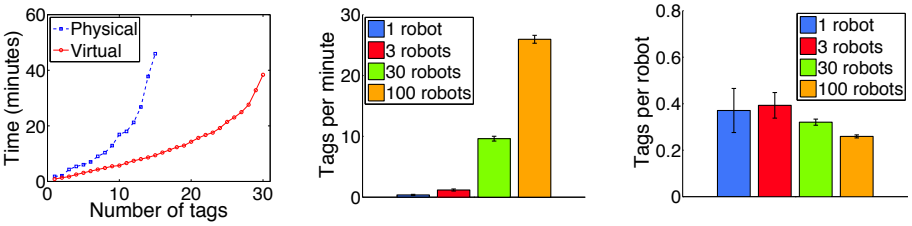
(a) Physical (b) Virtual

Fig. 5: Time to collect 25% of the tags from three different distributions for one and three robots.



(a) Physical (b) Virtual

Fig. 6: Rate of tag discovery calculated as total tags found normalized by experiment length in minutes.



(a) (b) (c)

Fig. 7: Effects of using pheromone trails on tag collection

increased distance the simulated robots travel–an unavoidable consequence of central place foraging.

Our results suggest that the approach of combining individual memory with communication at a central nest can transform simple robots into effective swarms that are scalable and robust to the loss or malfunction of a few individuals. Results of our 3 robot experiments include several instances in which one robot became lost or malfunctioned, but the other two robots continued their task. Such systems could be used for search and rescue, searching for resources or obstacles, and even biomedical applications using nano-robots. Our approach, similar to the approach by [7] helps to lay a foundation to further explore the interplay between simulation and experiments with real robots. Our next steps are to use GAs to optimize parameters that lead to maximum efficiency and/or robustness in the ABM, and then import those parameters into the robots. For example, currently the robots' turning angles during their random walk are based on a rough approximation of how our simulated ants evolved to forage from GAs. In future work, we will take the same approach and evolve optimal parameters given the physical attributes of the robots already encoded in the ABM. We will also evolve parameters to determine the optimal balance between reliance on individual memory versus pheromone communication. We will extend our analysis to different kinds of resource distributions, including ones that may be dynamic by encoding tags with resources that appear and disappear over time. Finally, this work will be extended by simulating and replicating in our robots, features of some large ant colonies–the use of mobile nests (as exemplified by army ants) and the use of multiple nests (as exemplified by invasive argentine ants).

References

1. T. Bailey and H. Durrant-Whyte. Simultaneous localization and mapping (SLAM): Part II. *Robotics & Automation Magazine, IEEE*, 13(3):108–117, 2006.
2. S. Banerjee and M. Moses. 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.
3. B. Beverly, H. McLendon, S. Nacu, S. Holmes, and D. Gordon. How site fidelity leads to individual differences in the foraging activity of harvester ants. *Behavioral Ecology*, 20(3):633–638, 2009.
4. E. Bonabeau, M. Dorigo, and G. Theraulaz. *Swarm intelligence: from natural to artificial systems*. Oxford University Press, USA, 1999.
5. Y. Cao, A. Fukunaga, and A. Kahng. Cooperative mobile robotics: Antecedents and directions. *Autonomous robots*, 4(1):7–27, 1997.
6. M. Dorigo, M. Birattari, et al. Swarmanoid, the movie. In *AAAI-11 Video Proceedings*. AAAI Press, 2011. Winner of the "AAAI-2011 Best AI Video Award".
7. M. Dorigo, D. Floreano, et al. Swarmanoid: a novel concept for the study of heterogeneous robotic swarms. Technical report, Technical Report TR/IRIDIA/2011-014, IRIDIA, Université Libre de Bruxelles, Brussels, Belgium, 2011.
8. M. Dorigo and E. Sahin. Swarm robotics–special issue editorial. *Autonomous Robots*, 17(2-3):111–113, 2004.
9. M. Dorigo, V. Trianni, E. Şahin, R. Groß, T. Labella, G. Baldassarre, S. Nolfi, J. Deneubourg, F. Mondada, D. Floreano, et al. Evolving self-organizing behaviors for a swarm-bot. *Autonomous Robots*, 17(2):223–245, 2004.
10. H. Durrant-Whyte and T. Bailey. Simultaneous localization and mapping: part I. *Robotics & Automation Magazine, IEEE*, 13(2):99–110, 2006.
11. T. Flanagan, K. Letendre, W. Burnside, G. Fricke, and M. Moses. How Ants Turn Information into Food. *Proceedings of the 2011 IEEE Conference on Artificial Life*, pages 178–185, 2011.

12. T. Flanagan, K. Letendre, W. Burnside, G. Fricke, and M. Moses. How Ants Turn Information into Food. *Proc. of the 2011 IEEE Conf. on ALife*, 2011.
13. T. Flanagan, K. Letendre, and M. E. Moses. Quantifying the Effect of Colony Size and Food Distribution on Harvester Ant Foraging. *PLoS ONE*, in review.
14. D. Goldberg. *Genetic algorithms in search, optimization, and machine learning*. Addison-Wesley Professional, 1989.
15. D. Gordon. The spatial scale of seed collection by harvester ants. *Oecologia*, 95(4):479–487, 1993.
16. B. Hölldobler. Recruitment behavior, home range orientation and territoriality in harvester ants, Pogonomyrmex. *Behav. Ecol. and Sociobio.*, 1(1):3–44, 1976.
17. M. Krieger, J. Billeter, and L. Keller. Ant-like task allocation and recruitment in cooperative robots. *Nature*, 406:992–995, 2000.
18. K. Letendre and M. E. Moses. Ant foraging strategies: Site fidelity and recruitment alone and in combination. in review.
19. R. Mayet, J. Roberz, T. Schmickl, and K. Crailsheim. Antbots: A feasible visual emulation of pheromone trails for swarm robots. *Swarm Intell.*, pages 84–94, 2011.
20. C. Moeslinger, T. Schmickl, and K. Crailsheim. Emergent flocking with low-end swarm robots. *Swarm Intelligence*, pages 424–431, 2011.
21. F. Mondada, G. Pettinaro, I. Kwee, A. Guignard, L. Gambardella, D. Floreano, S. Nolfi, J. Deneubourg, and M. Dorigo. SWARM-BOT: A swarm of autonomous mobile robots with self-assembling capabilities. In *Proc. of the Intl. Workshop on Self-organisation and Evolution of Social Behaviour*, pages 307–312, 2002.
22. M. Montemerlo, S. Thrun, D. Koller, and B. Wegbreit. FastSLAM: A factored solution to the simultaneous localization and mapping problem. In *Proceedings of the National conference on Artificial Intelligence*, pages 593–598. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2002.
23. M. Moses. *Metabolic scaling from individuals to societies*. PhD thesis, University of New Mexico, 2005.
24. M. Moses and S. Banerjee. Biologically Inspired Design Principles for Scalable, Robust, Adaptive, Decentralized Search and Automated Response (RADAR). *Proceedings of the 2011 IEEE Conference on Artificial Life*, pages 30–37, 2011.
25. J. Mull and J. MacMahon. Spatial variation in rates of seed removal by harvester ants (Pogonomyrmex occidentalis) in a shrub-steppe ecosystem. *American Midland Naturalist*, pages 1–13, 1997.
26. S. Nolfi and D. Florin. *Evolutionary robotics: The biology, intelligence, and technology of self-organizing machines*. MIT Press, 2000.
27. L. Parker. Designing control laws for cooperative agent teams. In *Robotics and Automation, 1993. Proceedings., 1993 IEEE International Conference on*, pages 582–587. IEEE, 1993.
28. A. Sharkey. Robots, insects and swarm intelligence. *Artificial Intelligence Review*, 26(4):255–268, 2006.
29. R. Solé, E. Bonabeau, J. Delgado, P. Fernández, and J. Marín. Pattern formation and optimization in army ant raids. *Artificial Life*, 6(3):219–226, 2000.
30. A. Stranieri, E. Ferrante, A. E. Turgut, V. Trianni, C. Pinciroli, M. Birattari, and M. Dorigo. Self-organized flocking with a heterogeneous mobile robot swarm. In T. Lenaerts, M. Giacobini, H. Bersini, P. Bourgine, M. Dorigo, and R. Doursat, editors, *Advances in Artificial Life, ECAL 2011: 11th European Conference on the Synthesis and Simulation of Living Systems*, pages 789–796. MIT Press, Cambridge, MA, 2011.
31. V. Trianni and S. Nolfi. Engineering the Evolution of Self-Organizing Behaviors in Swarm Robotics: A Case Study. *Artificial Life*, 17(3):183–202, 2011.
32. R. Vaughan, K. Stoy, G. Sukhatme, and M. Mataric. LOST: Localization-space trails for robot teams. *IEEE Robotics and Automation*, 18(5):796–812, 2002.
33. M. Wittlinger, R. Wehner, and H. Wolf. The ant odometer: stepping on stilts and stumps. *Science*, 312(5782):1965, 2006.

Ovarian cancer relapse: micro-carcinomas vary in form with peritoneal niche

Kimberly Kanigel-Winner[1], Mara Steinkamp[1], Suzy Davies[1], Abbas Shirinifard[2], Yi Jiang[3], and Bridget S. Wilson[1]

ü

***Short Abstract* — In ovarian cancer, the morphology of microscopic tumors depends on local characteristics of tissues to which cells initially attach in the peritoneal cavity. We use an integrated experimental and modeling approach to study tumor growth during cancer relapse, incorporating data from a mouse xenograft model into a cellular Potts model. Simulations include tumor spheroid attachment to organ surfaces in the abdominal cavity, followed by chemotactic invasion where permitted by the features underlying the mesothelial layer at different sites [1]. The *in silico* model also includes the essential features of angiogenesis; oxygen gradient fields indicate that new blood vessel formation is not dependent on the tumor mass reaching a hypoxic state.**

Keywords — **ovarian cancer relapse, cellular Potts, SKOV3-IP1, Compucell3d, *in silico* model, cellular automaton**

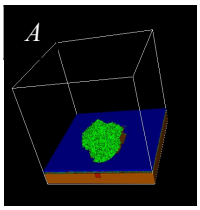
I. Purpose

In ovarian cancer, the majority of patients are not diagnosed until surgery is needed to remove large tumor masses. Following surgical debulking and chemotherapy, there is significant risk of relapse due to chemoresistant tumor cells remaining in the peritoneal cavity [2]. In our xenograft model, human ovarian tumor cells (SKOV3.ip-GFP) are injected into the peritoneum of immunocompromised mice. Tumors develop within a few weeks. We note that the morphology and angiogenesis potential of new micro-tumors is highly dependent on local physical and chemical characteristics of tissues to which they attach in the peritoneal cavity. We postulate that these features have the potential to determine the local efficacy of specific classes of cancer therapeutics (*i.e.* small molecules vs. protein-based therapies such as monoclonal antibodies). These hypotheses will be explored using mathematical models that consider local penetrance as well as route of delivery.

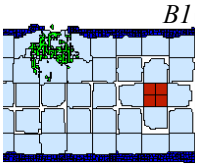
II. Methods

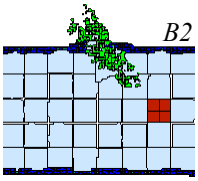
Data from the mouse model was used to parameterize mesoscopic cellular Potts models (using Compucell3d [3]) of micro-tumor morphologies on mesothelium overlying muscle or attached to the mesentery (a dual mesothelial membrane containing vascular bundles surrounded by fat). We incorporate tumor cell growth, cell division, chemotaxis, invasion, and O_2 and glucose consumption. When possible, we included experimental values directly in the simulation, such as adipocyte secretion of the chemotatic factor IL-8 [4], concentrations of O_2, glucose, and IL-8 in blood and peritoneal fluid, and normal cell volumes (such as of adipocytes) in our mice. Otherwise, model parameters were systematically tuned to re-create experimental values, such as rate of invasion/chemotaxis of ovarian cancer cells into mesothelium [5]. We compared simulation results to our mouse model of ovarian cancer peritoneal metastasis.

III. Results & Conclusions

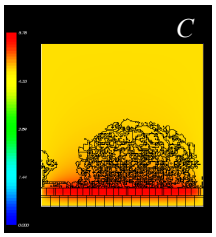


A. Models of spheroids attached to the mesothelium where the underlying tissue is characterized by "tight" cellular junctions with no space between cells (such as at the surface of the small intestine (image *A*)) generate a non-invasive semi-spheroidal morphology in small tumors, both *in silico* and *in vivo*.





B. A chemotactic chemical gradient originating from adipocytes generates a tumor growth pattern (*B1*) different from that created when dual signals from adipocytes and vessels are included (*B2*; images are 2-D sections of 3-D simulations.). Similarities to the dual-signal model are seen in GFP tumors in the mesentery of the xenografted mice. We will continue to explore this hypothesis using our mouse xenograft model.



C. Simulations show that tumors ~30 cells wide, comparable to those excised from mice at 1 week, are small enough that all cells are sufficiently oxygenated (image *C*: cell consumption of O_2 does not create a hypoxic area at the center of the spheroid; red = maxO_2). Nevertheless, tumors 1 and 3 weeks old are fully vascularized. Our microarray data also shows that these cells constitutively express the angiogenic factor VEGF, and upregulate another, Ang2, in tumors. We are building simulations of hypoxia-independent tumor angiogenesis and will perform morphometric analysis on simulated tumor vasculature patterns for comparison to the uniform vasculature seen *in vivo*.

These models lay the foundation for modeling tumor cell death after the delivery of different classes of drugs via either intravascular or intraperitoneal injection. Results of

ü Acknowledgements: This work was funded by NIH R01 CA119232 and aligned with the NM Center for Spatiotemporal Modeling.

[1]Pathology Department, University of New Mexico, Albuquerque, NM. E-mail: bwilson@salud.unm.edu, E-mail: kkanigel@unm.edu

[2]Biocomplexity Institute, University of Indiana. E-mail: ashirini@indiana.edu

[3]Department of Math and Statistics, Georgia State University. E-mail: jiang@lanl.gov

these models will guide the design of preclinical trials in mice engrafted with ovarian tumor cells.

REFERENCES

[1] Steinkamp, Mara, *et al*. "Ovarian tumor attachment, invasion and vascularization reflect unique microenvironments in the peritoneum: Insights from xenograft and mathematical models," in preparation.

[2] Shield, Kristy, M Leigh Ackland, Nuzhat Ahmed, and Gregory E Rice. "Multicellular spheroids in ovarian cancer metastases: Biology and pathology." *Gynecologic oncology* 113, no. 1 (April 2009): 143-8.

[3] http://compucell3d.org

[4] Iwanicki, M. P., *et al.* (2011) Ovarian Cancer Spheroids Use Myosin-Generated Force to Clear the Mesothelium. *Cancer Discovery*. July 2011, 1:144-157.

[5] Henrichot, Elvire, *et al*. (2005) Production of chemokines by perivascular adipose tissue: a role in the pathogenesis of atherosclerosis? *Arteriosclerosis, thrombosis, and vascular biology* 25 (12) (December 1): 2594-9.

Breaking the $O(nm)$ Bit Barrier: Secure Multiparty Computation with a Static Adversary

Varsha Dani
University of New Mexico

Valerie King
University of Victoria

Mahnush Movahedi
University of New Mexico

Jared Saia
University of New Mexico

Abstract

We describe scalable algorithms for secure multiparty computation (SMPC). We assume a synchronous message passing communication model, but unlike most related work, we do not assume the existence of a broadcast channel. Our main result holds for the case where there are n players, of which a $1/3 - \epsilon$ fraction are controlled by an adversary, for ϵ any positive constant. We describe a SMPC algorithm for this model that requires each player to send $\tilde{\mathrm{O}}(\frac{n+m}{n} + \sqrt{n})$ messages and perform $\tilde{\mathrm{O}}(\frac{n+m}{n} + \sqrt{n})$ computations to compute any function f, where m is the size of a circuit to compute f. We also consider a model where all players are selfish but rational. In this model, we describe a Nash equilibrium protocol that solve SMPC and requires each player to send $\tilde{\mathrm{O}}(\frac{n+m}{n})$ messages and perform $\tilde{\mathrm{O}}(\frac{n+m}{n})$ computations. These results significantly improve over past results for SMPC which require each player to send a number of bits and perform a number of computations that is $\theta(nm)$.

1 Introduction

In 1982, Andrew Yao posed a problem that has significantly impacted the weltanschauung of computer security research [22]. Two millionaires want to determine who is wealthiest; however, neither wants to reveal any additional information about their wealth. Can we design a protocol to allow both millionaires to determine who is wealthiest?

This problem is an example of the celebrated *secure multiparty computation (SMPC)* problem. In this problem, n players each have a private input, and their goal is to compute the value of a n-ary function, f, over its inputs, without revealing any information about the inputs. The problem is complicated by the fact that a hidden subset of the players are controlled by an adversary that actively tries to subvert this goal.

SMPC abstracts numerous important problems in distributed security, and so, not surprisingly, there have been thousands of papers written in the last several decades addressing this problem. However, there is a striking barrier that prevents wide-spread use: current algorithms to solve SMPC are not resource efficient. In particular, if there are n players involved in the computation and the function f can be computed by a circuit with m gates, then most algorithms require each player to send a number of messages and perform a number of computations that is $\Omega(mn)$ (see, for example, [11, 12, 5, 2, 15, 10, 16, 17, 3]).

Recent years have seen exciting improvements in the *amortized* cost of SMPC, where the number of messages and total computation done per player can be significantly better than $\Theta(mn)$ [7, 9, 8]. However, the results for these algorithm hold only in the amortized case where m is much larger than n, and all of them have additional additive terms that are large polynomials in n (e.g. n^6). Thus, there is still a strong need for SMPC algorithms that are efficient in both n and m.

1.1 Formal Problem Statement

We now formally define the SMPC problem. As previously stated, there are n players, and each player i has a private input, x_i. Further there is a n-ary function f that is known to all players. The goal is to to ensure that: 1) all players learn the value $f(x_1, x_2, \ldots, x_n)$; and 2) the inputs remain as private as possible: each player i learns nothing about the private inputs other than what is revealed by $f(x_1, x_2, \ldots, x_n)$ and x_i.

The main complication is the fact that up to a 1/3 fraction of the players are assumed to be controlled by an adversary that is actively trying to prevent the computation of the function. We will say that the players controlled by the adversary are *bad* and that the remaining players are *good*. The adversary is *static*, meaning that it must select the set of bad players at the start of the algorithm. A careful reader may ask: How can we even define the problem if the bad players control their own inputs to the function and thereby can exert control over the output of f?

The answer to this question is given by Figure 1. In the left illustration in this figure, there are 5 players that are trying to compute a function over their 5 private inputs. If there is a trusted external party, as shown in the center of this illustration, the problem would be easy: each player sends their input to this trusted party, the party performs the computation, and then sends the output of f back to all the players. In essence, this is the situation we want to simulate with our SMPC algorithm. The right illustration of Figure 1 shows this goal: the SMPC algorithm simulates the trusted party. In particular, we allow all players, both good and bad, to submit a single input to the SMPC algorithm. The SMPC algorithm then computes the function f based on all of these submitted inputs, and sends the output of f back to all players.

This problem formulation is quite powerful. If f returns the input that is in the majority, then

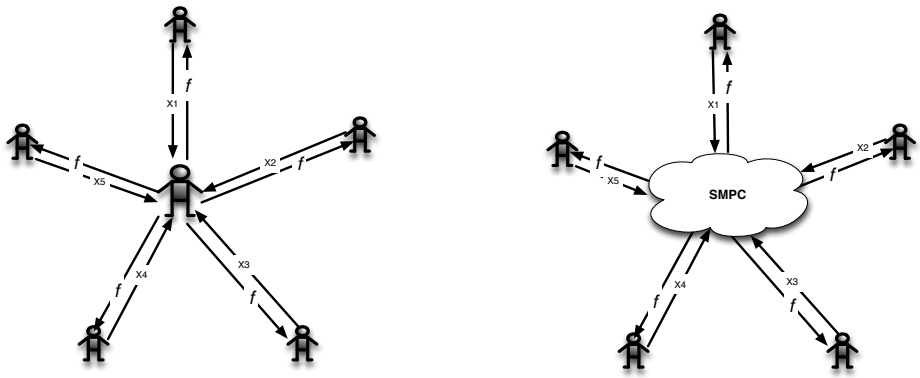


Figure 1: Schematic of SMPC problem

SMPC enables voting. If f returns a tuple containing 1) the index of the highest variable; and 2) the value of the second highest variable, then SMPC enables a simple Vickrey auction, i.e. the highest bidder wins and pays the second highest bid. If f returns the output of a digital signing function, where the private key equals the sum of all player inputs modulo Z_p for some prime p, then SMPC enables group digital signatures, i.e. the entire group can sign a document, but no individual player learns the secret key. In short, the only limitation is determined by whether or not the function f is computable.

Our communication model is as follows. We assume there is a private and authenticated communication channel between ever pair of players. However, we assume that the adversary is computationally unbounded, and so make no cryptographic hardness assumptions.[1] Also, unlike much past work on SMPC, we do *not* assume the existence of a public broadcast channel. Finally, we note that we assume that the good players strictly follow the protocol, and thus do not form coalitions in which information is exchanged (i.e. there are no so called "gossiping" good players).[2]

1.2 Our Results

The main result of this paper is as follows.

Theorem 1.1. Assume there are n players, no more than a 1/3 fraction of which are bad, and a n-ary function f that can be computed using m gates. Then if the good players follow Algorithm 1, with high probability, they can solve SMPC, while ensuring:

1. Each player sends at most $\tilde{\mathrm{O}}(\frac{n+m}{n} + \sqrt{n})$ messages,
2. Each player performs $\tilde{\mathrm{O}}(\frac{n+m}{n} + \sqrt{n})$ computations.

An additional result of this paper deals with the situation where all players are selfish but rational. Our precise assumption for the rational players is as follows. The rational players' utility functions are such that they prefer to learn the output of the function, but also prefer that other players not learn the output. Following previous work on rational secret sharing [19, 13, 14, 20], we assume all the players have the same utility function, which is specified by constants U_k where

[1] In the cryptographic community, this is frequently called *unconditional security*.

[2] Technically, we can maintain privacy, even with a certain amount of information exchange among the good players. See Section 3 for details.

k is the number of players who learn the output. Here U_1 is the utility to a player if she alone learns the output, U_n is the utility if she learns the secret and all other players learn it as well, and finally U_- is the utility when the player does not learn the output. We further assume that $U_1 > U_2 > \ldots > U_n > U_-$, so that the players' preferences are strict.

A key goal is to design a protocol that is a *Nash equilibrium* in the sense that no player can improve their utility by deviating from the protocol, given that all other players are following the protocol. Our main result in this model is the following.

Theorem 1.2. Assume there are n players and each player is rational with utility function given as above. Then there exists a protocol (see Section 2.5) such that 1) it is a Nash equilibrium for all players to run this protocol and 2) when all players run the protocol then, with high probability, they solve SMPC, while ensuring:

1. Each player sends at most $\tilde{\mathrm{O}}(\frac{n+m}{n})$ messages,
2. Each player performs $\tilde{\mathrm{O}}(\frac{n+m}{n})$ computations.

The rest of this paper is organized as follows. In Section 2, we describe our algorithms for scalable SMPC. We first present the case with an adversary, and then in Section 2.5, describe the changes that are needed to handle the case where all players are rational. The proofs of correctness for our algorithms are in Section 3. We conclude and give problems for future work in Section 4.

2 Our Algorithm

We now describe the algorithm that achieves our result for the model where players are either good or bad (Theorem 1.1).

The main idea behind reducing the amount of communication required for the computation is that rather than having each player communicate with all the other players, we will subdivide the players into groups called *quorums* of logarithmic size. The players in each group will communicate only with members of their own group and members of certain other groups. The number of other groups a particular group is required to communicate with is a function of the circuit size.

If any single quorum were to have too many bad players then they could severely disrupt the computation, so the subdivision must spread the bad players around so that the fraction of bad players in each quorum remains less than 1/3. We will call a quorum *good* if more than two thirds of the players in it are good. How are these quorums to be formed? This would be easy to achieve (with high probability) if there were a trusted mediator who could form the groups randomly and assign each player to their group. In the absence of a mediator, the players must achieve this subdivision themselves. To do this, we appeal to the following result of King, Lonergan, Saia and Trehan [18].

Theorem 2.1 (Theorem 2 of [18])**.** Let n be the number of processors in a fully asynchronous full information message passing model with a static adversary. Assume that there are at least $(2/3 + \epsilon)n$ good processors. Then for any positive constant ϵ, there exists a protocol which w.h.p. brings all good processors to agreement on n good quorums; runs in polylogarithmic time; and uses $\tilde{\mathrm{O}}(\sqrt{n})$ bits of communication per processor. If all players are rational the algorithm runs in polylogarithmic time; and uses $\tilde{\mathrm{O}}(1)$ bits of communication per processor.

We will also require certain primitives for multiparty protocols. A 1/3 fault-tolerant Verifiable Secret Sharing scheme for k players, henceforth VSS(k), is an algorithm for a dealer to deal shares of a secret which he holds to the players, such that (1) no set of fewer than a third of the players

can get any information about the secret and (2) the secret can be reconstructed from the shares even if upto a third of them are missing or corrupted (*i.e.* if upto a third of the players are bad) Moreover, the players then run a verification protocol, at the end of which the good players either agree that a valid secret has been shared or agree to disqualify the dealer (if he did not deal shares consistent with any secret) and take the secret to be a preset default value. Such a sharing and verification scheme is described in the work of Ben-Or, Goldwasser and Wigderson (BGW) [4]. This uses a constant number of rounds of communication and has zero probability of error. [3]

BGW [4] also describe an errorless protocol for SMPC that tolerates up to a third of the players being bad. (See BGW [4] Theorem 3). The number of rounds of communication depends at most linearly on the size of the circuit being computed.

We will make extensive black box use of both these primitives in our algorithm. Note that these protocols involve all-to-all communication amongst the players. For this reason we will refer to the SMPC primitive as HEAVYWEIGHT-SMPC. However, in our protocol, at most three quorums will be involved in any given run of the black box SMPC or VSS. Thus the amount of communication per run of HEAVYWEIGHT-SMPC will have only a polylogarithmic dependence on n.

We also note that the VSS and SMPC primitives require broadcast channels in addition to secure private channels. Within our quorums, we simulate broadcast channels using Byzantine Agreement to decide whether the same message was sent to everyone. Since the quorum size is just logarithmic, we can use any polynomial time and algorithm for Byzantine agreement, such as Bracha's protocol [6].

We assume that all computations are done over a finite field $\mathbb{F}$. It is convenient for presentation to assume that all gates have in degree 2 and out degree at most 2, but we can tolerate arbitrary constant degrees.

2.1 Setup

We first give a high level description of our algorithm. The first step is to build a network, which we call G, based on the circuit C. For every gate in the circuit C, there will be a node in G, which we will refer to as an internal node. In addition, G will have n extra nodes, corresponding to the n inputs of C. We will call these input nodes. For every wire from some gate to another gate in C, there will be an edge connecting the corresponding nodes in G. Further, for every wire from an input to a gate in C, there will be an edge from the corresponding input node to the corresponding internal node in G.

The players use the algorithm from Theorem 2.1 to divide themselves into n quorums each of size $\theta(\log n)$. Each quorum is assigned to some node in G. Recall we have $m + n$ nodes in G. We assume a canonical numbering of the nodes in G and of the n quorums, and we assign the quorum numbered i to to any node with number j s.t. $(j \mod n) = i$. Note that each quorum is thus assigned to at most $\lceil \frac{m+n}{n} \rceil$ nodes.

2.2 Extended Example

We now work through an extended example of our algorithm. The formal description of the algorithm is given in Section 2.4.

Figure 2 illustrates the example we will use to describe our algorithm. The top left illustration

[3] This scheme uses error correcting codes to achieve the verification. Other such schemes exist, which use Zero Knowledge proof techniques for verification and can tolerate up to half the players being faulty; see [21]. These, however, have an exponentially small but positive probability of error.

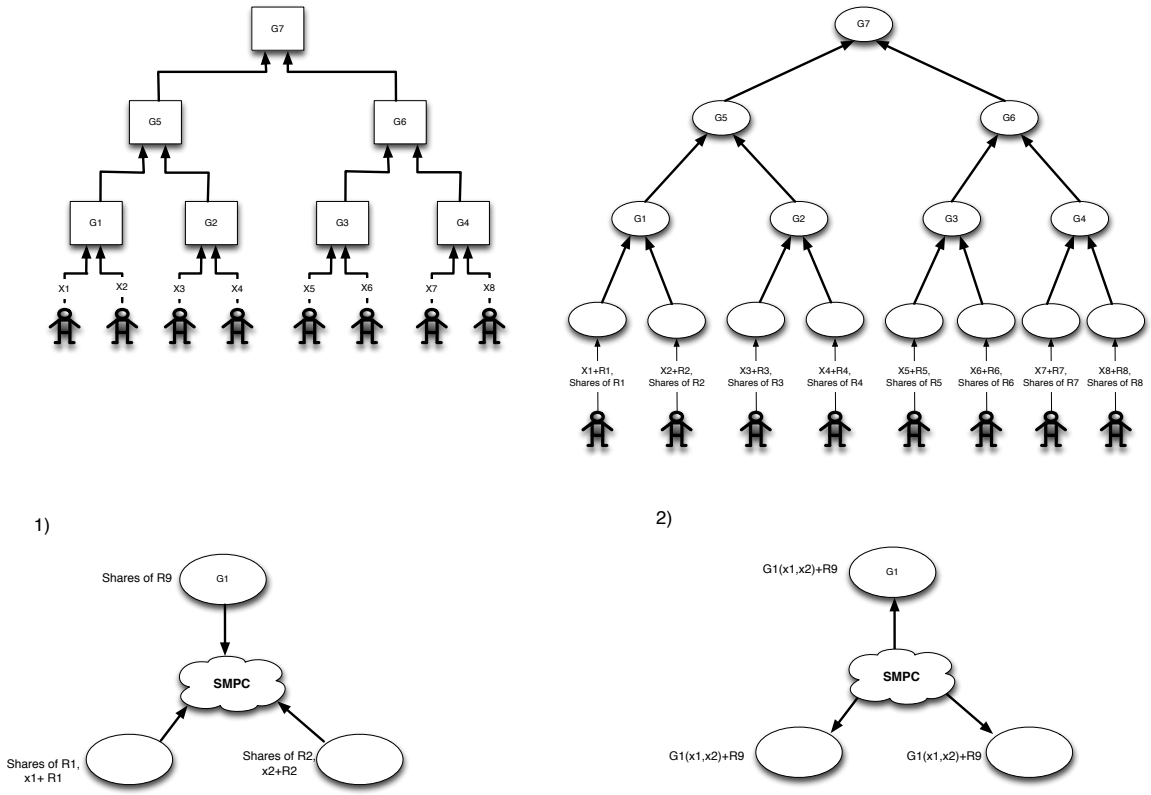


Figure 2: Example of Quorum based SMPC

in this figure describes a simple circuit with $m = 7$ gates and $n = 8$ inputs. For simplicity, this circuit is small; in a real application, we would expect both m and n to be much larger. Also, for simplicity, in this example, the circuit is a tree; however, our algorithm works for an arbitrary circuit. The circuit in this example computes an 8-ary function, f, that we want to compute in our SMPC. The gates have labels $G1, \ldots, G8$, that represent the functions computed by each gate. Each of the players at the bottom sends its input to some gate.

The top right illustration in the figure shows the layout of the quorums based on this circuit. Each oval in this illustration represents a quorum. There are $n + m$ ovals, m for each gate in the circuit and n for each player. Recall that using the algorithm from Theorem 2.1, we can create n quorums with the properties that 1) each quorum contains less than a $1/3$ fraction of bad players; 2) each quorum contains $\theta(\log n)$ players; and 3) each player is in $\theta(\log n)$ quorums. We will map these n quorums to the $m + n$ ovals. It will be the case that the number of ovals is larger than the number of actual quorums, requiring us to map some quorums to multiple ovals. However, each quorum will be mapped to at most $\lceil (n + m)/n \rceil$ ovals. Moreover, as we will see, it will not cause problems even if we map the same quorum to neighboring ovals. The algorithm begins by getting inputs from the players. In this illustration, each player i computes a value R_i selected uniformly at random from all values in the field $\mathbb{F}$. It then computes $x_i + R_i$, the value of its private input plus R_i and sends this value to all players in the quorum above it. Note that R_i "protects" the value x_i since $x_i + R_i$ is distributed completely uniformly at random. Finally, player i uses the

verifiable secret sharing (VSS) algorithm from [4] to create shares of R_i, and to send one share to each player in the quorum above. These shares have the property that any 2/3 fraction of them can be used to reveal the value R_i, but less than a 2/3 fraction reveals no information about R_i.

The two illustrations in the bottom part of the figure show how three quorums compute the output of each gate. We wish to maintain the following invariant: the value computed at any oval is the value that would be computed at the corresponding gate of C, masked by a random element of the field. The mask is jointly reconstructed by sufficiently many players at the oval, but it is not known to any individual player. For simplicity, these bottom illustrations focus solely on the computation occurring for $G1$; similar computations occur for all the other gates. Three quorums are involved in the computation for G1: the two bottom quorums provide the randomized inputs, and the top quorum provides a value ($R9$) that is used to randomize the output.

The bottom left illustration shows what is known at each quorum before the computation of $G1$. All players in the bottom left quorum know the value $x_1 + R_1$. Moreover, each player in this quorum has a share of the value R_1. These shares again have a 2/3 threshold property: any 2/3 fraction of them can be used to reconstruct R_1, but any set of less than 2/3 of them reveals no information about R_1. The players in the bottom right quorum have similarly knowledge: they all know $x_2 + R_2$ and they each have shares of R_2 with a 2/3 threshold property. Finally, the players in the top quorum have previously run a simple distributed algorithm to ensure that they each have a share of a value, R_9 that is selected uniformly at random from the field $\mathbb{F}$. These shares of R_9 are constructed with the 2/3 threshold property; this property can be ensured done by repeated applications of the VSS algorithm from [4]. Finally, the players in all three quorums use HEAVYWEIGHT-SMPC to compute the value $G_1(x_1, x_2) + R_9$.

We note two important facts about this SMPC. First, the inputs (x_1+R_1, x_2+R_2, shares of R_1, shares of R_2, shares of R_9) contain enough information to compute the value $G_1(x_1, x_2)+R_9$ in the SMPC, even if the bad players lie about their inputs. Second, the SMPC algorithm occurs over only $\theta(\log n)$ players, so even a heavyweight protocol which runs in time and message cost polynomial in the number of players will incur latency and message costs that are just polylogarithmic in n.

The bottom right illustration shows the result after the computation of G_1. Each player in the three quorums has learned the value $G_1(x_1, x_2) + R_9$. Note that no player at any of the three quorums has (individually) learned any information about the value $G_1(x_1, x_2)$, since the mask R_9 which no individual knows, is uniformly random, and hence the computed value, $G_1(x_1, x_2) + R_9$ is also uniformly random over the field. In addition, note that we now have a situation for the top quorum where 1) every player knows the output value plus a random element R_9; and 2) the shares of R_9 are distributed among the players in such a way that the value R_9 can be reconstructed if and only if the good players in the quorum send the shares to each other. Thus, the top quorum is in the same situation now with respect to the value $G_1(x_1, x_2)$ as the bottom quorums were in with respect to x_1 and x_2 previously. Hence, the same procedure can be repeated as compute the values for the gates in the next layer of the circuit.

2.3 Some Details

The output of the quorum associated with the root node in G is the output of the entire algorithm. The last step of the algorithm is to send this output to all players. To do that, we construct a complete binary tree using the n quorums, with root quorum equal to the quorum that knows the output of the circuit. We then use majority filtering to pass the output down to all the players. Specifically, when a player receives the output message from all players in its parent quorum, it

computes the majority of all messages, and considers the majority of the messages as his correct output; then, it sends the output to all players in any quorums below.

Note that it may be the case that a player p participates $k > 1$ times in the quorums performing HEAVYWEIGHT-SMPC in Figure 2. In such a case, we allow p to play the role of k different players in the SMPC, one for each quorum to which p belongs. This ensures that the fraction of bad players in the heavy-weight SMPC is always less than 1/3. Also, the heavy-weight SMPC protocol of [21] maintains privacy guarantees even in the face of gossiping coalitions of constant size. Thus, p will learn no information beyond the output and its own inputs after running this protocol.

We observe that the output of the last node of G is the output of the algorithm. The last step of the algorithm is to send the output to all players. To do that, players reuse their quorums and build a complete binary tree with n nodes and assign quorum i to node i in the tree. Each player receives the output message from all players in its parent node and considers the majority of the messages as its correct output. Then, it sends the output to all players of its children nodes.

Finally, note that in this algorithm, each player participates in $\theta(\log n)$ quorums; each quorum is responsible for at most $\lceil (n+m)/n \rceil$ ovals; and the SMPC performed at an oval has resource cost which is polylogarithmic in n. Moreover, each player runs the VSS algorithm to send its input to a single quorum initially. Thus, in this algorithm, each player sends $\tilde{\mathrm{O}}(\frac{n+m}{n})$ bits and is involved in the computation of $\tilde{\mathrm{O}}(\frac{n+m}{n})$ gates.

2.4 Formal Description

We assume that the function to be computed is presented as a circuit C with c gates, numbered $1, 2, \ldots, m$, where the gate numbered 1 is the output gate. The high level picture of the comminication network is a directed graph G, with $c+n$ nodes numbered $1, 2, \ldots c+n$. The first c of these are "gate nodes", node i corresponding with gate i of the circuit, and there are edges between pairs of them whenever the corresponding pair of gates is connected by a wire. The direction of the edge is the direction of flow of computation in the circuit C. Note that the node numbered 1 is the node corresponding to the output gate. We will sometimes refer to this as the root node and denote it ρ. The additional n nodes are "input nodes" and input node i has an edge pointing to gate node j if the ith input wire feeds into gate j in C. For a given node v, we will refer to any node w to which v has an edge as a *parent* of v, and we will refer to any node x which has an edge to v as a *child* of v. Finally, for a given node v, we will say the *height* of v is the number of edges on the longest path from any leaf node to v.

The basic structure of the algorithm is as follows. First, all the players form quorums and each quorum is assigned to multiple nodes in G, so that each node in G is represented by a unique quorum (Algorithm 2). Then each player commits its input to the quorum at the corresponding input node in G (Algorithm 3). Then all quorums representing gate nodes generate shares of uniformly random field elements. These shares will be needed as inputs to the subsequent heavyweight SMPC protocols.

Next we begin computation of the gates of the circuit. For every node g in G associated with a gate in C, we do the following. At a time proportional to the height of the gate g, all participants in the computation of g (*i.e.* the quorums at g and the quorums at the two nodes pointing to g in the circuit) will run a heavyweight SMPC protocol to compute a masked version of the value at g. (Algorithm 5). Then the quorum at the root node will unmask the output (Algorithm 6) and it will be sent to all the players via a binary tree (Algorithm 7). In order for the players to coordinate their operations, we will need to define the following quantities. Let

$T_{\text{QF}} = T_{\text{QF}}(n)$ denote an upper bound on the time taken for n players to run the quorum formation algorithm.

$T_{\text{VSS}} = T_{\text{VSS}}(\log n)$ denote an upper bound on the input commitment via VSS.

$T_{\text{R}} = T_{\text{R}}(\log n)$ is the maximum time taken by the players in a single quorum to jointly generate shares of a random field element.

$T_{\text{SMPC}} = T_{\text{SMPC}}(\log n)$ denote an upper bound on the time it takes $O(\log n)$ players to perform a heavyweight SMPC.

We remind the reader that in our model local computation is instantaneous, and that a single "time unit" refers to the time taken for a message sent by a processor to reach its intended recipient.

We now present a formal description of our scalable SMPC protocol in Algorithm 1 and related subroutines. For convenience, we will sometimes abuse notation by allowing a node $v \in G$ to refer both to the node itself and to the quorum associated with the node.

Algorithm 1 Main Algorithm

Phase 1

1. At time $t = 0$ all players run the quorum formation algorithm (Algorithm 2).
2. At time $t = T_{\text{QF}}$ all players run the input commitment algorithm (Algorithm 3).
3. At time $t = T_{\text{QF}} + T_{\text{VSS}}$, for each gate simultaneously, players run the random number generation algorithm (Algorithm 4).
4. At time $t = T_{\text{QF}} + T_{\text{VSS}} + T_{\text{R}}$, for each gate g simultaneously, players initiate the computation of gate g (Algorithm 5).

Phase 2

5. At time $t = T_{\text{QF}} + T_{\text{VSS}} + T_{\text{R}} + h_\rho T_{\text{SMPC}}$, the players at the root node reconstruct the output (Algorithm 6). Here h_ρ is the height of the root node.
6. At time $t = T_{\text{QF}} + T_{\text{VSS}} + T_{\text{R}} + (h_\rho + 1)T_{\text{SMPC}}$, all players perform the output propagation algorithm (Algorithm 7)

2.5 Rational Players

We now show how to modify Algorithm 1 to handle rational players (Theorem 1.2); First, we note for the rational case, the graph G is equivalent to that in Algorithm 1. Moreover, the mapping from quorums to nodes in G is equivalent, except for the efficiency of the algorithm that creates the quorums. In particular, in the case where all players are rational, as is stated in Theorem 2.1, we require each player to send only $\tilde{\text{O}}(1)$ bits in order to create the set of n quorums.

Once the quorums have been formed, much of the algorithm, remains the same, including the input commitment and the (masked) computation of each gate. It is only at the output reconstruction stage of the algorithm that things need to change. The problem is that the SMPC protocol being used as a black box does not make any guarantees about all the players learning

Algorithm 2 Quorum Formation

This algorithm begins at time $t = 0$ and all players participate.

1. Run the algorithm in [18] to form n good quorums of size $O(\log n)$, numbered $1, 2, \ldots, n$, with the following properties:
   - All quorums have at least a 2/3 fraction of good players.
   - Each player participates in $O(\log n)$ quorums.
2. Each player identifies the nodes in G represented by his quorums, and the neighboring nodes in the graph G. The rule here is that quorum i represents gate j if $i = j \mod n$.
3. At the end of this protocol, each player knows
   - which $O(\log n)$ quorums to participate;
   - which other players are in each of those quorums;
   - which gates/nodes are represented by those quorums; and
   - which quorums represent the neighboring nodes (with whom it is necessary to communicate) and which players are in each of those quorums.

Algorithm 3 Input Commitment

This protocol for each input node begins at time $t = T_{\text{QF}}$. Recall that x_i is the input associated with player i.

1. Each player i chooses a uniformly random element $r_i \in \mathbb{F}$.
2. Each player i computes $s_i \leftarrow x_i + r_i$
3. Each player i creates VSS shares of r_i for each player in the quorum at input node $m + i$, using the BGW scheme, and sends one share to each member of this quorum. These shares have the property that r_i can be reconstructed from them even if upto a third of them are suppressed or misrepresented.
4. Each player i sends s_i to each member of the quorum at input node $m + i$.
5. Quorums mapped to each input node $m+i$ do the following: Run the VSS verification protocol to determine whether a valid secret has been shared. Also verify, using Byzantine agreement, that the same s_i has been sent to everyone. If either of these checks fails, set x_i to some preset default value, r_i and its shares to zero.

Algorithm 4 Random Number Generation

This protocol is run simultaneously by each quorum associated with each gate node $v \in G$ at time $t = T_{\text{QF}} + T_{\text{VSS}}$.
The following is done by each player $p \in v$:

1. Player $p \in v$ chooses uniformly at random an element $r_{p,v} \in \mathbb{F}$ (this must be done independently each time this algorithm is run and independently of all other randomness used to generate shares of inputs etc.)

2. Player p creates verifiable secret shares of $r_{p,v}$ for each player in g and deals these shares to all players in g (including itself).

3. Player p participates in the verification protocol for each received share. If the verification fails, set the particular share value to zero.

4. Player p adds together all the shares (including the one it dealt to itself). This sum will be player p's share of the value r_v.

Algorithm 5 Computation of a gate

This protocol is run simultaneously for each gate node $g \in G$, starting at time $t = T_{\text{QF}} + T_{\text{VSS}} + T_{\text{R}} + (h_g - 1)T_{\text{SMPC}}$, where h_g is the height of g. Let $v_1, v_2, \ldots v_k$ be the children of the node g in the graph G; and let $O_1, O_2, \ldots, O_k$ be the outputs of the gates associated with these children. The algorithm maintains the invariant that for each child node v_i, there is a uniformly random element $r_i \in \mathbb{F}$ and a value $s_i = O_i + r_i$, such that each player in v_i knows s_i and a unique VSS share of r_i. Also, each player at g has a VSS share of a value r_g that is a uniformly random element of $\mathbb{F}$. Let $f_g(O_1, O_2, \ldots, O_k)$ be the function computed by the gate in the circuit C associated with g.

1. Every player in the quorums $g, v_1, v_2, \ldots, v_k$ run HEAVYWEIGHT-SMPC with the inputs $(s_1,$ shares of $r_1, s_2,$ shares of $r_2, \ldots, s_k,$ shares of $r_k,$ shares of $r_g)$ to compute a value s_g, where $s_g = f_g(O_1, O_2, \ldots, O_k) + r_g$. If a single player p appears in $k' > 1$ of these quorums p plays the role of k' different players in HEAVYWEIGHT-SMPC, one for each quorum to which p belongs.

2. The players in the quorum at g now have s_g and shares of r_g

---

Algorithm 6 Output Reconstruction

This protocol is run by all players in the quorum at the root node ρ, at time $t = T_{\text{QF}} + T_{\text{VSS}} + T_{\text{R}} + h_\rho T_{\text{SMPC}}$.

1. Reconstruct r_ρ from its shares using VSS.

2. Set the output $\mathbf{o} \leftarrow s_\rho - r_\rho$.

3. Send $\mathbf{o}$ to all players in the quorums numbered 2 and 3

Algorithm 7 Output Propagation

Performed by the players at each node by the players at each quorum, q other than the quorum numbered 1, starting at time $t = T_{\text{QF}} + T_{\text{VSS}} + T_{\text{R}} + (h_\rho + 1)T_{\text{SMPC}}$ wait)

1. $i \leftarrow$ quorum number of q
2. Each player $p \in q$ waits until it receives values from at least a 2/3 fraction of the players in the quorum numbered$\lfloor i/2 \rfloor$, and sets $\mathbf{o} \leftarrow$ the unique value that occurs as at least 2/3 of the received values.
3. Each player $p \in q$ sends $\mathbf{o}$ to all the players in quorums numbered $2i$ and $2i + 1$.

the output at the same time. This did not matter for the computations at internal gates since the intermediate output there was masked and therefore uniformly random, and gave the players no information about either the output or anybody's input. However at the end of the output reconstruction stage, players at the root actually learn the output. Thus if any single player learns it first, then he may simply stop sending messages and the other players will not learn the output. To overcome this difficulty, in the output reconstruction phase, instead of using the usual heavyweight SMPC protocol, we use a rational SMPC protocol due to Abraham, Dolev, Gonen and Halpern [1, Theorem 2(a)]. This ensures that all players at the root node learn the output simultaneously.

Finally the players at the root use Algorithm 7 in order to send the output to all n players. We note that to run Algorithm 7 at this point is a Nash equilibrium since if all other players are running Algorithm 7, there is no expected gain in utility for a single player by deviating from Algorithm 7.

3 Analysis

In this section, we give the proof of Theorem 1.1.

We begin by noting that the error probability in Theorem 1.1 comes entirely from the possibility that the quorum formation algorithm of Lonergan *et al.* [18] may fail to result in good quorums (see Theorem 2.1). All other components of our algorithm: the VSS and heavyweight black boxes, Byzantine agreement and majority filtering, are all exact algorithms with no error probability. For the remainder of this section we will assume that we are in the good event, *i.e.* that the players have successfully formed n good quorums.

For each node j in the graph G, let V_j be the value of the node in the computation of f. Thus, for input nodes, V_j is the input which has been committed to by the corresponding player (set to a default value if the player faulted on the input commitment algorithm), while for gate nodes, V_j is the value on the output wires of the gate associated with j in the circuit, once the inputs have been fixed to the committed values. Also for each node j we have a *mask* $r_j \in \mathbb{F}$. For input nodes, r_j is the random number set by the player in the input commitment algorithm (set to zero if the player faulted). For gate nodes, r_j is the random number jointly generated by the quorum at j. Let G' be the set of all nodes in G which are either input nodes corresponding to good players or gate nodes.

Lemma 1. The masks $\{r_j\}_{j \in G'}$ are fully independent and uniformly random in $\mathbb{F}$.

Proof. The masks corresponding to input nodes for good players are uniformly random by choice (see Algorithm 3). To see that the masks for the gates are uniformly random, recall that if j is

a gate node $r_j = \sum_i r_{j,i}$ where $r_{j,i}$ is the value selected by player i in Algorithm 4. The players commit to the $r_{j,i}$ values by sending each other VSS shares of them and then running the verification protocol on the shares. If player i is good, $r_{j,i}$ is uniformly random. If player i is bad then $r_{j,i}$ could be anything (including zero, if player i's shares failed the subsequent verification). However, once the players have committed to the values the bad players can no longer influence the sum of the $r_{j,i}$, nor can they bias the distributions of the $r_{j,i}$ in any way, because of the security provided by the VSS algorithm. Since the sum of elements of $\mathbb{F}$ is uniformly random if at least one of them is uniformly random, it follows that r_j is uniformly random. The independence of the $\{r_j\}$, $j \in G'$ follows from the fact that all players have sampled their values independently. □

In the following, the "computation of a node j" will refer either 1) the input commitment algorithm if j is an input nodes; or 2) Algorithm 5 if j is a gate node.

Lemma 2. For each node j in G, after the computation of j each player in the corresponding quorum knows a share of a number r_j. Moreover all good players in the quorum at j agree on a value $s_j \in \mathbb{F}$ such that $s_j - r_j = V_j$.

Proof. The players already have the shares of r_j at the end of the random number generation stage. We prove the claims about s_j by induction. For the base case, note that for each input node, since the corresponding quorum has at least two thirds good players, the conclusion follows from the correctness of the VSS protocol, and the Byzantine agreement protocol used in the input commitment algorithm.

Now let j be a gate node and suppose for all nodes j' whose height is less than the height of j, that all the good players at j agree on $s_{j'}$ and $s_{j'} - r_{j'} = V_{j'}$. Then the inductive hypothesis holds for all nodes v_1, v_2, v_k whose outputs are connected to the inputs of j. Thus, we can assume that for all i between 1 and k, the players at node v_i have shares of some value r_i chosen uniformly at random in $\mathbb{F}$, and that all players in node v_i know the value $s_i = V_i + r_i$. In the computation at node j, the $k+1$ quorums involved run HEAVYWEIGHT-SMPC with inputs $s_1, s_2, \ldots s_k$ and the shares of $r_j, r_1, r_2, \ldots, r_k$. At the end of this protocol, all good players agree on a common value s_g. (This is by the correctness of HEAVYWEIGHT-SMPC).

To see that this common value is actually $V_g + r_g$ we note that the function computed by HEAVYWEIGHT-SMPC consists of reconstructing $r_g, r_1, r_2 \ldots, r_k$ from their shares; inferring the values $V_1, V_2, \ldots, V_k$; computing V_g from them; and adding r_g back in. (All of this will, of course, be opaque to the players involved.) Attempts to corrupt this computation by lying about $s_1, s_2, \ldots, s_k$ are easily thwarted, because of the high redundancy in these as inputs. For each of these values, at least twice as many players provide them correctly as try to lie (since each of the input quorums have at most a third bad players). Moreover, note that the VSS used to reconstruct the masks from the shares can tolerate up to a third of the shares being corrupted. Thus, since all quorums are good, even if the bad players lie about their shares of the masks, they cannot change the value of the computation. It follows that $s_g = V_g + r_g$. By induction, all the nodes in G compute the correct masked values □

Corollary 1. After the Output Reconstruction (Algorithm 6), all players at the root node know the output.

Proof. By Lemma 2, at the end of Phase 1 of the main algorithm, all the players at the root node know the value s_ρ and shares of r_ρ, where $s_\rho - r_\rho$ is the output of the circuit. During Algorithm 6, these players run the VSS secret reconstruction protocol. Since at least two thirds of them are

good, by properties of VSS, they correctly reconstruct r_ρ. Since all players at the root node know the value of s_ρ, subtracting from it the reconstructed r_ρ, they all learn the correct output. □

Lemma 3. At the end of the algorithm, the correct output is learned by all good players.

Proof. This follows by induction. Since quorum 1 is at the root, Corollary 1 provides a base case. Now suppose the correct output has been learned by all the players in quorums numbered j for all $j < i$. Consider the players in quorum i. During the run of the output propagation algorithm, they will receive putative values for the output from the players at quorum $\lfloor i/2 \rfloor$. Since at least two thirds of the players at quorum $\lfloor i/2 \rfloor$ are good, and by induction hypothesis have learned the correct output, it follows that at least two thirds of the values received by the players at quorum i equal correct output. Since good players set their output to be the the unique value that occurs as at least 2/3 of the received values, they get the correct output. By induction, all the players learn the correct value. □

We devote the rest of the section to showing that privacy of the inputs is preserved. We remark that privacy is only guraranteed with high probability. However, as in the case of correctness, the error arises only from the possibility that the quorum formation algorithm fails to spread the bad players out so that less than a third of the players in any quorum are bad. Thus if we condition on having formed n good quorums, then all the privacy claims hold with probability 1. For what follows we will continue to condition on this good event.

As discussed in previous works (see [21]), we have no recourse against players who voluntarily send their inputs to other players, naturally we cannot preserve the privacy of such players. In particular, we are only concerned with preserving the privacy of good players, who perform no actions except those specified by the protocol.

We are primarily concerned with preserving the privacy of *inputs of players*. However, note that if some player's input feeds into a multiplication gate then learning that the value in the computation of that gate is zero, increases the Bayesian probability that the player's input is zero, and this is a privacy violation. Thus we are also concerned about the ability of players to learn the value of a gate other than the root or output gate.

Recall that G' is the set of nodes in G that are either input nodes corresponding to good players or gate nodes.

Lemma 4. Let j be any node in G', other than the root node, ρ. Using only messages sent to him as part of the algorithm, no player can learn any information about the value V_j, except what is implicit in his own input and the final output of the circuit.

Proof. We prove this for a gate node g. By Lemmas 1 and 2, the value recovered by HEAVYWEIGHT-SMPC during the computation of g is $s_g = V_g + r_g$, where r_g is a uniformly random element of $\mathbb{F}$, independent of all other randomness in the algorithm. In particular this means that s_g holds no information about V_g. If the player i is not in any of the quorums at g or its neighbors, then all the messages he receives during the algorithm are independent of r_g, and hence s_g, and hence he cannot learn anything about V_g. On the other hand, if player i is involved in the computation of g or one of its neighbors, then he may hold a share r_g as well as shares of other shares. In this case we appeal to the privacy of HEAVYWEIGHT-SMPC and the embedded VSS algorithm to see that although he may learn s_g, he cannot learn any information about the shares of r_g and hence about r_g itself. Thus, he cannot learn any information about V_g except what is implicit in his input and the circuit output.

The proof for an input node of a good player is similar except that we will have to appeal to the privacy of the black box VSS protocol rather than the privacy of HEAVYWEIGHT-SMPC. □

We now explore a stronger notion of privacy. BGW [4] distinguish between the two kinds of deviant behaviour among players. The bad players are players controlled by an adversary who may indulge in arbitrary kinds of erratic behaviour to try to break the protocol in any way they can. However BGW also consider players who are good, in the sense that they follow the protocol, but may also send and receive messages external to the protocol, to attempt to learn whatever additional information they can. Such players are called *gossiping* players. A protocol is called t-private if no coalition of size t (including coalitions of gossiping players) can learn anything more than what is implied by their private inputs and the circuit output. The SMPC protocol of BGW [4] is $(n/3 - \delta)$-private for any $\delta > 0$.

We note that our algorithm *is* susceptible to adaptively chosen coalitions of gossiping players. Indeed, if all the players in a quorum at a node j gossip with each other, they can reconstruct the corresponding random mask r_j and hence the value V_j. In particular, the players in the quorum at an input node can jointly reconstruct the corresponding input.

However, we can establish the following result, which shows that for large coalitions chosen non-adaptively (in particular, the adversarial players) our algorithm will preserve privacy.

Lemma 5. Let S be any set of players such that for every quorum Q, $S \cap Q$ contains fewer than a third of the players in Q. Let j be any node in G'. Then the coalition S cannot learn any information about V_j that cannot be computed from their (collective) private inputs and the circuit output.

Proof. Once again we prove this only for gate node g in G'. The proof for an input node is similar. We know that HEAVYWEIGHT-SMPC when run at g computes $s_g = V_g + r_g$, where r_g is uniform in $\mathbb{F}$ and independent of all other randomness in the algorithm. As noted in the proof of Lemma 4, the players in S who are not in the quorums at g or any of its neighbors are irrelevant to the coalition: all of the information that they hold is completely independent of r_g and s_g, so they cannot assist in uncovering any information about V_g, except what is implicit in their private inputs.

Now consider the players in the quorums at g or any of its neighbors. These players participate in one or more of the SMPCs which involve g: the computation of g itself or the computations in which the output of g is an input. Here we appeal to the privacy of HEAVYWEIGHT-SMPC to see that the players cannot learn any additional information that is not implied by their inputs. The players in S are unable to directly determine r_g, since the only relevant inputs are the shares of r_g, and they do not have enough of those.

Finally, let us consider the players from S at g itself. These players also do not have enough shares of r_g to reconstruct it on their own. However, they recieve shares of each of the other shares of r_g multiple times: once during the input commitment phase of each SMPC in which g is involved. Each time, they do not get enough shares of shares r_g to reconstruct any shares of r_g. However, can they combine the shares of shares from different runs of the VSS protocol for the same secret to gain some information? Since fresh, independent randomness was used by the dealers creating these shares on each run, the shares from each run are independent of the other runs, and so they do not collectively give any more information than each of the runs give separately. Since each run of the VSS input commitment does not give the players in S enough shares to reconstruct anything, it follows that they do not learn any information about r_g. Since r_g is uniformly random, so is s_g and it follows that the coalition S cannot get any extra information about V_g. □

Corollary 2. The bad players cannot learn any information, except what is implied by the output and the inputs to which they committed, about the input of any good player.

Proof. This follows immediately from Lemma 5, since each quorum consists of no more than a third bad players. □

Let q be the size of the smallest quorum. Recall that $q = \Theta(\log n)$.

Corollary 3. Our algorithm is $q/3$-private.

Proof. Since q is the size of the smallest quorum, any set of size $q/3$ intersects a quorum Q in at most a third of its members. The result follows from Lemma 5 □

We end with a simple analysis of the resource cost of our algorithm.

Lemma 6. If all good players follow Algorithm 1, with high probability, each players sends at most $\tilde{O}(\frac{n+m}{n} + \sqrt{n})$ messages. m is size of G

Proof. To analyze the cost of algorithm 1, we have to first analyze the cost of its sub-algorithms.

Cost of Algorithm 2 and Algorithm 3: Based on the theorem 2.1 we need to send $\tilde{O}(\sqrt{n})$ messages to build the quorums. In Algorithm 3, each player must commit its secret and a random variable using verified secret sharing between $O(\log n)$ players of a quorum (input node). This requires sending a polylogarithmic number of messages.

Cost of Algorithm 5: Each player will participate in $\theta(\log n)$ quorums. For each quorum, he has to participate in a secure multi-party computation for $\theta(\frac{m+n}{n})$ (m is number of operations in circuit G) gates between three quorums or $3 \log n$ players which is polylogarithmic, so this algorithm requires sending $\tilde{O}(\log n \frac{n+c}{n})$ messages.

Cost of Algorithm 7: output tree, Each player should send $\tilde{O}(1)$ messages (output message) to the players of its children.

So the cost of the algorithm 1 is $\tilde{O}(\frac{n+m}{n} + \sqrt{n})$.

□

4 Conclusion

We have described scalable algorithms to perform Secure Multiparty Computation in a scalable manner. Our algorithms are scalable in the sense that they require each player to send $\tilde{O}(\frac{n+m}{n}+\sqrt{n})$ messages and perform $\tilde{O}(\frac{n+m}{n} + \sqrt{n})$ computations. They tolerate an adversary that controls up to a $1/3 - \epsilon$ fraction of the players, for ϵ any positive constant. We have also described a variant of this algorithm that tolerates the case where all players are rational; this variant requires each player to send $\tilde{O}(\frac{n+m}{n})$ messages and perform $\tilde{O}(\frac{n+m}{n})$ computations.

Many open problems remain including the following. First, Can we design scalable algorithms to solve SMPC in the completely asynchronous communication model? We believe this is possible with some work. Second, Can we prove lower bounds for the communication and computation costs for Monte Carlo SMPC? Finally, Can we implement and adapt these algorithms to make them practical for a SMPC problem such as the one described in [5].

References

[1] I. Abraham, D. Dolev, R. Gonen, and J. Halpern. Distributed computing meets game theory: robust mechanisms for rational secret sharing and multiparty computation. In *Proceedings of the twenty-fifth annual ACM symposium on Principles of distributed computing*, pages 53–62. ACM, 2006.

[2] B. Applebaum, Y. Ishai, and E. Kushilevitz. From secrecy to soundness: efficient verification via secure computation. *Automata, Languages and Programming*, pages 152–163, 2010.

[3] Z. Beerliova and M. Hirt. Efficient multi-party computation with dispute control. In *Theory of Cryptography Conference*, 2006.

[4] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computing. In *Proceedings of the Twentieth ACM Symposium on the Theory of Computing (STOC)*, pages 1–10, 1988.

[5] P. Bogetoft, D. Christensen, I. Damgård, M. Geisler, T. Jakobsen, M. Krøigaard, J. Nielsen, J. Nielsen, K. Nielsen, J. Pagter, et al. Secure multiparty computation goes live. *Financial Cryptography and Data Security*, pages 325–343, 2009.

[6] Gabriel Bracha. An asynchronous [(n - 1)/3]-resilient consensus protocol. In *PODC '84: Proceedings of the third annual ACM symposium on Principles of distributed computing*, pages 154–162, New York, NY, USA, 1984. ACM.

[7] I. Damgård and Y. Ishai. Scalable secure multiparty computation. *Advances in Cryptology-CRYPTO 2006*, pages 501–520, 2006.

[8] I. Damgård, Y. Ishai, M. Krøigaard, J. Nielsen, and A. Smith. Scalable multiparty computation with nearly optimal work and resilience. *Advances in Cryptology–CRYPTO 2008*, pages 241–261, 2008.

[9] I. Damgård and J.B. Nielsen. Scalable and unconditionally secure multiparty computation. In *Proceedings of the 27th annual international cryptology conference on Advances in cryptology*, pages 572–590. Springer-Verlag, 2007.

[10] W. Du and M.J. Atallah. Secure multi-party computation problems and their applications: a review and open problems. In *Proceedings of the 2001 workshop on New security paradigms*, pages 13–22. ACM, 2001.

[11] K.B. Frikken. Secure multiparty computation. In *Algorithms and theory of computation handbook*, pages 14–14. Chapman & Hall/CRC, 2010.

[12] O. Goldreich. Secure multi-party computation. *Manuscript. Preliminary version*, 1998.

[13] S. Gordon and J. Katz. Rational secret sharing, revisited. *Security and Cryptography for Networks*, pages 229–241, 2006.

[14] J. Halpern and V. Teague. Rational secret sharing and multiparty computation: extended abstract. In *Proceedings of the thirty-sixth annual ACM symposium on Theory of computing*, page 632. ACM, 2004.

[15] W. Henecka, A.R. Sadeghi, T. Schneider, I. Wehrenberg, et al. Tasty: Tool for automating secure two-party computations. In *Proceedings of the 17th ACM conference on Computer and communications security*, pages 451–462. ACM, 2010.

[16] M. Hirt and U. Maurer. Robustness for free in unconditional multi-party computation. In *Advances in CryptologyCRYPTO 2001*, pages 101–118. Springer, 2001.

[17] M. Hirt and J. Nielsen. Upper bounds on the communication complexity of optimally resilient cryptographic multiparty computation. *Advances in Cryptology-ASIACRYPT 2005*, pages 79–99, 2005.

[18] V. King, S. Lonergan, J. Saia, and A. Trehan. Load balanced scalable byzantine agreement through quorum building, with full information. In *International Conference on Distributed Computing and Networking (ICDCN)*, 2011.

[19] G. Kol and M. Naor. Games for exchanging information. In *Proceedings of the 40th annual ACM symposium on Theory of computing*, pages 423–432. ACM, 2008.

[20] A. Lysyanskaya and N. Triandopoulos. Rationality and adversarial behavior in multi-party computation. *Advances in Cryptology-CRYPTO 2006*, pages 180–197, 2006.

[21] T. Rabin and M. Ben-Or. Verifiable secret sharing and multiparty protocols with honest majority. In *Proceedings of the twenty-first annual ACM symposium on Theory of computing*, pages 73–85. ACM, 1989.

[22] A.C. Yao. Protocols for secure computations. In *Proceedings of the 23rd Annual Symposium on Foundations of Computer Science*, pages 160–164, 1982.

Life Won't Wait!
(On the Slowdown of Asynchronous Automata Networks)

Thomas P. Hayes* Michael Janes* Christopher Moore *

Abstract

Nakamura [1974], and later independently, Toffoli [1978] and Nehaniv [2002] proved that any synchronous cellular automaton can be simulated by an asynchronous cellular automaton. Their constructions effectively discard many of the updates in order to create a limited artificial synchrony between the cells. We consider the overhead cost of this procedure, assuming that cells update in random order. In particular, we prove explicit upper bounds on this overhead that depend only on the maximum degree of the network.

1 Introduction

When *you* think of cellular automata, what example comes to mind first? For many, the answer would be John Conway's Game of Life, in which the cells of a square lattice change states from alive to dead and back according to simple rules, which nevertheless lead to surprisingly complex behaviors over time. This CA is *synchronous* in the sense that, depending on our exact implementation, the cells must either all update simultaneously, or at least in a more or less deterministic order; therefore, some sort of global clock is required to ensure that none of the cells race ahead or lag behind, in terms of how many times they have updated.

Since one of the nice features of cellular automata is the extent to which all computation is done locally, we would prefer to avoid the need for globally coordinated update

*University of New Mexico, Department of Computer Science. email: hayes,mjanes,moore@cs.unm.edu

timing. Indeed, we would like our computation to be almost totally independent of the order in which cells are updated.

Fortunately, this is always possible! As first shown in 1974 by Nakamura [1], and later independently rediscovered by Toffoli [3] and Nehaniv [2], there is a simple construction that allows any synchronous CA to be simulated by a completely asynchronous CA, as long as each cell eventually updates infinitely often. The elegant idea is to augment each cell's state space slightly, to maintain a mod-3 counter of the number of times it has successfully updated, and to ensure that each cell only changes state when its neighbors have all caught up to it or are one step ahead.

This construction ensures that each cell will go through the same sequence of updates as the original synchronous CA, although each cell does this on its own schedule. For two cells that are far apart in the network, the coupling between their update schedules may be very loose indeed.

In this paper, we examine the overhead incurred by the above construction, in terms of the fraction of attempted updates which must be wasted because the counters of neighboring cells have fallen behind. For simplicity, we will assume that, at any given time, the next cell to update is chosen uniformly at random. More specifically, we will think of each cell as updating at each ring of its own independent Poisson clock, that rings with an average rate of 1.

Our main result is the following, which partially answers open problems (1) and (2) from [2]. For a fixed graph G, define the *asymptotic slowdown*, S, as the asymptotic ratio of the elapsed time to the number of successful updates of a given cell.

Theorem 1. *Let G be a graph with maximum degree Δ. Then, assuming i.i.d. Poisson updates, the asymptotic slowdown, S, for asynchronous simulation of any synchronous CA network on G, satisfies*

$$S \exp(1 - S) \geq \frac{1}{1 + \Delta},$$

or, equivalently,

$$S \leq W_{-1}\left(\frac{-1}{\mathrm{e}(1 + \Delta)}\right),$$

where W_{-1} denotes the lower branch of the Lambert W function.

Note that in the worst case when G is the complete graph on n vertices, the slowdown equals the expected time for the standard Coupon Collector problem, i.e., $\ln(n)$. The bound from Theorem 1 is approximately $\ln(n) + \ln(\ln(n))$ in this case.

Lattice	Δ	$S \leq$
1-D (or path or cycle)	2	3.289
2-D square lattice	4	3.994
3-D cubic lattice	6	4.436
2-D square lattice with diagonals (Game of Life)	8	4.757

Figure 1: Bounds to the slowdown of some lattices of interest

2 Preliminaries

We will assume throughout the paper, that the automata network is given by the graph G on n vertices with maximum degree Δ. We will be concerned with the following random process, which models the number of successful updates at each vertex.

Each vertex v has its own i.i.d. Poisson clock, with parameter 1. At time t, we denote by $c(v,t)$ the count of successful updates at v. Initially, $c(v,0) = 0$. Subsequently, each time v's Poisson clock rings at time t, we set $c(v,t) = c(v,t^-) + 1$ if, for all $w \in N(v)$, $c(v,t^-) \leq c(w,t)$, and otherwise, $c(v,t) = c(v,t^-)$. Here, the shorthand t^- refers to any time before t but after the preceding ring of v's Poisson clock.

By the *asymptotic efficiency*, we mean the limit

$$\frac{1}{S} := \lim_{t\to\infty} \frac{c(v,t)}{t},$$

of the number of successful updates to the expected number of total updates. We refer to the inverse efficiency, S, as the *asymptotic slowdown*.

Our assumption of continuous time is really just a convenience. By standard techniques, all our results immediately translate into analogous results in discrete time, assuming each vertex is sampled independently uniformly at random.

3 A Comparison Argument for the 1-D Case

Note that, although the successful update counts $c_t(v) = c(v,t)$ tend to infinity, if we mod out by additive constants, there are only finitely many possibilities, for a fixed graph, G. Indeed, in this case, the function $c_t - \min c_t$ is a finite Markov chain with respect to the continuous time parameter t. It follows from general principles that there is a unique stationary distribution that this chain approaches in the limit as $t \to \infty$.

The asymptotic slowdown is therefore a function of this stationary distribution. Indeed, we make the following observation.

Observation 2. *If, for a vertex v, the stationary probability is p that, for all $w \in N(v)$, $c(v) \leq c(w)$, then $S = 1/p$.*

Note that this probability cannot depend on the vertex v, since no two vertices can ever have their counts differ by more than n (we assume G is connected).

Unfortunately, the stationary distribution for c_t (mod translation) is not so easy to describe. In this section, we consider the special case when G is a $2n$-cycle, for which a slight variant of the Markov chain c_t converges to a uniform distribution. Thereby, through an easy comparison argument, we can prove a clean upper bound on the slowdown for this graph.

Definition 3. Define the variant count functions $\widetilde{c}(v,t)$ by the same update rule as $c(v,t)$, but with the initial condition

$$\widetilde{c}(v,0) = \begin{cases} 0 & \text{if } v \text{ is even} \\ 1/2 & \text{if } v \text{ is odd} \end{cases}$$

Now, $\widetilde{c}$ differs from c in that, for adjacent vertices v, w, we can never have a tie, $\widetilde{c}_t(w) = \widetilde{c}_t(v)$. Surprisingly, this leads to the following consequence. For convenience, we will assume that $\widetilde{c}$ is only defined up to translation by additive constants.

Lemma 4. *The stationary distribution for $2\widetilde{c}$ is uniform over all functions $2\widetilde{c} : [2n] \to \mathbb{Z}$ that satisfy $|\widetilde{c}(i+1) - \widetilde{c}(i)| = 1$, for all i, where $i+1$ is computed mod $2n$.*

Proof. It is clear by induction that every $\widetilde{c}$ is of the form described. A greedy construction shows that every function of the form described is achievable starting from $\widetilde{c}_0$. All that remains is to verify that the uniform distribution is stationary.

To see this, fix a function $\widetilde{c}_t$. Observe that the following are all equal:

- the number of distinct successor states from $\widetilde{c}_t$
- the number of distinct predecessor states to $\widetilde{c}_t$
- 1 plus the number of local maxima of $\widetilde{c}_t$
- 1 plus the number of local minima of $\widetilde{c}_t$.

This is clear, because each successful update converts a (preceding) local minimum into a (subsequent) local maximum. And the number of maxima equals the number of minima because ties are not allowed, so minima and maxima must alternate. Thus, under the uniform distribution, the detailed balance condition is satisfied. □

Lemma 5. *The $\widetilde{c}_t$ process has asymptotic slowdown ≤ 4.*

Proof. Consider a sequence of $2n$ coin flips used to generate $\widetilde{c}_t$ (heads correspond to a change of $+1/2$ from the previous value, tails to a change of $-1/2$). Since a local minimum corresponds to a tails followed by a heads, we expect a $1/4$ fraction of these. This is in fact the asymptotically correct value for the slowdown as $n \to \infty$, by an application of the central limit theorem. However, for small n, the expected number of local minima is larger, because the cycle topology creates some negative dependence between successive differences. □

Lemma 6. *The c_t process on the cycle has asymptotic slowdown ≤ 4.*

Proof. For any fixed order of vertex updates, the process $\widetilde{c}_t - \widetilde{c}_0$ will always be $\leq c_t$. Whenever they are equal, the $\widetilde{c}_t$ process declines to update the odd cells unless they are strict local minima, whereas the c_t process will allow updates to non-strict local minima. Since successful updates always increase the counts by 1, there is no way for $\widetilde{c}$ to "come from behind" to pass up c. The result follows from Lemma 5 and the definition of slowdown. □

Before going on, we observe that the uniformity of the stationary distribution of $\widetilde{c}$ allows us to prove all sorts of other nice properties of it, such as that, with high probability, $\max_v \widetilde{c}(v,t) - \min_w \widetilde{c}(w,t) = O(\sqrt{n})$. Unfortunately, it seems that the actual distribution of $c(v,t)$ is quite far from uniform. Furthermore, we do not know how to generalize the techniques of this section to higher dimensions.

4 Proof of Theorem 1

To prove a bound on the slowdown for general graphs, we will consider the ways a missing update at some other vertex could interfere with the counter at vertex v reaching a certain value. Fortunately, the locality of updates in a CA network means that the count at each cell can only be influenced by events taking place within a certain "light cone" in space-time (Figure 2). In our context, space is measured through the shortest-path distance in G, while local time is measured in rings of the Poisson clock at a node.

Proof of Theorem 1. Construct a "space-time" graph $G' = (V', E')$, where $V' = V \times \{0, 1, 2, 3, \dots\}$, and E' consists of all directed edges $((v,t),(w,t+1))$, where either $v = w$ or $\{v, w\} \in E$. See Figure 3.

In this graph, the node (v,t) corresponds to the event that the count $c(v,t')$ is at least t. The edges into (v,t) correspond to the requirements that v and all of its neighbors must previously have reached at least $t-1$.

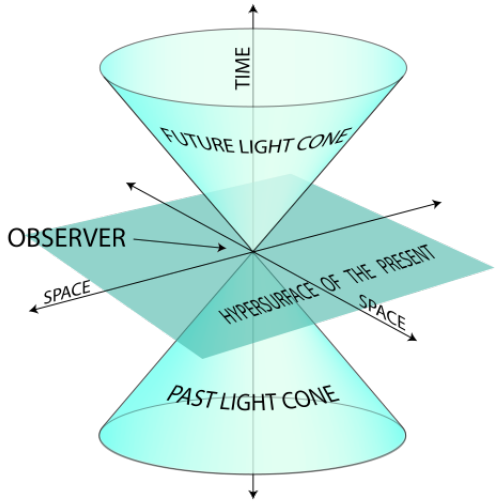
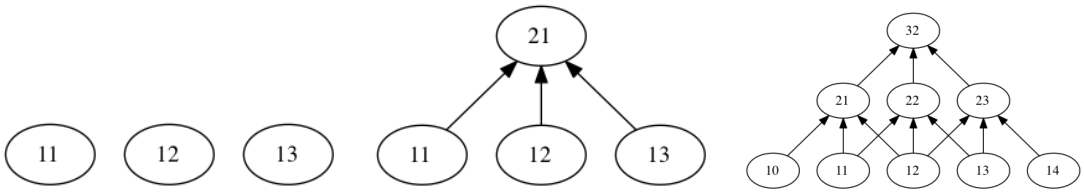


Figure 2: A Light Cone.
[Source: Wikimedia Commons Image:World_line.png]



(a) First layer of G' (b) First two layers of G' (c) First three layers of G'

Figure 3: A "past" light cone when G is a path or cycle.

Say that node (v, t) in G' is *occupied* at time t' if $c(v, t') \geq t$. Then, starting from $c_0 = 0$, each time the Poisson clock at a node u rings, we find the maximum t such that all in-neighbors of (u, t) are occupied, and make (u, t) occupied, if it was not already.

The time it takes for a node $v \in V$ to achieve ℓ updates equals to the time it takes for node (v, ℓ) to be occupied. By an easy induction, this in turn equals the sum of times between consecutive Poisson clock rings along the slowest path from the base layer to v_ℓ.

Now, for any fixed path of length ℓ, the probability that t time units do not suffice for the ℓ Poisson clocks to ring in the correct order, equals

$$\mathrm{e}^{-t} \sum_{j<\ell} \frac{t^j}{j!}$$

Thus, by a union bound over the $(1+\Delta)^\ell$ paths into (v,ℓ), the probability that (v,ℓ) is unoccupied at time t is at most

$$(1+\Delta)^\ell \mathrm{e}^{-t} \sum_{j \le \ell} \frac{t^j}{j!}$$

Setting $t = S\ell$ and approximating the sum by its dominant term, corresponding to $j = \ell$, we obtain

$$\frac{(1+\Delta)^\ell}{\mathrm{e}^{S\ell}} \frac{(S\ell)^\ell}{\ell!} \le \left(\frac{\mathrm{e}S(1+\Delta)}{\mathrm{e}^S} \right)^\ell$$

which tends asymptotically to zero exactly below the threshold from the Theorem statement. Since the convergence is exponential in ℓ, the approximation of the ℓ-term sum by its maximum cannot affect the answer. Therefore, the proof is complete. □

It is worth noting that despite our inability to describe the stationary distribution, and the huge union bound in its proof, Theorem 1 still gives a noticeably better result in the 1-D case than the comparison argument from Section 3 (namely 3.289 instead of 4).

5 Tugs-of-War on Distributions

Brain-teasers of the style "What is the value of $\sqrt{1+\sqrt{1+\sqrt{1+\ldots}}}$" are commonplace. In this case, the answer is well-defined, namely the golden ratio, $1.618...$. We can view this as a "tug-of-war" between alternating operations: "add one" and "square root." The first tends to drive us upward towards its fixed point at $+\infty$, while the second tends to drive us down towards its fixed point at 1. In general, the question of which of two operations will "win" in the long run can be quite interesting.

Consider the following pair of operations A_2, M_2, on distributions. For a distribution X, to sample from A_2X, take two independent samples from X, and average them. To sample from M_2X, take two independent samples from X, and return the larger one. More generally, we can consider operations A_k, M_k that sample k times independently and then average or max respectively. The M operation increases its argument. The A operation concentrates its argument, while keeping its mean fixed. Both operations have the same infinity of fixed points, namely all distributions supported on a single real value. What can we say in general about which operation wins in a tug-of-war?

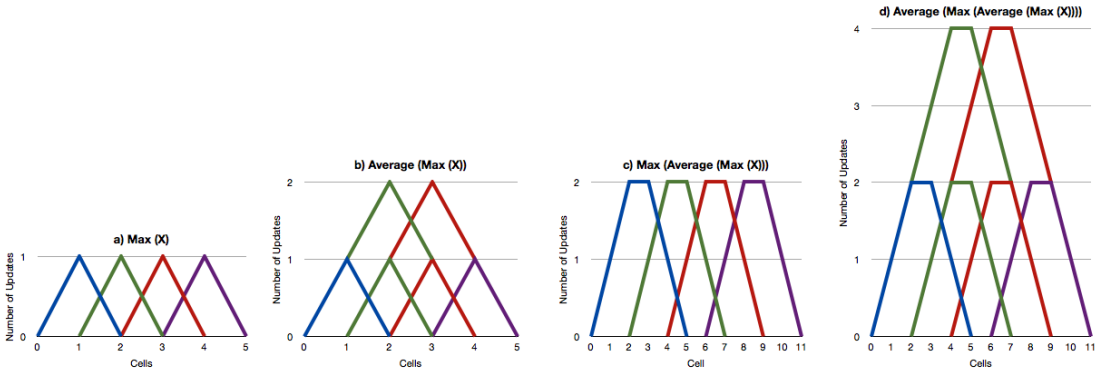


Figure 4: A visual representation of repeatedly maxing and averaging on a space-time graph. Here the underlying topology is one-dimensional, and the alternating operations are M_4 and A_2.

This question is motivated by an alternative approach to Theorem 1 that proved to be less general. Basically, the idea is that in our time-space graph G', the delay to occupy vertex $(v, 2)$ along a particular path equals the average of two independent Poisson clocks. The overall delay to occupy vertex $(v, 2)$ is the maximum of these delays, taken over the 3 different children of $(v, 2)$. More generally, the recursive structure of the light cone suggests a long alternating sequence of averaging and max operations. However, there are two caveats: firstly, the max operations are not taken over independent samples (this turns out to not really matter in our analysis), and secondly, in order to get the operations to take the same number of arguments at each level, we take advantage of a high degree of self-similarity for lattices, that is not present in more general graphs. See Figure 4 Although the application to slowdowns yields inferior results to Theorem 1, we feel that this approach may be of some interest on its own.

If p is the probability that $X \leq A$, where X is a random variable and A is constant, then clearly the probability that the sum of ℓ independent copies of X is at most ℓA is at least p^ℓ, since this is the likelihood that each X_i are at most A. The next result shows that, for exponential moment bounds, something akin to a converse holds.

Lemma 7. *If p is the best possible exponential moment bound for $\mathbf{Pr}(X \leq A)$ then p^ℓ is the best possible exponential moment bound for $\mathbf{Pr}\left(\sum^\ell X \leq \ell A\right)$.*

Proof. By Markov's inequality we have the following exponential moment bound, for all

$t > 0$,

$$\mathbf{Pr}\left(X \geq A\right) = \mathbf{Pr}\left(e^{tX} \geq e^{tA}\right) \leq \frac{\mathbf{E}\left(e^{tX}\right)}{e^{tA}}$$

A second application of Markov's inequality gives the following exponential bound

$$\mathbf{Pr}\left(\sum^{\ell} X \geq \ell A\right) = \mathbf{Pr}\left(e^{t'\sum^{\ell} X} \geq e^{t'\ell A}\right)$$
$$\leq \frac{\mathbf{E}\left(e^{t'\sum^{\ell} X}\right)}{e^{t'\ell A}} = \frac{\mathbf{E}\left(\Pi^{\ell} e^{t'X}\right)}{e^{t'\ell A}} = \left(\frac{\mathbf{E}\left(e^{t'X}\right)}{e^{t'A}}\right)^{\ell}$$

Setting $t' = t$, where t was chosen optimally, we conclude

$$\mathbf{Pr}\left(\sum^{\ell} X \geq \ell A\right) \leq p^{\ell}$$ □

Now we are ready to prove that, as long as our starting distribution doesn't decay too slowly, each averaging operator beats every max operator.

Lemma 8. *Suppose X has a finite exponential moment, and let $k, \ell \geq 2$ be fixed integers. Let A_k denote the "average k i.i.d. copies" operator, and M_ℓ denote any operator that takes the max of ℓ copies, arbitrarily correlated. Then the alternating sequence $(A_k M_\ell)^i X$ converges to a finite real number. Indeed, let $p = (1/\ell)^{1-1/k}$. Then if a is such that p is an exponential moment upper bound on $\mathbf{Pr}\left(X \geq a\right)$, then $(A_k M_\ell)^i X$ converges to a real number between $\mathbf{E}\left(X\right)$ and a.*

Proof. It is easy to verify that, when p is an exponential moment bound on $\mathbf{Pr}\left(X \geq a\right)$, that ℓp is an exponential moment bound on $\mathbf{Pr}\left(M_\ell X \geq a\right)$. By Lemma 7, we know that $(\ell p)^k$ is an exponential moment bound on $\mathbf{Pr}\left(A_k M_\ell X \geq a\right)$. But, by choice of p, $(\ell p)^k \leq p$. By an induction argument, we can deduce that the probabilities $\mathbf{Pr}\left((A_k M_\ell)^i X \geq a\right)$ converge exponentially to zero, and hence the distributions $(A_k M_\ell)^i X$ converge in measure to a distribution supported on reals $\leq a$. Since the operator M_ℓ increases the expectation of any non-constant distribution, it follows that $(A_k M_\ell)^i$ converges in probability to a real constant between $\mathbf{E}\left(X\right)$ and a. □

6 Conclusion

We have seen that, under a random vertex order, the overhead for implementing the asynchronous simulation of synchronous CA's is not only bounded as a function of the vertex degrees, but is a rather small constant for most networks one might seriously consider building. This can be viewed as further evidence of the practicality of such simulations, and of the advantage of planning synchronously, but implementing asynchronously.

We conjecture that there are interesting results that remain to be proven about the stationary distribution of the Markov chain c_t. For instance, it seems plausible that the "lag," $\max_{v,w} c(v,t) - c(w,t)$ is $O(\sqrt{n})$ with high probability, which would tell us that, viewed globally, the asynchronous local counters look much closer to being a distributed global clock than we currently know.

References

[1] K. Nakamura. Asynchronous cellular automata and their computational ability. *Systems, Computers, Controls* **5**:5 (1974) 58–66.

[2] C. L. Nehaniv. Asynchronous Automata Networks Can Emulate Any Synchronous Automata Network *International Journal of Algebra & Computation*, **14**:5-6 (2004), 719–739.

[3] T. Toffoli Integration of Phase-Difference Relations in Asynchronous Sequential Networks. In: G. Ausiello and C. Bohm, eds., *Automata, Languages, and Programming (Fifth Colloquium, Udine, July 1978)*, Lecture Notes in Computer Science 62, Springer Verlag, (1978) 457–463.

Optimal Population Size in Island Model Genetic Algorithms

Neal Holtschulte

American Proverb: Two's company, three's a crowd. Genetic Algorithm Proverb: One's a hill climber and a thousand's random search. Population size is one of the key parameters affecting the success of genetic algorithms (GAs). Assuming a limited number of fitness evaluations (the most time-intensive factor in virtually all optimization problems), there exists an optimal population size for a genetic algorithm for a given application. Intuitively, a GA with population size one is a hill climber and a GA with maximal population size performs random search. Somewhere in between lies the sweet spot. The Island Model GA divides a single population into semi-isolated subpopulations connected by migration. On the extreme of high migration, the subpopulations function as a single large population. On the extreme of no migration, the subpopulations mights as well be independent runs of smaller population size GAs. Somewhere in between lies the sweet spot. In this paper we propose to explore the dynamics of optimal population size as a function of migration in island model GAs.

This is more of a research proposal than anything else. I am seeking constructive criticism.

Introduction

Genetic Algorithms (GAs) encompass a variety of search heuristics based on biological evolution. The traditional GA applies the operators: selection, crossover, and mutation to a population of candidate solutions in order to evolve better solutions to a given problem. Traditionally, population size is an input parameter to the GA and remains fixed throughout the course of search.

Modifying the population size can have a large impact on the performance of a GA. Understanding how and why pop size has such an impact will lead to improved performance for heuristic search algorithms and a better understanding of how they scale. Such insights may also be broadly applicable to populations of organisms or

collections of intelligent agents such as swarm robots.

For a fixed number of evaluations there exists an optimal population size that is problem dependent. That is, the quality of solutions discovered by a genetic algorithm in a fixed number of fitness evaluations varies based on the population size.

This is not surprising when one considers the extreme cases. On the small population end, a genetic algorithm with population size one is essentially a hill climber. Hill climbers ascend the nearest peak in the fitness landscape reachable by mutation but are then stuck atop the peak which may be a local, but not global, optimum. On the large population end, a genetic algorithm with population size equal to the total number of allowed fitness evaluations merely performs random search since the GA would only be able to initialize the population and determine the members' fitness values within the fitness evaluation limit. No selection, crossover, or mutation would be performed.

In Island Model GAs, individuals are divided up into subpopulations (islands). Individuals are then periodically exchanged between the islands (migration) over the course of the GA run.

The parameters added by the island model include: number of islands, migration size (the number or percentage of individuals to migrate), migration interval (number of generations between migrations), and topology, specifying which islands individuals migrate to. Further implementation decisions such as how migrants are selected and whether or not they replace individuals in the population to which they migrate must also be determined.

Related work

To our knowledge there has been no extensive empirical study of optimal population sizes in genetic algorithms across a wide variety of benchmark problems as a function of migration. However, many others have been interested in varying the population size for genetic algorithms.

Fernandez et. al. empirically studied population size for a fixed computational effort on a set of genetic programming benchmarks [3]. They found that for each benchmark there was a point of diminishing returns, after which, adding individuals to the population had no effect or degraded the search results.

Surprisingly, they found that for island GP models with migration the optimal population size remained constant regardless of the number of islands into which

the population was divided. For example, on one of the benchmarks, for a migration of 10% of individuals between the populations once every ten generations, two populations of 5,000 was more effective than two populations of 10,000. And, one population of 10,000 was more effective than one population of 15,000 (for a limited number of fitness evaluations, of course).

Though the authors varied the migration parameters, there was no follow up experiment to determine the effects of the different migration parameter values on optimal population size. In the extreme case of zero migration it seems unlikely that two populations of 5,000 or to be even more extreme, 5,000 populations of two would be optimal. Out goal is to tease apart the relationship between migration rate and optimal pop size in this paper.

Skolicki and De Jong [8] studied migration size and migration interval on island GA performance. They found that moderately large migration intervals and small migration size were optimal and that these parameter settings maintained high diversity, which was important for finding novel solutions. They fixed the population size throughout their experiments as they were not interested in the interaction of pop size with the other parameters.

A number of papers vary population size in an attempt to improve performance but do not make the population size itself the focus of study such as [4, 5] and [9] for multiobjective problems. [2] compares a handful of on-the-fly population resizing mechanism for best performance.

The authors of [10] are interested in both population size and island models, but their conclusions are general and their results are difficult to compare to the results in this paper due to differing methodologies. For example, they used a binary representation for all of their experiments. The binary string was converted to a float for problems such as F5 Rastrigin's function, which we also study, but chose to use a floating point chromosome for. Also, in some of their other experiments they either removed mutation to eliminate it as a complicating factor or used much larger population sizes and effort limits (ex: pop 5000, effort limit: 300,000).

They acknowledge that using a smaller population with mutation is more standard and follow up by performing such experiments.

Setup

The Problems

Without superstition, 13 problems are evaluated for optimal population sizes including: F1 through F9 (table 1), Royal Road, a simple deceptive problem, a one-dimensional optimization problem, and an NP-complete thread mapping problem.

F1 through F9 are used as benchmark problems in [6]. They are minimization problems exhibiting every combination of the binary properties: (uni/multi)modal, (a)symmetric, and (in)separable (with one repeat to make 9). We explored these problems in 10 dimensions using floating point numbers. More details on these problems can be found in table 1. The fitness landscape for F5 is shown in figure 4 and the landscape for F7 is shown in 6.

The Royal Road is a class of functions introduced in [7] to study the types of fitness landscapes on which GA's perform well. Our royal road is identical to the one described in [7]. With a 64 bit chromosome, 4 tiers, and an optimal fitness of 256.

A simple, piecewise linear, deceptive problem:

$$f(x_i) = \sum_{i=1}^{n} \begin{cases} (\frac{-x_i}{d} + 1)/dim, & \text{if } x_i < d \\ ((x_i - d) \cdot \frac{0.7}{1-d})/dim, & \text{otherwise} \end{cases}$$

where d is the deceptiveness parameter (counterintuitively, smaller d corresponds to greater deceptiveness) and dim is the number of dimensions. Thus the global optimum has fitness of 1.0 and the local optimum has fitness 0.7. The chromosome for this problem consists of an array of 10 floating point numbers.

A one-dimensional optimization problem:

$$f(x) = 2^{-2((x-0.1)/0.9)^2} sin(5\pi x)^6$$

For this problem, 32 bit binary chromosomes are used and are translated to integers using a gray code. Finally, the integers are divided by $2^32 - 1$ to get a float in the range 0 to 1.

The last problem is an NP-complete thread mapping problem in which 64 threads are mapped one-to-one to 64 cores on a multicore processor so as to minimize the communication cost between the cores. The cores lie in a 16x16 grid (without wrapping) and communication cost is the sum of the manhattan distances between communicating threads. Not all communication between threads is created equally. A fixed communication graph describes which threads communicate and the energy cost associated with the communication.

The representation of this problem is an array containing a permutation of the values 0 through 63. Special crossover and mutation operators are used to maintain the invariant that a chromosome is a permutation of 0-63 without duplicates.

Genetic Algorithm Settings

Pseudocode for the genetic algorithm used in all experiments is given in figure 2. Parameter settings used for most benchmark problems are summarized in table 3. Genetic operators are described in more detail and exceptions are also described below.

Tournament selection uses replacement. Meaning that an individual can be selected to be included in the next generation multiple times. Two individuals compete per tournament.

Gaussian mutation consists of replacing a gene with a value randomly selected from a gaussian distribution with mean equal to the gene value and standard deviation equal to 20% of the range (so if a gene has min 0 and max 1 then the standard deviation is 0.2).

The mutation rate was fixed at 0.01 per gene. That is, each gene had a 1% chance of being mutated per generation. More than one mutation could occur in the same chromosome in a single generation though such an event is unlikely.

The crossover rate was fixed at 0.9 per individual, meaning that each individual had a 90% chance of reproducing by crossover with another randomly selected individual per generation. Note that the number of generations varied based on the population size due to the fact that the number of fitness evaluations (effort) was chosen as the limiting factor.

Though these rates are relatively standard, parameter sweeps for a fixed population size of 80 individuals were performed on F7 Schwefel's function to verify that the rates are optimal. The parameter sweeps confirmed this.

Exceptions:

The mutation operator for the one dimensional optimization problem and Royal Road problem (which use binary chromosomes) flips the bit.

Mutation and crossover for the locality optimization problem must preserve the invariant that each integer value 0 through 63 appears exactly once in each chromosome. So inversion is used as the mutation operator and a special order crossover operator designed for traveling salesman-like problems (see [1]) is used for crossover.

F1 Sphere Model unimodal symmetric separable $F(x_i) = \sum_{i=1}^{n} x_i^2$ $-5.12 \leq x_i < 5.12$ $F(x_i)_{min} = F(0, 0, \cdots, 0) = 0$
F2 Ridge's Function unimodal symmetric inseparable $F(x_i) = \sum_{i=1}^{n} (\sum_{j=1}^{i} x_j)^2$ $-64 \leq x_i < 64$ $F(x_i)_{min} = F(0, 0, \cdots, 0) = 0$
F3 Exponential Function unimodal asymmetric separable $F(x_i) = \sum_{i=1}^{n} e^{ix_i} + \alpha$ $\alpha = \sum_{i=1}^{n} e^{-5.12i}$ $-5.12 \leq x_i < 5.12$ $F(x_i)_{min} = F(-5.12, -5.12, \cdots, -5.12) = 0$
F4 Modified Double Sum unimodal asymmetric inseparable $F(x_i) = \sum_{i=1}^{n} (\sum_{j=1}^{i} (x_j - j)^2)$ $-10.24 \leq x_i < 10.24$ $F(x_i)_{min} = F(1, 2, 3, \cdots, n) = 0$
F5 Rastrigin's function multimodal symmetric separable $F(x_i) = 10n + \sum_{i=1}^{n} (x_i^2 - 10cos(2\pi x_i))$ $-5.12 \leq x_i < 5.12$ $F(x_i)_{min} = F(0, 0, \cdots, 0) = 0$
F6 Griewank's function multimodal symmetric inseparable $F(x_i) = 1 + \frac{1}{4000} \sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n} \frac{x_i}{\sqrt{i}}$ $-512 \leq x_i < 512$ $F(x_i)_{min} = F(0, 0, \cdots, 0) = 0$
F7 Schwefel's function multimodal asymmetric separable $F(x_i) = \frac{1}{n} \sum_{i=1}^{n} (-x_i sin(\sqrt{\lvert x_i \rvert}) + \alpha$ $\alpha = 418.982887$ $-512 \leq x_i < 512$ $F(x_i)_{min} = F(420.968746, 420.968746, \cdots, 420.968746) = 0$
F8 Rosenbrock's saddle multimodal asymmetric inseparable $F(x_i) = \sum_{i=1}^{n-1} (100(x_i^2 - x_{i+1})^2 + (1 - x_i)^2)$ $-2.048 \leq x_i < 2.048$ $F(x_i)_{min} = F(1, 1, \cdots, 1) = 0$
F9 Whitley's function multimodal asymmetric inseparable $F(x_i) = \sum_{i=1}^{n} \sum_{j=1}^{n} (\frac{100(x_i^2 - x_{i+1})^2 + (1 - x_i)^2}{4000} - cos(100(x_i^2 - x_{i+1})^2 + (1 - x_i)^2) + 1)$ $-10.24 \leq x_i < 10.24$ $F(x_i)_{min} = F(1, 1, \cdots, 1) = 0$

6

Figure 1

```
#Elitism
best = getBestIndividual(population)

#Cull the expanded population down to pop_size
population = tournamentSelection(population, pop_size)

children = []
for p in population:
  if random < prob_crossover:
    cross p with a random individual in population
    children.append(resulting solutions)

mutants = []
for p in population:
  mutate p
  mutants.append(resulting solution)

for c in children:
  mutate c
  replace c with the mutated result

evaluate_fitness(children)

evaluate_fitness(mutants)

population = population + children + mutants + best
```

Figure 2

Population size:	varied
Crossover Probability:	90%
Mutation Probability:	1%
Crossover Operator:	One point
Mutation Operator:	Gaussian
Selection:	Tournament with replacement

Figure 3

Finding the Sweet Spot

> The vulgar crowd values friends according to their usefulness.
> - Ovid

We attempted to establish the optimal population sizes on the benchmark problems. For each problem we ran the genetic algorithm until 4000 unique fitness evaluations had been performed. By unique, we mean that solution, fitness pairs are cached, and later evaluation of a cached solution did not count towards the effort limit.

Population sizes tested include: 8, 40, 80, 120, 240, and 480 individuals. We ran 100 trials on each of the 13 benchmarks, recording the best final fitness value for each population size and then averaging.

> The intelligence of the creature known as a crowd, is the square root of the number of people in it.
> - Terry Pratchett

Surprisingly we found that for a large number of problems the smallest population size had the best performance. We verified this by running additional experiments with a true hill climber and found that it outperformed genetic algorithms of all population sizes on every problem except for the Royal road, deceptive problem, and F7 Schwefel's function.

The best average fitness values for 100 runs across a range of population sizes are shown for F5 Rastrigin's function (figure 5), F7 Schwefel's function (figure 7), and deceptive (figure 8). The GA with the smallest population size outperformed all others on the F5 function despite the rugged landscape as shown in figure 4. The fitness landscape of F7 is shown in figure 6.

Hill climbers outperforming genetic algorithms on so many difficult problems flies in the face of every intuition. Follow up experiments ferreted out the flaw in the experimental design: the effort limit was too low. Figure 9 shows a single run of a hill climber and multiple GA population sizes on F7 Schwefel's function. For this run, the effort limit was increased from 4000 to 40,000. Figure 9 shows that the hill climber and smaller GA populations rapidly converge and successfully exploit local optimum. The GA with population size 480 takes longer to converge, but when it finally does so, it finds a much higher quality (lower fitness since this is a minimization problem) than any of the other search strategies.

Clearly much larger population sizes need to be used for higher effort limits, at least for F7 Schwefel's function and probably for other benchmark problems as well.

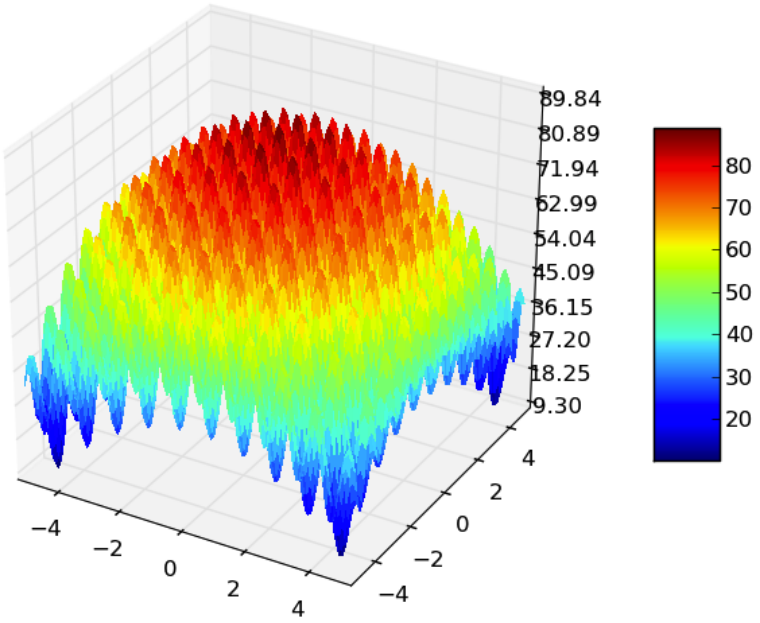


Figure 4: Fitness landscape of F5 Rastrigin's, a minimization problem.

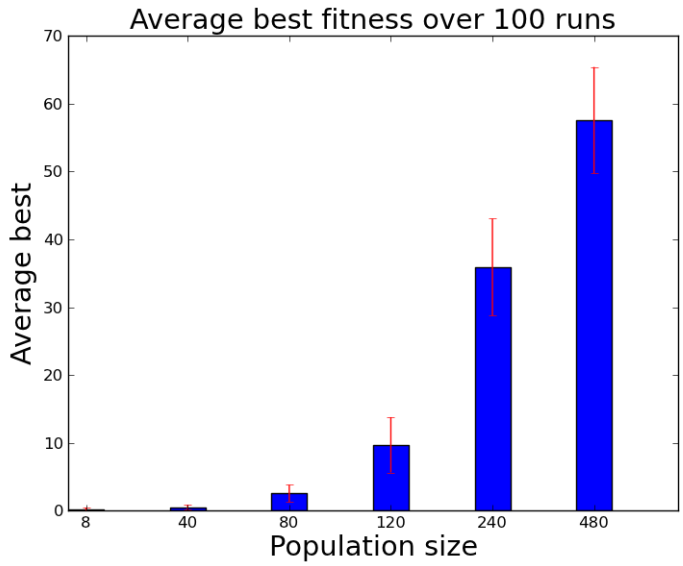


Figure 5: Average best fitness with standard deviation for varying population sizes for F5 Rastrigin's function, a minimization problem.

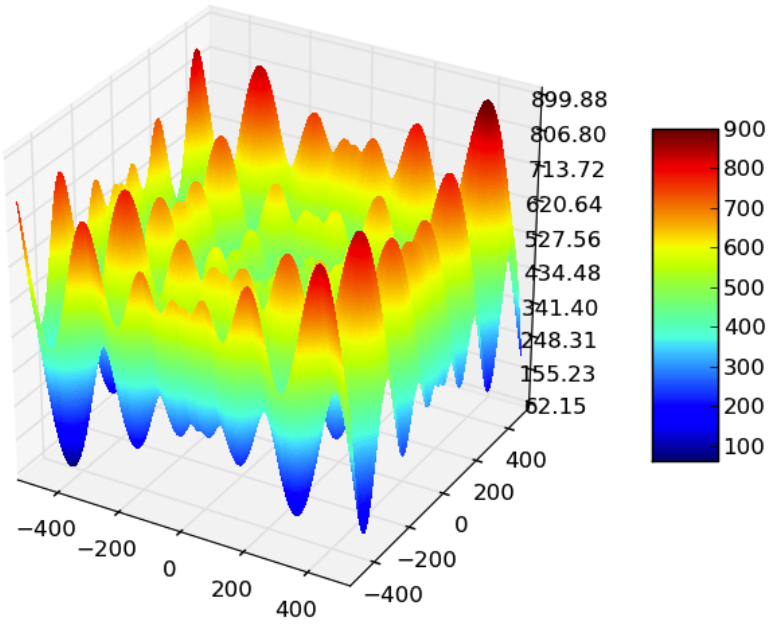


Figure 6: Fitness landscape of F7 Schwefel's, a minimization problem.

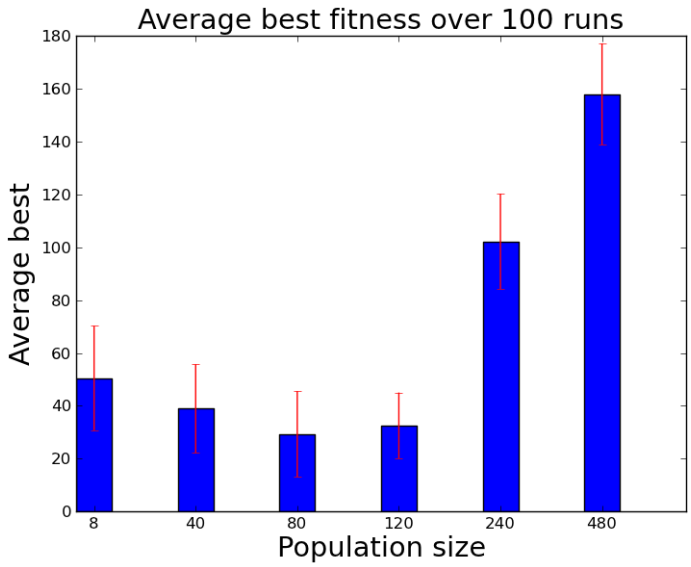


Figure 7: Average best fitness with standard deviation for varying population sizes for F7 Schwefel's function, a minimization problem.

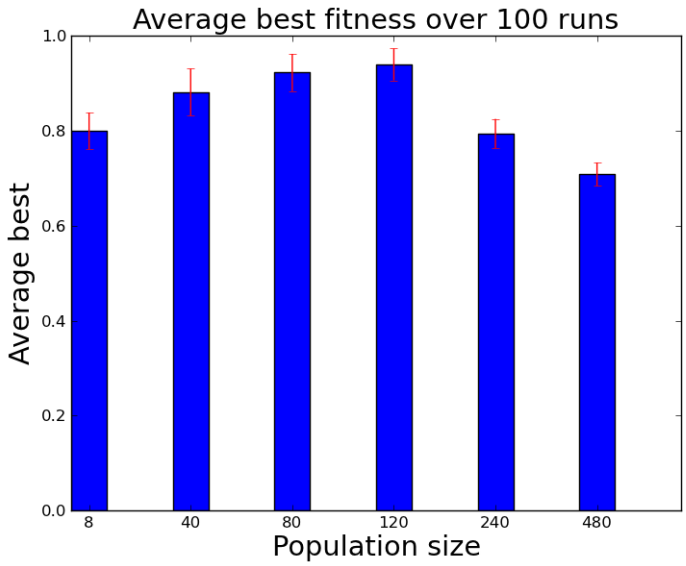


Figure 8: Average best fitness with standard deviation for varying population sizes for the deceptive problem, a maximization problem.

To avoid the problem of a specific effort limit creating a bias towards population sizes that converge just prior to reaching the limit we will instead evaluate the quality of a given population size based on how many fitness evaluations the population takes to reach the optimal fitness value.

Population sizes that get stuck in a local optimum, never to reach optimal fitness will be ended once the population's diversity passes below a certain threshold and after no improvements in best fitness are found after a specified number of fitness evaluations. Both of these cut-off parameters will have to be carefully crafted to prevent them from prematurely terminating actively searching GAs.

Some optimal fitness solutions are virtually impossible to find due to the high precision of floating point numbers. For such problems, the fitness landscape will be adjusted so that a GA gets credit for finding the optimal solution if it finds a solution within some epsilon of the true optimal.

Finally, we will get back to the main question, "What is the relationship between optimal population size and migration in island model genetic algorithms?"

All feedback is welcome.

Special thanks to Ben Edwards, Kenneth Letendre, and Professor Melanie Moses

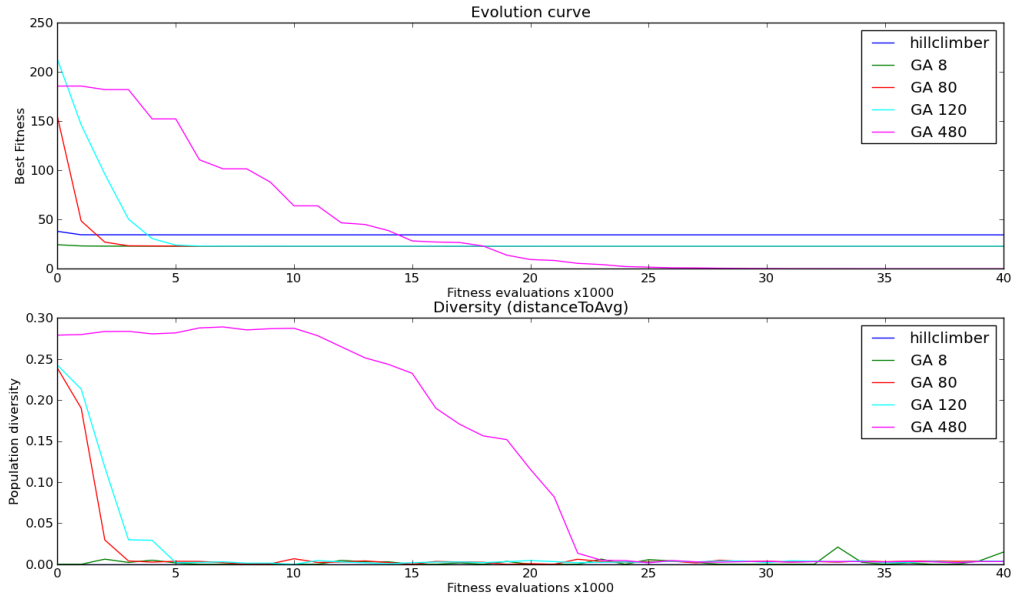


Figure 9: Top: Best fitness after a given number of fitness evaluations. Bottom: Population diversity measured by distance to average point.

for advice on this project.

References

[1] Buthainah Fahran Al-Dulaimi and Hamza A Ali. Enhanced Traveling Salesman Problem Solving by Genetic Algorithm Technique (TSPGA). In *Proceedings of World Academy of Science Engineering and Technology*, volume 28, pages 296–302, 2008.

[2] A. E. Eiben, E. Marchiori, and V. A. Valk. Evolutionary algorithms with on-the-fly population size adjustment. In *Parallel Problem Solving from Nature PPSN VIII, LNCS 3242*, pages 41–50. Springer, 2004.

[3] Francisco Fernández, Marco Tomassini, and Leonardo Vanneschi. An empirical study of multipopulation genetic programming. *Genetic Programming and Evolvable Machines*, 4:21–51, March 2003.

[4] Pedro Antonio Gutiérrez, César Hervás, and Manuel Lozano. Saw-tooth algorithm guided by the variance of best individual distributions for designing evolutionary neural networks. In *Proceedings of the 8th international conference on Intelligent data engineering and automated learning*, IDEAL'07, pages 1131–1140, Berlin, Heidelberg, 2007. Springer-Verlag.

[5] R.L. Haupt. Optimum population size and mutation rate for a simple real genetic algorithm that optimizes array factors. In *Antennas and Propagation Society International Symposium, 2000. IEEE*, volume 2, pages 1034 –1037 vol.2, 2000.

[6] Takuma Jumonji, Goutam Chakraborty, Hiroshi Mabuchi, and Masafumi Matsuhara. A novel distributed genetic algorithm implementation with variable number of islands. In *IEEE Congress on Evolutionary Computation*, pages 4698–4705, 2007.

[7] Melanie Mitchell, Stephanie Forrest, and John H. Holland. The royal road for genetic algorithms: Fitness landscapes and ga performance. In *Proceedings of the First European Conference on Artificial Life*, pages 245–254. MIT Press, 1991.

[8] Zbigniew Skolicki and Kenneth De Jong. The influence of migration sizes and intervals on island models. In *Proceedings of the 2005 conference on Genetic and evolutionary computation*, GECCO '05, pages 1295–1302, New York, NY, USA, 2005. ACM.

[9] K.C. Tan, T.H. Lee, and E.F. Khor. Evolutionary algorithms with dynamic population size and local exploration for multiobjective optimization. *Evolutionary Computation, IEEE Transactions on*, 5(6):565 –588, dec 2001.

[10] Darrell Whitley, Soraya Rana, and Robert B. Heckendorn. The island model genetic algorithm: On separability, population size and convergence. *Journal of Computing and Information Technology*, 7:33–47, 1998.

Implementation of an Embodied General Reinforcement Learner on a Serial Link Manipulator

Nicholas Malone[1], Brandon Rohrer[2], Lydia Tapia[3], Ron Lumia[4] and John Wood[5]

***Abstract*— BECCA (a Brain-Emulating Cognition and Control Architecture software package) was developed in order to perform general reinforcement learning, that is, to enable unmodeled embodied systems operating in unstructured environments to perform unfamiliar tasks. It accomplishes this through automatic paired feature creation and reinforcement learning algorithms. This paper describes an implementation of BECCA on a seven Degree of Freedom (DoF) Barrett Whole Arm Manipulator (WAM) undergoing a series of experiments designed to test the reinforcement learner's ability to adapt to the WAM hardware. In the experiments, the following is demonstrated, 1) learning to transition the WAM between states, 2) learning to perform at near optimal levels on one, two and three dimensional navigation tasks, 3) applying learning in simulation to hardware performance, 4) learning under inconsistent, human-generated reward, and 5) combining the reinforcement learner with Probabilistic Roadmap Methods (PRM) to improve scalability. The goal of the paper is to demonstrate both the scalability of the BECCA reinforcement learning approach using different formulations of the state space and to show the approach in this paper operating on complex physical hardware.**

I. Introduction

Robotic path planning and control is a challenging task and often requires intimate knowledge of a specific platform to build a path planner for it. Reinforcement learning is an alternative approach to hand designing a path planner. There are many different reinforcement learning techniques but they all have the machine learn how to find a path which maximizes a metric called the reward by exploring state-action sequences [1], [11], [5], [4], [3], [2], [12]. Abtahi et al. in [1] describe a reinforcement learning technique which combines deep belief networks with a function-based reinforcement learner. [11] combines a traditional reinforcement learning algorithm with additional input from a human trainer. The work in [5] deals with combining the approximation value function approach with the discretization approach to reinforcement learning. Cuccu's work in [4] uses a type of reinforcement learning for artificial neural networks operating on the mountain-car benchmark. Clouse's work on reinforcement learning in [3] is about combining a traditional reinforcement learner with apprentice learning or mimic learning. Finally, the work of Legenstein et al. discuses a technique for handling high dimensional inputs [12].

The Barrett Whole Arm Manipulator (WAM), as seen in Fig 1, is a 7 DoF robotic system. Instead of handcrafting a control and path planning algorithm for the WAM, machine learning techniques can be used to learn how to control and path plan the WAM. BECCA is a general reinforcement learner designed to learn how to control arbitrary robotic platforms. In this paper BECCA is implemented on subsets of the WAM platforms DoFs. The goal of this work is to first provide a proof of concept on a physical platform and then to investigate the scalability of different approaches.The reinforcement learner is combined with the Probabilistic Roadmap Methods (PRMs) in order to improve scalability.

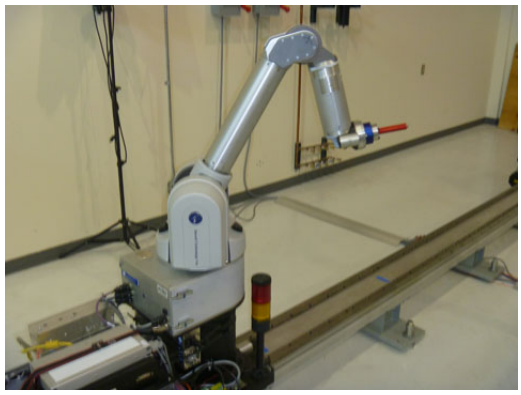


Fig. 1. Whole Arm Manipulator (WAM).

II. Related Work

Creating a general learning machine has been one of the grand goals of artificial intelligence (AI) since the field was born. Efforts to achieve this goal may be divided into two categories. The first category uses a depth first approach, solving problems that are complex, yet limited in scope, such as playing chess. The assumption underlying these efforts is that an effective solution to one problem may eventually be generalized to solve a broad set of problems. The second category emphasizes breadth over depth, solving large classes of simple problems. The assumption underlying these efforts is that a general solution to simple problems may be scaled

This work was supported by Sandia National Laboratories PO# 1074659

[1]N. Malone is a student at the University of New Mexico namlone@cs.unm.edu

[2]B. Rohrer is with Sandia National Laboratories brrohre@sandia.gov

[3]L. Tapia is with the Faculty of Computer Science, University of New Mexico ltapia@cs.unm.edu

[4]R. Lumia is with the Faculty of Mechanical Engineering, University of New Mexico lumia@unm.edu

[5]J. Wood is with the Faculty of Mechanical Engineering, University of New Mexico jw@unm.edu

up to address more complex ones. An example of the first category would be a master level chess playing agent, while an example of the second category would be an agent with the capabilities of an ant worker. The work described here falls into the second category, focusing on breadth. The motivating goal for this work is to find a solution to natural world interaction, the problem of navigating, manipulating, and interacting with arbitrary physical environments to achieve arbitrary goals. In this context, environment refers both to the physical embodiment of the agent and to its surroundings, which may include humans and other embodied agents. The agent design presented here is loosely based on the structure and function of the human brain and is referred to optimistically as a Brain-Emulating Cognition and Control Architecture (BECCA) [16], [17].

A Brain-Emulating Cognition and Control Architecture agent interacts with the world by taking in actions, making observations, and receiving reward (see Fig. 2). Formulated in this way, natural world interaction is a general reinforcement learning problem, [19] and BECCA is a potential solution. Specifically, at each discrete time step, it performs three functions:

1) reads in an observation, a vector $o \in \Re^m \mid 0 \leq o_i \leq 1$.
2) receives a reward, a scalar $r \in \Re \mid -\infty \leq r \leq \infty$.
3) outputs an action, a vector $a \in \Re^n \mid 0 \leq a \leq 1$.

Because BECCA is intended for use in a wide variety of environments and tasks, it can make very few assumptions about the environments beforehand. Although it is a model-based learner, it must learn an appropriate model through experience. There are two key algorithms to do this: an unsupervised feature creation algorithm and a tabular model construction algorithm.

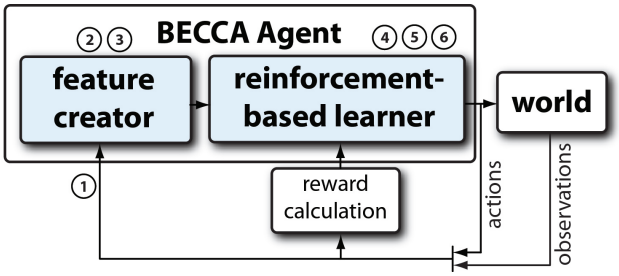


Fig. 2. At each timestep, the BECCA agent completes one iteration of the sensing-learning-planning-acting loop, consisting of six major steps: 1) Reading in observations and reward, 2) Updating feature set, 3) Expressing observations in terms of features, 4) Predicting likely outcomes based on an internal model, 5) Selecting an action based on the expected reward of action options, and 6) Updating the model.

The feature creator component identifies repeated patterns in the input vector [15]. It then groups loosely correlated elements of the input vector. The groups are treated as subspaces and unit vectors of these subspaces are features [15]. New inputs are also projected onto existing features and the single feature in each group which has the greatest response is turned on while all others in that group are turned off [14] [15] [18].

The reinforcement learning component receives feature activity, reward, and direct input from the environment. Each feature is associated with an approximate reward. It keeps track of recent actions and recent features in working memory which is then used to update the model. The actual model is a table of cause-effect pairs. The cause is the working memory and the effect is the current feature. Considering this in standard reinforcement learning language, the model can be thought of as a sequence of state-action pairs. Entries in the table which are rarely observed are deleted from the model [14] [15] [18].

To chose an action the reinforcement learner compares the current working memory to the entries in the model and selects the entry which both matches the current working memory and which has the highest recorded reward. With a set probability, an exploratory action is chosen instead [14] [15] [18].

III. Methods

The methods section first describes the WAM hardware, then the interface with the WAM, and finally, how a general task is built for BECCA to utilize the WAM interface. The reinforcement learner is designed for general purpose learning on any platform but there is still a small amount of configuration needed to allow it to interface with a specific hardware. A task is a framework for incorporating the specific requirements for any hardware. A task is a translator for the action vector produced by BECCA to the hardware, and a translator for the sensory information into an input vector for the reinforcement learning approach. The task also is responsible for calculating the reward for any given state.

A. Robotic Hardware

The Barrett Whole Arm Manipulator (WAM) platform used in the experiments is a seven degree of freedom (DoF) robotic arm as seen in Fig. 1. It is a cable driven system controlled with joint position encoders and torque sensors. For the experiments in this paper, the WAM has been connected to a GE Intelligent Platforms reflective memory network in a spoke design that allows multiple computers to share memory at speeds ranging from 43 MB/s to 170 MB/s. The reflective memory network allows remote computers to handle the planning, learning processing, and sensor processing, while leaving a small and fast computer on-board the WAM to handle simple motion control.

B. WAM Interface

The WAM is connected to a xPC Target Kernel running Matlab Simulink 7.7.0 R2008b [13]. The controller for the WAM is written in Simulink and interfaces with remote computers via the reflective memory network. The Simulink code responsible for directly issuing commands to the WAM, henceforth the WAM controller, receives a command vector by reading a specific block of reflective memory. The command vector is a length seven vector containing the desired joint angles in radians of each for the seven WAM joints.

The WAM controller, upon receiving a command vector, places the command vector into a buffer, which only stores

one move. The command vector is first sanitized so that each entry is within the WAM's joint limits. If the WAM is not executing a move, it compares its current location to the command vector buffer. If the command vector buffer is sufficiently different from the current location, the WAM controller computes a linear interpolation in joint space between the two joint angles and executes the path within the allowable WAM workspace. However, the velocity follows a fifth-order smooth polynomial as seen in Fig 3, and is used both for safety and for mimicking biological motion [6]. Slow beginnings and endings to moves provide safe joint torques. In the current architecture a move cannot be interrupted.

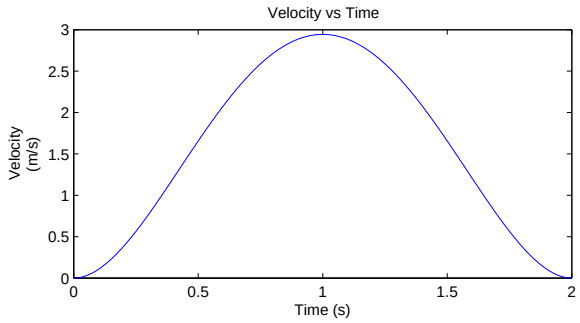


Fig. 3. Example Velocity Profile for a Single Joint.

C. Tasks

BECCA is designed to be a reinforcement learner for general robotic applications. Thus, there is a simple interface between the external world and its internal representation of the world. At the simplest level, it takes a vector of inputs and transmits a vector for outputs. The structure of the inputs and outputs is intended to be irrelevant to BECCA and so it must learn the structure of both. Therefore, a designer must define an input vector and an action (output) vector as well as a reward structure for each task. The task must also define how to interpret an action vector, and how to translate a sensory information into an input vector. The manner in which the task interprets outputs and delivers inputs to BECCA is completely at the discretion of the task designer. Algorithm 1 is a pseudocode example of an interpretation of an action vector, for a 1 DoF task.

For the 1 DoF task the joint space is partitioned into equally spaced bins and BECCA is constrained to move between bins. Algorithm 1 calculates the direction and number of bins the action vector specifies to move. Line 1 first calculates the offset move for a given action vector. For this task the action vector is treated much like a stepper motor move. The vector has 8 components that are either 0 or 1. Vector component 1 corresponds to a move of one bin to the right while component 2 corresponds to a move of one bin to the left. Components 3 and 4 represent a move of two to the right or two to the left and so on for the rest of the components. A total move is calculated by summing together all of the components to determine how many bins to move and in which direction. Lines 2 and 3 calculate

Algorithm 1 interpretActionVector(action)

1: $move \leftarrow 1 * action[1] - 1 * action[2]$
$+2 * action[3] - 2 * action[4]$
$+3 * action[5] - 3 * action[6]$
$+4 * action[7] - 4 * action[8]$
2: $currentPos \leftarrow getJointPositionFromWam()$
3: $actualMove \leftarrow (move * 0.3142 + currentPos[4])$
4: **if** $actualMove < jointMinLimit$ **then**
5: $actualMove \leftarrow jointMinLimit$
6: **end if**
7: **if** $actualMove > jointMaxLimit$ **then**
8: $actualMove \leftarrow jointMaxLimit$
9: **end if**
10: **return** $actualMove$

the current position of the arm and then calculate the joint angle that the arm should be at given the desired bin move. Lines 4 through 9 then sanitize the actual move to be within the joint limit constraints. It is important to note that at initialization, BECCA does not know the transition function between states, where a state corresponds to a particular input vector, and an input vector is a vector of all zeros except for the bin the WAM is currently in is a 1. It must learn what each action vector does in a given state.

A task is a specific instance of a problem for BECCA to learn. Algorithm 2 outlines the basic steps in a simple task. In line 1 the action vector is retrieved from BECCA and then interpreted by the task in line 2 by calling Algorithm 1. In line 4 the move is sent to the WAM via the WAM interface discussed in section III.B. An input vector is then generated given the new state and sent to the reinforcement learner in lines 5 through 10. Finally, a reward is calculated based on the new state by lines 11 through 17. In the Algorithm 2 example, there is a single reward bin, which is given a reward of +10, the edge bins are punished by a reward of -10 and any other bin is punished by a reward of -1.

IV. Experiments

The experiments section starts with a simple binning formulation for a 1-DoF pointing task. It then progresses to a 2-DoF pointing task to demonstrate the scalability issues reinforcement learners have with specific problem formulations. The manual training section then demonstrates that BECCA can be used in a real time human-trained environment for simple tasks. This is important since in order for learning robots to be useful they will need to be trained by human operators in specific tasks. Finally, the PRM section proposes a solution to the scaling issues presented by the simple binning formulation.

A. 1-DoF Task

The first experiment is a 1-DoF task. On the WAM joint 4, the elbow joint, is used for the single DoF. Joint 4 has a range of motion from 0 radians to π radians. For simplicity, the joint space is divided into 10 equally spaced bins, such that any angle between 0 and 0.3142 radians is bin 1, any

Algorithm 2 Task Loop

```
loop
    action ← BECCA.getAction
    move ← interpretActionVector(action)
    sendMoveToWAM(move)
    waitForMoveToFinish()
    currentPos ← getJointPositionFromWam()
    bin ← findBin(currentPos, numBins)
    binVector = [0, 0, 0, 0, 0, 0, 0, ..., 0]
    binVector[bin] ← 1
    BECCA.inputVector = binVector
    if bin == rewardBin then
        BECCA.reward ← 10
    else if isEdgeBin(bin) then
        BECCA.reward ← −10
    else
        BECCA.reward ← −1
    end if
end loop
```

angle greater than 0.3142 and less than 0.6284 radians is in bin 2 and so on until π radians. The binned joint space becomes the state space and is used as the input vector to BECCA. The input vector to BECCA is then of length 10, and has zeros in every position except for the current bin. For example if the WAM's joint 4 is at angle 0.1125 radians then it is in bin 1, and the input vector is: [1,0,0,0,0,0,0,0,0,0].

The action vector sent by BECCA for the 1-DoF task is a length 8 vector and each entry in the vector is constrained to be either a 0 or a 1. For the 1-DoF task, the action vector is interpreted by Algorithm 1. The action vector for the 1-DoF task was chosen to be a simple design without significant regard for how well the reinforcement learner would handle it. Algorithm 1, shows how the action vector for the 1-DoF task is interpreted. Once the action vector and the input vector operations have been specified they can be used in the 1-DoF task. Lastly, a reward function must be specified. In this experiment, the basic algorithm (Algorithm 2) is altered so that bin 5, the center bin, is given a reward of +10, and every other bin is given a reward of -1 except the edge bins 1 and 10 which are given a -10 reward.

BECCA is first trained in simulation and then ported to hardware partially through the experiment. Each data point is the cumulative reward the reinforcement learner receives after 100 iterations (100 iterations is a block).

Fig. 4 is an average of 10 runs on the 1-DoF task. BECCA was first trained in simulation and then ported to hardwar at block 25. Averaging the runs slightly smoothes out the learning curve for BECCA and better illustrates the climb to optimal performance at 700 units of reward. Fig. 4 is a proof of concept for interfacing with a specific task. It appears that BECCA does poorly, reaching only approximately 700 units of rewards, however there is a 30 percent exploration rate at minimum, thus on the simple 1-DoF task 700 units of reward is approximately optimal due to the -10 edge punishment and the fact that some exploratory actions have no effect. Here optimal is receiving maximal reward.

It should be noted that even on the relatively small state space of 10 bins and an action vector of length 8, BECCA still takes on average 3,000 iterations to converge on optimal behavior. 3,000 iterations with an average move time of 2 seconds on the WAM is approximately 1.7 hours of operation time on the WAM, which presents a problem for larger state/action spaces. The design of the WAM controller interface allows for BECCA to be run in simulation mode until convergence and then be connected to the WAM. A simulation of the WAM is not a perfect matching to the actual hardware due to idiosyncrasy of the hardware differing minutely from the theoretical. However, BECCA is a general learning program and is able to compensate for the difference between simulation and real hardware after exposure to the hardware. Thus, the experiments can be safely run in simulation, which operates significantly faster than hardware and reduces the time it takes BECCA to learn a task. An iteration in simulation takes a fraction of second, while in hardware a single iteration takes approximately 2 seconds. In Fig. 4 the structures, which have been learned, are ported to the WAM hardware at block 25.

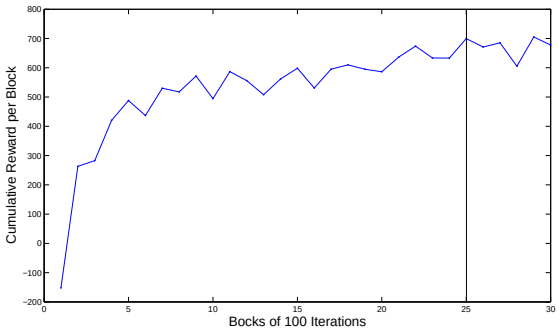


Fig. 4. Average of 10 1 DoF Task Learning Curves. The black vertical bar shows the transition from simulation to the WAM hardware.

B. Manual Training

BECCA is also capable of being trained manually. In the current version, version 0.3.9, manual training can be tedious given the number of iterations required for convergence. To compensate for the convergence time, a very simple 3 bin task with an action vector of length 4 is used during the manual training experiment. A trainer can reward or punish BECCA at any time, however only the last reward or punishment is registered per move, where a move is a single loop through Algorithm 2. Thus, this task differs from the normal training task in that the reward function is at the discretion of the trainer. The trainer can choose to change how BECCA is rewarded at will. Standard training tasks have a reward function which is coded directly into the task. The 1-DoF manual training task is running on the WAM robot without simulation bootstrap training. The trainer only rewards BECCA when joint 4 is in the middle bin, which places the elbow joint at $\pi/2$.

Fig. 5 shows the results of a typical manual training task. In Fig. 5 BECCA is only rewarded for being in state 2. The trainer sometimes intentionally fails to give reward for being in state 2, to demonstrate that BECCA can handle inconsistent rewards which it will likely receive in practice from a human trainer. The trainer never gives punishment during the experiment. Fig. 5 is slightly different from the previous cumulative reward figure in that the block size is only 5 iterations, and the reward is at most +10 per iteration making the maximum 50 units of reward. The cumulative reward seen in Fig. 5 shows that BECCA has converged at around 50 iterations (block 10 on the independent axis). The rapidity of convergence is due to the simplicity of the task, but it demonstrates that BECCA can learn a task with inconstant reward and can be trained by hand.

Fig. 6 shows the percentage of time spent in each bin. The blue vertical striped bar shows the overall percentage of time BECCA has spent in a bin, while the red horizontal striped bar shows the percentage of time spent in a bin for the last 5 iterations. On average 70 percent of its time is spent in bin 2, which is the only bin it receives reward in. 70 percent is optimal due to a 30 percent exploration rate. The red bar being at 100 percent in bin 2 indicates that BECCA chose to spend all of its time in bin 2 over the last 5 iterations. The results of both Fig. 5 and Fig. 6 show that BECCA is correctly learning to select bin 2 over the other two bins during the manual training task.

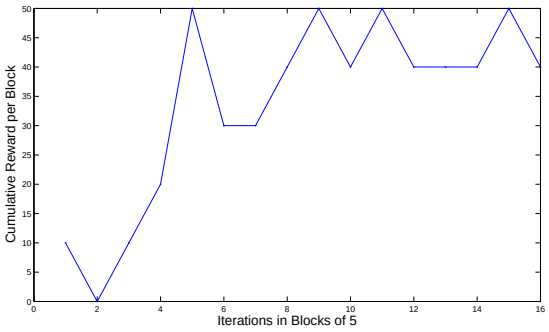


Fig. 5. Manual Training 1 DoF Task Learning Curve.

C. 2-DoF Task

The previous two experiments have been on a single degree of freedom and have had relatively small and simple state spaces and action vectors. The 2-DoF task has an exponentially larger state space than the 1-DoF task. The 2 degree of freedom task uses joint 2 and joint 4. Joint 2 has a range of motion from -1.99 radians to 1.99 radians. Both joint 2 and joint 4 are divided into 10 bins each, meaning that the state space is now a 10 by 10 matrix of states. The action vector must also be enlarged to 16. The first 8 components control joint 2 and the second 8 components control joint 4 in the same way as the first 1-DoF task. Thus the state spaces from the 1-DoF and 2-DoF task can be compared. The state space for the 1-DoF task had 10 states with an

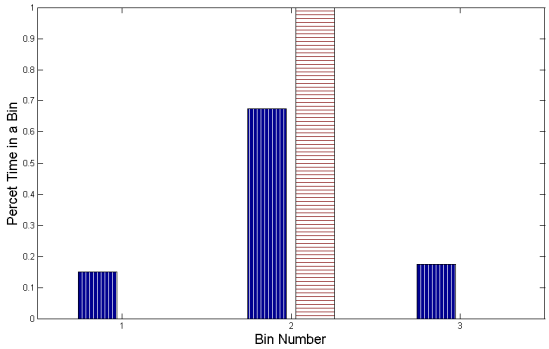


Fig. 6. Percentage of Time BECCA Spends in a Bin. This graph corresponds to block 15 in Fig 5. The red horizontal striped bar shows that BECCA chose to spend every step in bin 2, and thus received maximum reward for that block.

action vector of length 8, resulting in a state/action space of $10 * 2^8$ because BECCA can send an arbitrary bit string of length 8. The state space for the 2-DoF task has 100 states and an action vector of length 16 giving a state/action space of $100 * 2^{16}$. The reward function has been modified to $Reward = rJ2(Joint2Bin) + rJ4(Joint4Bin)$ where rJ2 and rJ4 are defined as:

$$rJ2(x) = \begin{cases} 10 & \text{if } x = 5 \\ -10 & \text{if } x = 0 \vee x = 10 \\ -1 & \text{otherwise} \end{cases}$$

$$rJ4(x) = \begin{cases} 10 & \text{if } x = 5 \\ -10 & \text{if } x = 0 \vee x = 10 \\ -1 & \text{otherwise} \end{cases}$$

In rJ2 and rJ4 bin 5 corresponds to the middle bin, and bins 0 and 10 correspond to the first and last bin, so the robot will be in a right angle when it is in the correct location.

Fig. 7 shows the cumulative reward for the 2-DoF task. After approximately 5000 iterations the reward stabilizes to an oscillation between 500 and 1500 units of reward. Based on the reward structure the maximum reward is 2000 per block, which indicates that the 2-DoF task is performing under optimum. The reason for the underperformance can be better seen by looking at Fig. 8 and Fig. 9, which show the percentage of time each joint spends in a particular bin. Fig. 8 indicates that BECCA spends the majority of its time in bin 5 for joint 2, but Fig. 9 indicates that not enough time is spent in bin 5 for joint 4. The underperformance stems from not fully learning the large state/action space. If BECCA correctly finds joint 2 it still must find joint 4 simultaneously in order to locate the optimal state. An action of length 16 has 2^{16} possible bit vectors, and there are 100 states, thus BECCA would have to visit $100 * 2^{16}$ state action combinations to fully explore the environment.

D. Probabilistic Roadmap Methods and BECCA

PRMs are a path planning technique used with robots with high DoFs to reduce the complexity searching in a high-dimensional and continuous space of possible conformations.

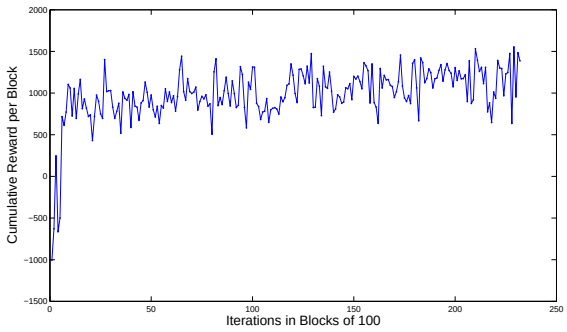


Fig. 7. 2-DoF Task Cumulative Reward.

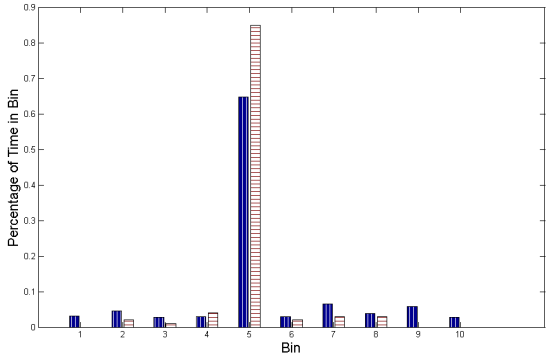


Fig. 8. 2-DoF Task Percentage of Time Spent in Each Bin for Joint 2. The blue vertical striped bar shows the cumulative percentage of time spent in a bin, while the red horizontal striped bar shows percentage of time spent in a bin during the last block.

They have been applied to a variety of complex robot types including manipulators [9], walking robots [8], and nonholonomic robots [7]. PRMs tackle the planning problem by working in *conformation space* (C-space) rather than the workspace. In regards to joints, this means that each DoF of a joint is mapped to a single dimension in C-space. Then, the path planning problem is reduced to finding a sequence of feasible states in this conformation space, those in C-free.

PRMs work by building a roadmap of possible feasible motions in C-free [10]. They do this by randomly selecting points in C-free. Then, nearby points are connected by simple connection methods. Nearby can be defined by low-cost Euclidean distance calculation to identify the k nearest neighbors. Connection can be achieved using straight-line interpolation.

The goal of incorporating PRMs was to help guide the searching. For example, BECCA was successful at automatically searching the large search space with 2 joints, however, it struggled. This would be magnified with a 3 joint problem moving from 100 states to 1,000 states, which would increase the complexity at least by a factor of 10. Incorporating PRMs allowed us to reduce the state space for 3-DoF tasks to the number of nodes in the roadmap. For the results shown, the state space has 50 nodes, but the number of nodes can be adjusted.

For the 3-DoF task, joints 1, 2, and 3, are mapped into a 3

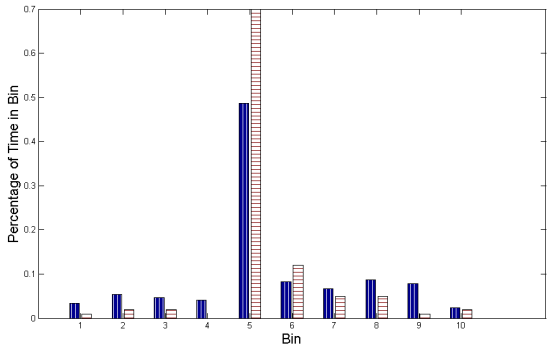


Fig. 9. 2-DoF Task Percentage of Time Spent in Each Bin for Joint 4. The blue vertical striped bar shows the cumulative percentage of time spent in a bin, while the red horizontal striped bar shows percentage of time spent in a bin during the last block.

dimensional C-space. Then, fifty random points are sampled in the C-space using a uniform distribution. The fifty points are then connected probabilistically based on the distance between the points, such that closer points have a higher probability of being connected. Fig. 10 shows an example of a PRM generated for the 3-DoF task.

Fig. 11 shows the cumulative reward per block for BECCA operating on the 3-DoF PRM task. The maximum reward that can be receive per iteration is 100, making the maximum per block 10,000 units of reward. The reward structure for the PRM task assigns a reward of 100 to the target node, a reward of 10 to all neighbors of the target node, and a reward of 1 to the neighbors of the neighbors. Every other node is given a reward of 0. The goal state is chosen at random from the 50 points, without loss of generality. BECCA's action vector is interpreted as which neighbor to transition to. Each node in the PRM is numbered and the input vector is 50 long, with each entry in the vector corresponding to a particular numbered node. All values in the vector are set to 0 except for the current node's number is set to 1. The PRM covers a wide area in the WAM's range of motion, but only takes 900 iterations to reach a very high cumulative reward. 900 iterations is significantly fewer than the 5,000 iterations required for the 2-DoF task to converge, which indicates that PRM's are very effective at reducing the convergence time of BECCA.

To better see the convergence speed of the PRM method compared to the engineering solution presented in the first 3 experiments, another experiment was performed as seen in Fig. 12. In this last experiment, two 3-DoF tasks are created, a simple and a hard task, using the same method as the 2-DoF task except a third joint is added. The simple 3-DoF task has 3 bins per joint, and an action vector of length 12. The hard 3-DoF task has 4 bins per joint, and an action vector of length 18. Both simple and hard tasks are rewarded by $+(100/3)$ for being in bin 2 and receive 0 reward for being in any other bin. The reward structure has been altered so that the simple and hard tasks have a reward structure more similar to the PRM task. The altered reward structure has a maximum reward of 100 per iteration

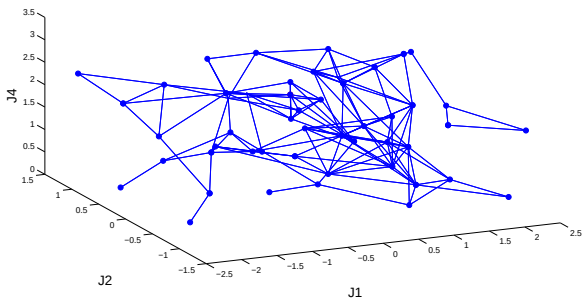


Fig. 10. Probabilistic Road Map for a 3-DoF WAM Task. Vertices are possible configurations. Edges are possible transitions between configurations.

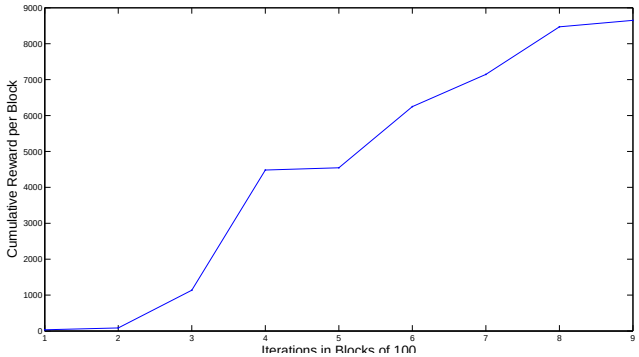


Fig. 11. PRM Cumulative Reward for a 3-DoF Task.

and thus 10,000 per block, just like the PRM experiment. The simple task has 27 possible states. The hard task has 64 possible states and the PRM has 50 states. Thus, the simple and hard tasks bound the PRM in number of states. However, it is important to note that the simple and hard tasks have larger action vectors than the PRM due to how they were engineered. Fig. 12 shows that the PRM method converges much faster than either the simple 3-DoF or hard 3-DoF task. The PRM method has reached the optimal of 7,000 units of reward by around 1,000 iterations while, the simple 3-DoF task has only reached approximately 6,000 units of reward by 7,000 iterations. The 3-DoF hard task has only reached approximately 3,500 by 7,000 iterations. Thus we can see that the PRM task converges much faster than either the simple or hard task.

To further show the scalability of the PRM approach we produce Fig. 13 which plots the average reward of 10 runs for each DoF from 1 to 7. This graph confirms that PRM-BECCA is unaffected by the Degrees of Freedom with a constant number of nodes. However, there is a problem with just testing the Degrees of Freedom and holding the number of nodes constant. By holding the number of nodes in the PRM constant, the density of nodes decreases as the DoF increases. Thus we must also test to see how BECCA scales with the number of nodes.

In the following experiments we vary the number of nodes from 60 to 200 in steps of 20, and the k neighbor parameter is

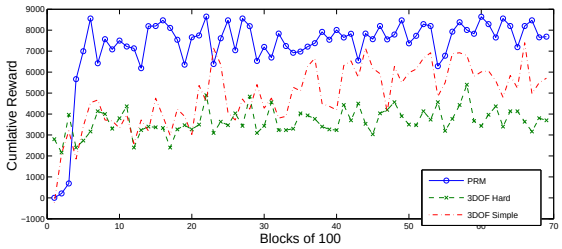


Fig. 12. Cumulative reward for PRM, 3-DoF Simple, and 3-DoF Hard tasks. The 3-DoF simple task has 3 bins per joint, giving a state space of 3^3. The 3-DoF hard task has 4 bins per and an action vector length of 18, giving a state space of 4^3. The PRM task has 50 points which correspond to 50 states.

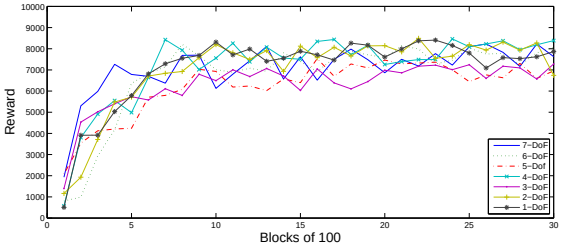


Fig. 13. Cumulative reward per block of 1-DoF to 7-DoF with PRMs.

set to 4. Since the previous experiment showed that BECCA would converge at the same number of steps regardless of DoF we chose to do this experiment with 3 DoF. Again 10 runs are done for each number of nodes and the results are averaged. Fig. 14 shows the average cumulative reward for each test. It shows us that BECCA also converges at the same time regardless of number of nodes in the graph. Fig. 14 is practically indistinguishable from Fig. 13, thus showing that BECCA converges at the same rate regardless of DoF and regardless of the number of nodes in the PRM.

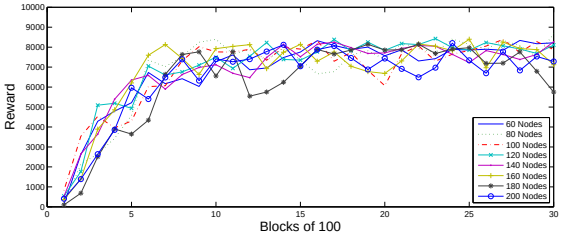


Fig. 14. Reward per Block for Varying Number of Nodes.

It is important to note that this is a novel use of PRMs. In previous work, they have been used to plan the motions for complex robot systems [7], [8], [9]. However, by integrating PRMs with BECCA, we are able to demonstrate automatic learning of controls to achieve motion in complex problems.

V. Conclusions

BECCA is intended to be a general reinforcement learning operating in unmodeled environments. The experiments in this paper demonstrate BECCA running on a single WAM

platform under different constraints. BECCA performs very well on small state/action spaces as seen in the 1-DoF task, but struggles under larger state/action spaces as seen in the 2-DoF task. Large state/action spaces are a problem for many reinforcement learning algorithms. The experiments in this paper show that BECCA can learn how to operate a complex machine such as the WAM in state/action spaces with varying complexity. However, the experiments also show that the complexity of those environments has a large impact on the convergence time of BECCA. The hope is that BECCA can learn any unconstrained environment, but the complexity of the environment is such a large factor that realistically the state/action space of the environment has to be carefully engineered to insure feasible convergence times.

The PRM formulation of the problem demonstrates that careful construction of the state space allows the reinforcement learner to overcome the scalability issues. BECCA scales very nicely from 1 to 7 DoF and under varying numbers of nodes in the roadmap using the PRM formulation. However, it fails to scale using the binning formulation. Therefore, PRMs work to improve the efficiency of the BECCA reinforcement learning agent.

VI. ACKNOWLEDGMENTS

This work was supported by Sandia National Laboratories PO# 1074659. Tapia supported in part by the National Institutes of Health (NIH) Grant P20RR018754 to the Center for Evolutionary and Theoretical Immunology. We also thank Dr. Dave Vick for his help with the robotic hardware setup.

REFERENCES

[1] ABTAHI, F., AND FASEL, I. Deep belief nets as function approximators for reinforcement learning, 2011.

[2] ARGALL, B. D., CHERNOVA, S., VELOSO, M., AND BROWNING, B. A survey of robot learning from demonstration. *Robot. Auton. Syst. 57* (May 2009), 469–483.

[3] CLOUSE, J. On integrating apprentice learning and reinforcement learning. Tech. rep., University of Massachusetts, Amherst, MA, USA, 1997.

[4] CUCCU, G., L. M.-S. J., AND GOMEZ, F. Intrinsically motivated neuroevolution for vision-based reinforcement learnin. *International Conference on Development and Learning and Epigenetic Robotics* (August 2011).

[5] FERNÁNDEZ, F., AND BORRAJO, D. Two steps reinforcement learning. *Int. J. Intell. Syst. 23* (February 2008), 213–245.

[6] FLASH, T., AND HOGANS, N. The coordination of arm movements: An experimentally confirmed mathematical model. *Journal of neuroscience 5* (1985), 1688–1703.

[7] HASHIM, S., AND LU, T.-F. A new strategy in dynamic time-dependent motion planing for nonholonomic mobile robots. In *Proceedings of the 2009 international conference on Robotics and biomimetics* (Piscataway, NJ, USA, 2009), ROBIO'09, IEEE Press, pp. 1692–1697.

[8] HAUSER, K., BRETL, T., CLAUDE LATOMBE, J., AND WILCOX, B. Motion planning for a sixlegged lunar robot. In *The Seventh International Workshop on the Algorithmic Foundations of Robotics* (2006), pp. 16–18.

[9] JUNG-JUN PARK, J.-H. K., AND SONG, J.-B. Path planning for a robot manipulator based on probabilistic roadmap and reinforcement learning. In *International Journal of Control, Automation, and Systems* (2008), pp. 674–680.

[10] KAVRAKI, L., SVESTKA, P., CLAUDE LATOMBE, J., AND OVERMARS, M. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. In *IEEE International Conference on Robotics and Automation* (1996), pp. 566–580.

[11] KNOX, W. B., AND STONE, P. Augmenting reinforcement learning with human feedback. In *ICML 2011 Workshop on New Developments in Imitation Learning* (July 2011).

[12] LEGENSTEIN, R., WILBERT, N., AND WISKOTT, L. Reinforcement learning on slow features of high-dimensional input streams. *PLoS Comput Biol 6*, 8 (08 2010), e1000894.

[13] MATLAB. *version 7.7.0 R2008b*. The MathWorks Inc., 2007.

[14] ROHRER, B. Biologically inspired feature creation for multi-sensory perception. *BICA* (2011).

[15] ROHRER, B. A developmental agent for learning feature, environment models, and general robotics tasks. *ICDL/Eprirob* (2011).

[16] ROHRER, B. A developmental agent for learning features, environment models, and general robotics tasks. In *Joint IEEE International Conference on Development and Learning and on Epigenetic Robotics* (2011).

[17] ROHRER, B. An implemented architecture for feature creation and general reinforcement learning. In *Workshop on Self-Programming in AGI Systems, Fourth International Conference on Artificial General Intelligence* (2011).

[18] ROHRER, B. BECCA: Reintegrating AI for natural world interaction. *AAAI Spring Symposium on Designing Intelligent Robots: Reintegrating AI 2012* (2012).

[19] SUTTON, R., AND BARTO, A. *Reinforcement learning: An introduction*. Adaptive computation and machine learning. MIT Press, 1998.

Quantification of Uncertainty in Parameters Characterizing Within-Host West Nile Virus Infection

Soumya Banerjee[1], Melanie Moses[2], Alan S. Perelson[3,*]

1 Department of Computer Science, University of New Mexico, Albuquerque, New Mexico, USA

2 Department of Computer Science, University of New Mexico, Albuquerque, New Mexico, USA

3 Theoretical Biology and Biophysics, Los Alamos National Laboratory, Los Alamos, New Mexico, United States of America

∗ **E-mail: soumya@cs.unm.edu**

Abstract

West Nile virus (WNV) is a neurotropic flavivirus that has emerged globally as a significant cause of viral encephalitis. Currently, little is known about the within-host viral kinetics of WNV during infection. We used a series of mathematical models of increasing complexity to examine WNV dynamics in mice and birds. To the best of our knowledge, this is the first effort to model within-host dynamics of WNV. We use a computationally intensive method to quantify the uncertainty in parameter estimates given uncertainty in input parameters. We set up a framework to explore really large search spaces after imposing constraints from biology. Our method of quantifying uncertainty estimates of model parameters in terms of uncertainty in input parameters could be more generally applicable to modeling of other diseases where precise estimates of input parameters are hard to obtain.

These go to eleven: Cranking up the knobs on IDS scaling performance

Sunny James Fugate
University of New Mexico
Department of Computer Science
Albuquerque, New Mexico

ABSTRACT

Signature-based intrusion detection system (IDS) approaches represent the brunt of modern threat detection methods. This is primarily due to their specificity and low false-positive rates and in spite of scalability issues. The inherent scaling issues have meant that measurements of these systems generally ignore the scaling of systems beyond conventional parameter spaces. In particular, while signature-based systems are conventionally measured for total packet processing throughput and false alarm rates, performance is highly dependent on ruleset size. While IDS packet processing performance may appear to be well understood, IDS scaling performance has not been adequately characterized beyond available rulesets. In this paper I present my measurement methods, describe a straightforward method for generating large random rulesets, and present an analysis of the scaling performance of the Snort IDS system.

1. INTRODUCTION

Signature-based detection systems rely on explicit patterns within input traffic. Such approaches offer very precise detection of known threats at the cost of poor recall and poor coverage in the face of new threats and new variants of old threats. On many networks, *qualified events* (those resulting in an alert within the IDS) account for only a small portion of total network traffic. As a result, the usable coverage of signature-based IDS systems is severely limited. Expanded coverage and partial match information might be retained by an IDS if it were not for poor performance scaling.

Commonly, commercial systems must often be tuned for a particular environment to achieve acceptable performance[1]. The most common "tuning“ method is simply to remove unused signatures. The costs of performing a signature-based detection are roughly proportional to the sum of packet length and total signature length[2]. As network bandwidths have increased, so have the number and variants of threats, driving the need for expanded coverage, but remaining bounded by proportional increases in detection costs. It is the belief of the author that the high cost of modern IDS, is inextricably due to poor coverage. If performance scaling of these systems is well-understood, methods may be identified to ameliorate long-term performance degradation.

The contribution of this paper is an experimental analysis of Snort IDS scaling performance as ruleset size is varied. In addition to performance scaling, the frequency distribution of events is assessed and an assessment of IDS alerts in respect to their information content. The remainder of this paper reviews prior research in the field, defines a set of metrics and cost functions for describing performance scaling of signature-based systems, and describes the result of series of experiments used to determine the scaling performance of recent versions of the widely used Snort IDS.

2. BACKGROUND

2.1 Signatures & Coverage

Signature-based IDS are precise and due to their simplicity have been employed within many large-scale, commercially available systems. In many systems each input (e.g. header, packet, TCP session, event, event sequence, derived feature set, etc.) is compared against sets of thousands of signatures in a more-or-less brute-force manner. The patterns used and the algorithms employed for pattern matching are very efficient. Significant gains have been made in the last two decades in the area of pattern matching, leading to substantial performance improvements[3, 4, 5, 6, 7].

Commonly, signatures are written to be very precise with respect to vulnerabilities, exploits, and other “known bad" sequences. False negatives are common and false-positives are often manually “tuned" out of rulesets over time[11]. Such systems have the benefit of precision, but are generally poor at detecting new exploits and are very labor intensive to maintain in the face of large numbers of vulnerabilities and exploits. The rule-tuning process often consumes a significant portion of the time spent managing a signature-based system’s ruleset. Adaptive systems have been proposed and constructed to alleviate some of these issues[12, 13, 14]. In many instances adaptation occurs only in direct response to operator feedback in identifying false-positives or operator information overload.

There has been a number of independent studies regarding the expansion of IDS coverage. Aikelin et al. describe an approach to expanding Snort’s coverage of previously undetected variants by relaxing and varying signature parameters[10]. In their approach, rules are generalized by allowing lower-priority partial matches when most of the features of a rule are matched. By generalizing each rule in this way, expanded coverage is gained. Other approaches by Brumley et al. have focused on automated generation of vulnerability signatures[15, 16]. This has the benefit of potentially catching zero-day attacks, but appears to be limited to host-based detection systems. Network-based detection has also benefited by the advent of automated signature generation, such as the polymorphic worm signature approach by Zhang et al..[17] or the use of honeypots and attack fingerprints by Portokalidis et al[18]. As automated signature generation techniques evolve, the need for improved IDS detection performance characteristics become paramount.

2.2 Costs & Concessions

When implementing an IDS some form of cost analysis (e.g. computing, latency, hardware, training, etc.) is generally performed in order to choose the right IDS technology and ruleset for a given network. There are many trade-offs which result in sub-optimal de-

tection, but which decrease the IDS cost substantially. Many modern IDS also rely on labor-intensive tuning and rule-refinement to match particular network characteristics and known host vulnerabilities. While improving performance, this process also introduces a greater chance of false negatives. Work by Fan, et al. describe cost metrics in terms of operational costs, attacker induced damage, and incident response, citing the need to consider cost within the development and deployment of IDS[13]. Others have cleverly incorporated cost assessment into decision support systems to propose or enact response actions[19], IDS reconfiguration, and dynamic performance tuning[14].

Cost-driven optimizations allow IDS to be tractable given limited computing resources. However, they can easily be based on tenuous assumptions. Any risk analysis which performs this type of trade-off analysis needs to consider the possibility of attackers gaming a system using knowledge of financial or computational concessions. For example, the rule ordering of Snort and other signature-based IDS are partially under user control and partially under the control of engineering optimizations. This ordering matters. For example, the Snort IDS, by default only returns the first (or at best a limited number) of alerts for a packet in order to eliminate as many comparisons as possible [20]. Careful management of ruleset in order to achieve desired performance goals may also mask performance issues and can actually decrease the usefulness of IDS by removing contextual knowns from the stream of true-positives being displayed to an analyst. In a perfectly "tuned" system one might expect only the most high-priority events to be displayed and all other events to be discarded (or at least hidden). If it were at all possible to perform intrusion detection without making such concessions, we would greatly simplify the task of the IDS designer (or maintainer). The challenge is achieving these goals while also improving capabilities and performance.

2.3 Performance Measurement

IDS experimentation and test has tended to focus on three characteristics, namely: detection performance, resource usage, and resilience[21]. Detection performance uses ground truth data to determine detection rates: the true and false positives and negatives detected by a system. The resources used by a particular system can be measured as the average computing cost per packet or stream processed, where packet sizes are specified as part of the test criteria. Resilience has a broad array of meanings: performance in resource constrained environments; effects of resource contention; effects of high alarm rates during abnormal attacks; losses incurred during high network loads; resilience to artificial attacks used to mask attacker activities; and resilience to various attacks to the IDS itself[21].

It is important to note that IDS performance characteristics are generally not independent (though many experimenters have treated them as such). Of primary importance is the coupling of resource utilization and detector performance, performance degrading as contention for the CPU or other shared resources increases[22, 23]. Similar to the approach given in this paper many studies also consider the affect of ruleset size on successful packet processing rates for a given system[22].

The use of synthetic or generated attack data is also common for assessing IDS performance and resilience[24, 25]. Privacy and confidentiality concerns are the primary motivation. As such, creating and making available labeled datasets is no easy task. Synthetic approaches are not without their pitfalls, occasionally leading to erroneous conclusions. Other testing approaches (such as those used within the DARPA 1998 Evaluation) use real traffic on real networks, but may still mis-characterize IDS performance due to closed network topologies and unrealistic attack sequences[26]

I have taken a different perspective on IDS performance and have chosen to measure its scaling performance in respect to ruleset sizes well beyond rulesets which are commonly available. Within the following experiments, the size of the IDS ruleset is intended to serve as a proxy for IDS coverage. The actual coverage of an IDS is is difficult to measure given that changing environments, vulnerabilities, and network traffic each may contribute to whether alerts occur or are relevant. I am concerned with the long-term scalability of such systems as new threats emerge and new rules are added.

3. IDS SCALING PERFORMANCE

3.1 Generating Large Rule-sets

An obvious issue with testing an IDS's rulesets size scaling performance is that there don't exist that many rules in the wild. A significant deviation from traditional testing was the introduction of a randomly generated ruleset. This was necessary in order to test the performance of the system beyond the scale and scope of available signature sets. It was desirable that the generated rules were statistically similar to the existing ruleset in respect to the string and regular expression features used. For the purposes of generating a large number of random rules, a straightforward algorithm using non-parametric statistics was used to generate a ruleset of approximately 800K rules. It is important to note that the random rules do not have any semantics and are solely used to test scalability of the IDS. As such, traditional performance measures (such as rates of true positives and false negatives) are meaningless.

To generate a large number of random rules: First, an existing ruleset is split into individual features and each feature appended to a file according to its label. Each labeled file contains as many duplicates and unique features as there are duplicates and unique features in the actual ruleset, totaling approximately 27,000 unique features. Second, based on the number of rules desired, each rule in an actual Snort ruleset is permuted thousands of times by replacing each labeled feature with a sample drawn randomly from the labeled file for the feature. This was done in order to retain a similar distribution of the feature labels and total number of features that were used. As a result, random rules were generated that have the same distribution of features and feature values as rulesets that are normally used. Lastly, invalid and duplicate rules were removed by eliminating rules which did not pass Snort's rule parser.

One issue with this approach is that many invalid and duplicate rules are generated and must be removed in order for the ruleset to be used. Of 1.2M rules randomly generated, only 800K could be retained after removal of duplicates. This leads to the random rules being slightly biased towards more complex rules with a larger number of features. As this results in an overestimate of performance costs it is not an issue in respect to my research goals.

3.2 Measuring Performance Costs

Although the true cost function for performing detection using a given IDS configuration and computing system is unlikely to be known, it can be estimated using experimental measurements. In particular, we can easily define a performance metric incorporating cost functions C_{time} (total CPU time) and C_{loss} (percent packet loss) and parameterized by the signature set size $|n|$. The cost functions can be multiplicatively combined to mean CPU-time per % of packets processed, appropriately penalizing high CPU-time or high packet loss. Note that the cost function will change (particularly with respect to packet drop rates) as the line speed is decreased.

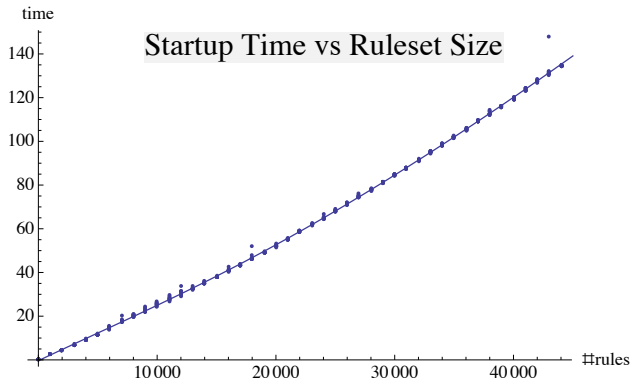


Figure 1: 5th order polynomial fit for Snort's startup time as measured in seconds. R^2: 0.9998.

$$C(n) = C_{time}(n) \cdot C_{loss}(n) = \frac{C_{time}(n)}{1 - C_{loss}(n)} \quad (1)$$

The cost functions C_{time} and C_{loss}, need to be experimentally determined. Depending on the application, the cost functions should represent conservative estimates of the actual performance costs.

My test system was an Ubuntu Linux 11.04 system running on a pair of 2GHz Dual-core AMD Operation processors, 28GB of RAM, and consumer-grade Broadcom BCM5780 Gigabit ethernet cards. The default Snort configuration (for version 2.9.0.3) was used, but with secondary detection engines disabled or suppressed. When a traditional ruleset was used, it was using the SourceFire® provided "VRT" release downloaded on Oct. 17, 2011.

My measurement approach obtains functional forms of the cost functions for a particular system and IDS configuration. Many of the performance characteristics of IDS systems are highly dependent on system and software configuration[22]. Even small changes in configuration or run-time options can have significant effects on overall performance[23]. Hardware configuration, system settings, and software configurations were kept consistent between tests.

3.2.1 Startup Performance

One of the first issues discovered in the current version of Snort is extremely slow startup times for large ruleset sizes. Startup performance is relevant as it can result in experiment run-time being dominated by startup time. As a result, experiment sizes were kept relatively small ($< 50K$ rules) in respect to the ruleset available ($\sim 800K$ rules). Startup time was also required so that measurements could be delayed until the IDS was ready for input.

An experiment was run to determine startup time which would limit experiments. This experiment made use of the Linux "time" program while running Snort against a small file containing a small sample of 10 packets. The packet processing time in this case did not have a measurable impact on the results for the sizes of rulesets tested. 10 trials were run at increments of 1,000 rules for each ruleset size between 0 and $45,000$ ($n = 450$). Figure 1 shows a best-fit polynomial function for Snort's startup time on my test system. A single test run using 100K rules required approximately 17 minutes which corresponded closely to the polynomial fit. Consequently, the current startup performance hinders experimentation with extremely large rulesets. Large rulesets might still be used, but would need to be split between multiple Snort instances to achieve reasonable startup times. It is also possible that there are configuration options which may alleviate the issue, though these are not known.

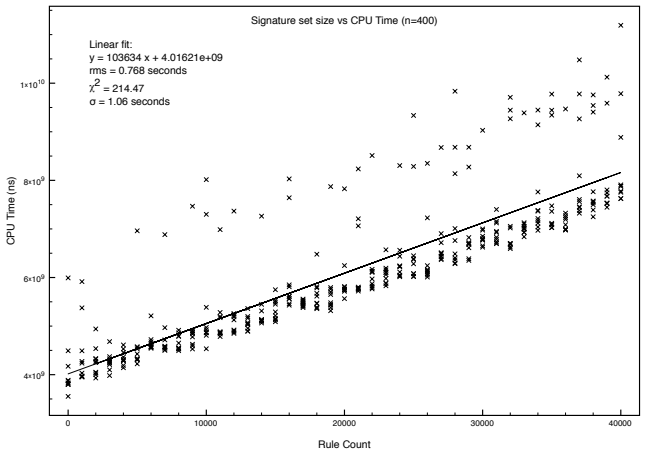


Figure 2: Linear scaling of total CPU time of the Snort IDS with ruleset size. Test system was Linux Ubuntu 11.04, running on a pair of 2GHz Dual-core AMD Opteron processors.

3.2.2 CPU Time Scaling Performance

Since we are comparing the cost of Snort instances running with various sized rulesets, we can gauge the computing cost by measuring the performance on a test platform as ruleset size is increased and finding an acceptable function to model the system's behavior as shown in Figure 2.

A linear fit resulted in a simple estimated cost function in terms of nanoseconds of CPU time:

$$C_{time}(x) = 103634x + 4.01621 \times 10^9 \quad (2)$$

The y-intercept represents the total CPU-time for the entire system when the IDS isn't doing anything. The slope of the function represents the cost in nanoseconds of CPU-time incurred due to ruleset size increases. It was expected that the scaling performance of the Snort IDS was linear in respect to ruleset size as the version of Snort being assessed uses an Aho-Corasick algorithm for string matching[2, 6]. This algorithm's complexity is linear in the sum of the length of patterns, the string being matched, and output length[28]. Snort's resource usage appears to scale linearly as shown in Figure 2 with the glaring exceptions of startup time and packet drops as described in the next section.

3.2.3 Packet Drop-rate Scaling Performance

Unfortunately, estimating performance by simply measuring total system CPU-time is eventually confounded by shared-resource issues when measuring performance of Snort running with large rulesets. Large rulesets, while incurring only a proportional increase in CPU-time, result in substantial packet loss on the front-end of the IDS. This finding is the reason that packet loss is included in the cost function for the system. Without considerations of packet loss we might only slightly overestimate performance costs for small rulesets but we would grossly underestimate performance costs for larger rulesets.

Several experiments were run to learn the packet drop rates as a function of ruleset size. The randomly generated ruleset was used to determine a worst case scaling rate. For these tests, the alerting and logging facilities of Snort were disabled. In this way Snort will not block due to I/O constraints on output alerting and logging costs. This is particularly relevant when measuring performance

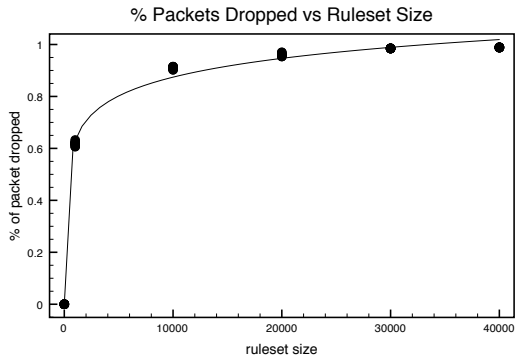


Figure 3: Packets dropped by Snort as ruleset size increases (n=60). Packets were replayed at the network card's maximum throughput ($\sim$ 400 Mbps).

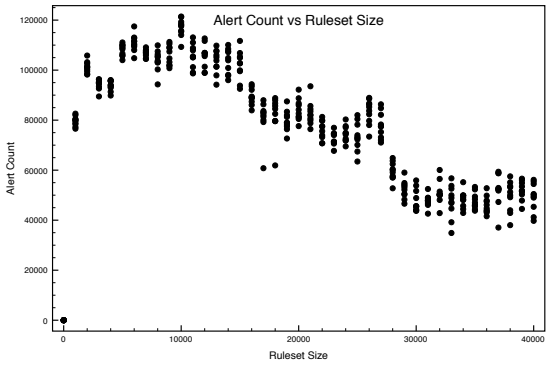


Figure 4: 10 trials at each of 1K through 40K ruleset sizes for a 1M packet sample on a live ethernet interface. Packets were replayed at the maximum interface speed ($\sim$ 400 Mbps). The non-monotonicity is due to high packet drop rates. Coverage measurements must therefore be based on cached PCAP data to eliminate drops from I/O blocking and CPU contention.

in respect to ruleset size as even small sets of randomly generated rules are likely to produce a larger number of alerts than conventional rules.

Figure 3 demonstrates that while rulesets of size n increase the system overhead proportionally, the larger complexity of the ruleset increases packet drop rates by up to a factor of 10. The cause for the high packet drop rate is likely that the Snort process is starved for CPU-time due to processor contention or conflicting I/O IRQs, possibly due to configuration issues such as a high NAPI (New API) budge rate as suggested in [23]. The unfortunate side effect is that the total number and type of alerts produced is not monotonically increasing as ruleset size increases (see Figure 4).

High packet-loss issues have been noted by others and various methods have been used to limit packet loss[23]. In our case, however, as the number of rules is increased, contention for the CPU will eventually result in the same problem. As a result, the overall shape of the packet loss function is unlikely to change significantly, though this has not been thoroughly explored.

If we measure Snort's packet processing rate using the built-in performance monitor we can estimate the performance bottleneck as an exponential function in terms of the percentage of received packets processed. On my test system, an acceptable functional

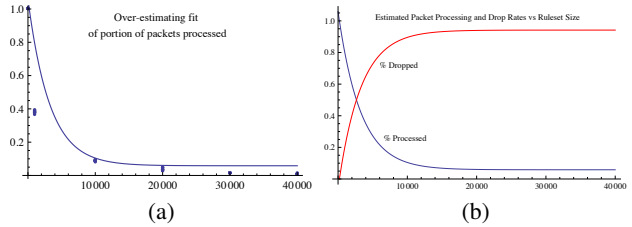


Figure 5: (a) Over-estimating fit for packet processing rate. (b) Curves estimating packet processing rate as ruleset size is increased.

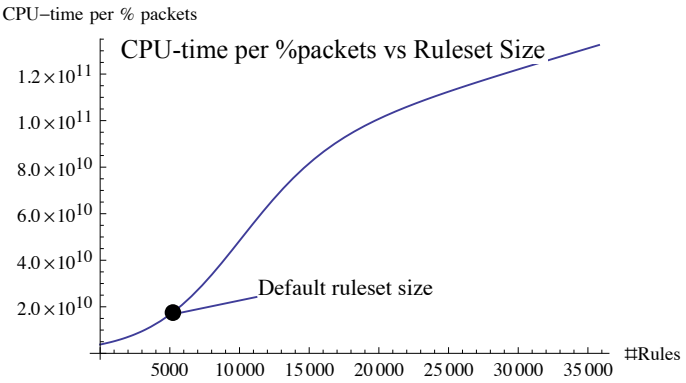


Figure 6: The cost function for the Snort IDS running on a test system. CPU Time per percent packet units are nanoseconds per %.

model was found:

$$PacketProcessingRate \leq 1.0392^{-0.008x} + 0.0583 \quad (3)$$

As can be seen from Figs. 5a and 5b, this fit is a gross over-estimation of experimental measurements, but sufficient for our purposes. An over-estimation of the number of packets successfully processed will result in an under-estimate of the effects of increasing the number of rules in a particular IDS instance. For my test system the packet-loss rate is:

$$C_{loss}(n) \geq 0.9417 - 1.0392^{-0.008n} \quad (4)$$

It is important to reiterate that other architectures, operating system kernels, and configurations will result in different scaling performance, though the basic functional shape is likely to remain similar. Figure 6 shows the scaling performance of the Snort IDS for my test system. Note that the default ruleset size is well below the knee of the performance curve.

4. DISCUSSION & FUTURE WORK

The basic measurement approaches presented here are not novel except for the explicit representation of scaling performance beyond traditional ruleset sizes. As is the case with many scaling performance studies, I have simply abused the IDS system by forcing it to perform detection outside of its design parameters. However, the end goal of all of this hand-wringing over IDS performance scaling is to show how the performance characteristics are amenable to new optimization methods[30, 31]. The promise is that much larger

rulesets might use the less computing power by trading an increase in unlikely false negatives for smaller detector costs. For prediction to play a significant role in improving IDS performance, new rulesets with superior coverage are needed along with new performance models and testing methods.

5. REFERENCES

[1] N. Stakhanova, Y. Li, and A. A. Ghorbani. Classification and Discovery of Rule Misconfigurations in Intrusion Detection and Response Devices. Proceedings of the 2009 World Congress on Privacy, Security, Trust and the Management of e-Business. 2009).

[2] V. Dimopoulos, I. Papaefstathiou, and D. Pnevmatikatos. A Memory-Efficient Reconfigurable Aho-Corasick FSM Implementation for Intrusion Detection Systems. Proceedings of the 2007 International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation. pp. 186-193. (2007).

[3] S. Kim. Pattern Matching Acceleration for Network Intrusion Detection Systems. SAMOS'05: Proceedings of the 5th international conference on Embedded Computer Systems: architectures, Modeling, and Simulation. (2005).

[4] C.-H. Lin and S.-C. Chang. Efficient pattern matching algorithm for memory architecture. IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 19, no. 1, pp. 33-41. (2011).

[5] D. Luchaup, R. Smith, C. Estan, and S. Jha. Multi-byte regular expression matching with speculation. Recent Advances in Intrusion Detection. (2009).

[6] M. Norton. Optimizing Pattern Matching for Intrusion Detection. Technical Report, SourceFire Inc (2004).

[7] N. Schear, D. R. Albrecht, and N. Borisov. High-Speed Matching of Vulnerability Signatures. LNCS, vol. 5230, no. 2008, pp. 155-174. (2008).

[8] L. Vespa, M. Mathew, and N. Weng. Predictive Pattern Matching for Scalable Network Intrusion Detection. Information and Communications Security. LNCS, vol. 5927, pp. 254-267. Springer, Heidelberg (2009).

[9] G. Tripp. A Finite-State-Machine based string matching system for Intrusion Detection on High-Speed Networks. Proceedings of EICAR. (2005).

[10] U. Aickelin, J. Twycross, and T. Hesketh-Roberts. Rule Generalisation in Intrusion Detection Systems using Snort. International Journal of Electronic Security and Digital Forensics. (2008).

[11] I. Dubrawsky and R. Saville. SAFE: IDS Deployment, Tuning, and Logging in Depth Authors. CISCO SAFE Whitepaper, pp. 1-58. (2003).

[12] Z. Yu, J. Tsai, and T. Weigert. An adaptive automatically tuning intrusion detection system. ACM Transactions on Autonomous and Adaptive Systems (TAAS), vol. 3, no. 3. (2008).

[13] W. Fan, W. Lee, S. Stolfo, and M. Miller. A multiple model cost-sensitive approach for intrusion detection. LNCS, vol. 1810, pp. 142-154. (2000).

[14] W. Lee, J. Cabrera, A. Thomas, N. Balwalli, S. Saluja, and Y. Zhang. Performance adaptation in real-time intrusion detection systems. In Proceedings of the 5th International conference on Recent advances in intrusion detection, vol. 2416, pp. 252-273. (2002).

[15] D. Brumley, J. Newsome, D. Song, Hao Wang, and S. Jha. Towards Automatic Generation of Vulnerability-Based Signatures. 2006 IEEE Symposium on Security and Privacy, pp. 2-16. (2006).

[16] D. Brumley, J. Newsome, and D. Song. Theory and techniques for automatic generation of vulnerability-based signatures. IEEE Transactions on Dependable and Secure Computing, vol. 5, no. 4. (2008).

[17] J. Zhang, H. Duan, L. Wang, Y. Guan, and J. Wu. 2008 International Conference on Computer and Electrical Engineering. in 2008 International Conference on Computer and Electrical Engineering (ICCEE), 2008, pp. 8-13. (2008).

[18] G. Portokalidis, A. Slowinska, H. Bos, G. Portokalidis, A. Slowinska, and H. Bos. Argos: an emulator for fingerprinting zero-day attacks for advertised honeypots with automatic signature generation. ACM SIGOPS Operating Systems Review, vol. 40, no. 4, pp. 15-27. (2006).

[19] C. Strasburg, N. Stakhanova, S. Basu, and J. Wong. Intrusion response cost assessment methodology. Proceedings of the 4th International Symposium on Information, Computer, and Communications Security. (2009).

[20] M. Roesch and C. Green. SNORT Users Manual 2.8.6. pp. 1-191. (2010).

[21] N. J. Puketza, K. Zhang, M. Chung, B. Mukherjee, and R. A. Olsson. A methodology for testing intrusion detection systems. IEEE Transactions on Software Engineering, vol. 22, no. 10, pp. 719-729. (1996).

[22] L. Schaelicke, T. Slabach, B. Moore, and C. Freeland. LNCS, vol. 2820, no. 9. pp. 155-172. Springer, Heidelberg (2003).

[23] K. Salah and A. Kahtani. Improving Snort Performance Under Linux. Communications, IET, vol. 3, no. 12, pp. 1883-1895. (2009).

[24] J. Haines, R. Lippmann, D. Fried, and M. Zissman. 1999 DARPA intrusion detection evaluation: Design and procedures. Technical Report ESC-TR-99-061, MIT (2001).

[25] D. Mutz, G. Vigna, and R. Kemmerer. An Experience Developing an IDS Stimulator for the Black-Box Testing of Network Intrusion Detection Systems. Proceedings of the 19th Annual Computer Security Applications Conference. pp. 374-383. (2003).

[26] M. V. Mahoney and P. K. Chan. An analysis of the 1999 DARPA/Lincoln Laboratory evaluation data for network anomaly detection. Recent Advances in Intrusion Detection. (2003).

[27] J. Novak and S. Sturges. Target-Based TCP Stream Reassembly. Technical Report, SourceFire Inc (2007).

[28] A. V. Aho and M. J. Corasick. Efficient string matching: an aid to bibliographic search. Communications of the ACM, vol. 18, no. 6, pp. 333-340. (1975).

[29] S. Sen. Performance Characterization & Improvement of Snort as an IDS. Technical Report, Princeton University (2007).

[30] S. Fugate. Using Prediction to Improve the Performance of Network Intrusion Detection. Proceedings of the 2011 UNM Computer Science Student Conference. pp. 65-69. (2011).

[31] S. Fugate. Go for broke: Speculatively Bootstrapping Better IDS Performance. Proceedings of Research in Attacks, Intrusion, and Defenses. (2012). –Submitted

Oriented and Degree-generated Block Models: Generating and Inferring Communities with Inhomogeneous Degree Distributions

Yaojia Zhu[1⋆], Xiaoran Yan[1], and Cristopher Moore[1,2]

[1] University of New Mexico
Albuquerque NM 87131 USA
[2] Santa Fe Institute
1399 Hyde Park Road, Santa Fe NM 87501, USA

Abstract. The stochastic block model is a powerful tool for inferring community structure from network topology. However, it predicts a Poisson degree distribution within each community, while most real-world networks have a heavy-tailed degree distribution. The degree-corrected block model can accommodate arbitrary degree distributions within communities. However, since it takes the vertex degrees as parameters rather than generating them, it cannot use them to help it classify the vertices, and its natural generalization to directed graphs cannot even use the orientation of the edges. In this paper, we present variants of the block model with the best of both worlds: they can use vertex degrees and edge orientations in the classification process, while tolerating heavy-tailed degree distributions within communities. We show that for some networks, including synthetic networks and networks of word adjacencies, these new block models achieve a higher accuracy than the standard or degree-corrected block models.

Keywords: We would like to encourage you to list your keywords within the abstract section

1 Introduction

In many real world networks, vertices can be divided into *communities* or *modules* based on their connecting patterns. Social communities can be forged by interactions in daily activities like karate training [22]. The blogosphere contains groups of linked blogs with similar political views [1]. English words can be tagged as different parts of speech based on their adjacencies in large texts [15]. Understanding these taxonomic structures is crucial in deciphering these topology data. There has been a great deal of work on efficient algorithms for community detection in networks (see [10, 19] for reviews).

The *Stochastic block model* (SBM) [9, 11, 20, 3] is a popular model-based approach for *functional* community detection. It partitions the vertices into communities or *blocks*, where vertices belonging to the same block are *stochastically*

⋆ This work was supported by the McDonnell Foundation.

equivalent [21] in the sense that the probabilities of a connection with all other vertices are the same for all vertices in the same block. With this general definition of *functional* communities, block models can capture various community structures, including assortative, disassortative, satellite communities and mixtures of them [16, 17, 14, 13, 8, 7].

Given the block memberships, the SBM assumes that each edge is generated independently, and follows a Bernoulli distribution solely determined by block memberships of its endpoints. Since edges in the SBM are independent, and since every pair of vertices in a given pair of blocks have a link with the same probability, for large n the degree distribution within each block is Poisson. As a consequence, the SBM dictates that vertices with very different degrees are unlikely to be in the same block. This leads to problems when modeling the networks like the political blogs, since within each community, e.g. liberal or conservative, there are both highly popular and isolated vertices at the same time.

Recently, Karrer and Newman [12] developed the *degree-corrected block model* for undirected networks (DC). They add a parameter for each vertex, which controls its expected degree. By setting these parameters equal to the observed degrees, the DC can accommodate arbitrary degree distributions within communities. This removes the model's tendency to separate high-degree and low-degree vertices into different communities.

On the other hand, the degree-corrected model cannot use the vertex degrees to help it classify the vertices, precisely because it takes the degrees as parameters rather than as data that needs to be explained. For this reason, DC may actually fail to recognize communities that differ significantly in their degree distributions. Thus we have two extremes: the SBM separates vertices by degree even when it shouldn't, and DC fails to do so even when it should.

For directed graphs, the natural generalization of DC, which we call *directed degree-corrected block model* (DDC), has two parameters for each vertex: the expected in-degree and out-degree. But this model cannot even take advantage of edge orientations. For instance, in English adjectives usually precede nouns but rarely the other way around. The ratio of each vertex's in- and out-degree could be very indicative for its block membership, and leveraging this part of the degree information would be essential.

In this paper, we first propose the *oriented degree-corrected block model* (ODC), which combines the strengths of the degree-corrected and uncorrected block models. ODC is able to utilize the edge orientations for community detection by only correcting the total degrees instead of the in- and out-degrees separately. We show that for networks with strongly asymmetric behavior between communities, including synthetic networks and networks of word adjacencies in English text, ODC achieves a higher accuracy than either the original stochastic block model or the degree-corrected block model.

We then propose the *degree-generated block model* (DGBM), which treats the expected degree of each vertex as generated from prior distributions in each community, such as power laws with different exponent and cutoffs in each com-

munity. We then include the probability of these degrees in the likelihood of a given block assignment. In this way, the model captures the dependence of the degree distributions on the community structure. These degree-generated block models automatically strike a balance between allowing vertices of different degrees to coexist in the same community on the one hand, and using vertex degrees to separate vertices into right communities on the other. DGBM works especially well for detecting community structures where communities have highly inhomogeneous degree distributions. These degree distributions differ enough between communities so that we should use vertex degrees to help us classify the vertices.

Empirical study show that DGBM are indeed a very robust choice especially when the connecting pattern alone is not enough to detect the community structure. DGBM has a further advantage in faster convergence as it reshapes the landscape of the parameter space, providing the searching algorithm a short cut to the desired community structure.

These new variants of the stochastic block model give us the best of both worlds: they can tolerate heavy-tailed degree distributions within communities, but can also use degrees and edge orientations to help classify the vertices. In addition to their performance on these networks, our models illustrate a valuable point about generative models and statistical inference: when inferring the structure of a network, you can only use the information that you try to generate.

2 The models

In this section, we review the degree-corrected block model of [12], and present our variations on it, namely oriented and degree-generated block models.

2.1 Background: degree-corrected block models

Throughout, we use N and M to denote the number of vertices and edges, and K to denote the number of blocks. The problem of determining the number of blocks is a subtle model selection problem, which we do not address here.

In the original stochastic block model, the entries A_{uv} of the adjacency matrix are independent and Bernoulli-distributed, with $P(A_{uv} = 1) = p_{g_u,g_v}$. Here g_u is the block to which u belongs, where p is a $K \times K$ matrix. Karrer and Newman [12] consider random multigraphs where the A_{uv} are independent and Poisson-distributed,

$$A_{uv} \sim \mathrm{Poi}(\theta_u \theta_v \omega_{g_u,g_v}) \,.$$

Here ω replaces p, and θ_u is an overall propensity for u to connect to other vertices. Note that since the A_{uv} are independent, the degrees d_u will vary somewhat around their expectations; however, the resulting model is much simpler to analyze than one that controls the degree of each vertex exactly.

The likelihood with which this model generates a graph G is then

$$P(G \mid \theta, \omega, g) = \prod_{u,v} \frac{(\theta_u \theta_v \omega_{g_u g_v})^{A_{uv}}}{A_{uv}!} \exp\left(-\theta_u \theta_v \omega_{g_u g_v}\right) \,. \tag{1}$$

To remove the obvious symmetry where we multiply the θ's by a constant C and divide ω by C^2, we can impose a normalization constraint $\sum_{u:g_u=r} \theta_u = 1$ for each block r. Under these constraints, the maximum likelihood estimates (MLEs) for the θ parameters are $\hat{\theta}_u = d_u/\kappa_{g_u}$ where $\kappa_r = \sum_{u:g_u=r} d_u$ is the total degree of the vertices in block r. For each pair of blocks r, s, the MLE for ω_{rs} is then m_{rs}, the number of edges connecting block r to block s (where edges within blocks are counted twice). Substituting these MLEs for θ and ω then gives the log-likelihood

$$\log P(G \mid g) = \frac{1}{2} \sum_{r,s=1}^{K} m_{rs} \log \frac{m_{rs}}{\kappa_r \kappa_s} \,. \tag{2}$$

2.2 Directed and oriented degree-corrected models

The natural extension of the degree-corrected model to directed networks, which we call the directed degree-corrected block model (DDC), has two parameters $\theta_u^{\text{in}}, \theta_u^{\text{out}}$ for each vertex. The number of directed edges from u to v is again Poisson-distributed,

$$A_{uv} \sim \text{Poi}(\theta_u^{\text{out}} \theta_v^{\text{in}} \omega_{g_u, g_v}) \,.$$

With the constraints $\sum_{u:g_u} \theta_u^{\text{in}} = \sum_{u:g_u} \theta_u^{\text{out}} = 1$ for each block r, the MLEs for these parameters (see the online version) are

$$\hat{\theta}_u^{\text{out}} = \frac{d_u^{\text{out}}}{\kappa_{g_u}^{\text{out}}} \,, \quad \hat{\theta}_u^{\text{in}} = \frac{d_u^{\text{in}}}{\kappa_{g_u}^{\text{in}}} \,, \quad \hat{\omega}_{rs} = m_{rs} \,, \tag{3}$$

where $\kappa_r^{\text{in}}, \kappa_r^{\text{out}}$ denote the total in- and out-degrees in block r and m_{rs} is the number of directed edges from block r to block s. Substituting these MLEs gives the log-likelihood

$$\log P(G \mid g) = \sum_{r,s=1}^{K} m_{rs} \log \frac{m_{rs}}{\kappa_r^{\text{out}} \kappa_s^{\text{in}}} \,. \tag{4}$$

In the DDC, the in- and out-degrees of each vertex are completely specified by the θ parameters, at least in expectation. Thus the DDC lets vertices with arbitrary in- and out-degrees to fit comfortably together in the same block. On the other hand, since the degrees are given as parameters, rather than as data that the model must generate and explain, the DDC cannot use them to infer community structure. Indeed, it cannot even take advantage of the orientations of the edges, and as we will see below it performs quite poorly on networks with strongly asymmetric community structure.

To deal with this, we present a partially degree-corrected block model capable of taking advantage of edge orientations, which we call the *oriented degree-corrected block mode* (ODC). Following the maxim that we can only use the information that we try to generate, we separate the generation of the edge orientations from the degrees.

Let $\bar{G}$ denote the undirected version of a directed graph G, i.e., the multigraph resulting from erasing the arrows for each edge. Its adjacency matrix is $\bar{A}_{uv} = A_{uv} + A_{vu}$, so (for instance) $\bar{G}$ has two edges between u and v if G had one pointing in each direction. The ODC can be thought of as generating $\bar{G}$ according to the undirected degree-corrected model, and then choosing the orientation of each edge according to another matrix ρ_{rs}, where an edge (u, v) is oriented from u to v with probability ρ_{g_u, g_v}. Thus the total log-likelihood function for such a model is

$$\log P(G \mid g, \rho) = \log P(\bar{G} \mid g) + \log P(G \mid \bar{G}, g, \rho)\,. \tag{5}$$

Writing $\bar{m}_{rs} = m_{rs} + m_{sr}$ and $\bar{\kappa}_r = \kappa_r^{\text{in}} + \kappa_r^{\text{out}}$, we can set θ_u and ω_{rs} for the undirected model to their MLEs as in Section 2.1, giving

$$\log P(G \mid g) = \frac{1}{2} \sum_{r,s=1}^{K} \bar{m}_{rs} \log \frac{\bar{m}_{rs}}{\bar{\kappa}_r \bar{\kappa}_s}\,. \tag{6}$$

The orientation term is

$$\log P(G \mid G, g, \rho) = \sum_{rs} m_{rs} \log \rho_{rs} = \frac{1}{2} \sum_{rs} (m_{rs} \log \rho_{rs} + m_{sr} \log \rho_{sr})\,, \tag{7}$$

For each r, s we have $\rho_{rs} + \rho_{sr} = 1$, and the MLEs for ρ are

$$\hat{\rho}_{rs} = m_{rs} / \bar{m}_{rs}\,. \tag{8}$$

As (7) is maximized when $\hat{\rho}_{rs}$ are near 0 or 1 for $r \neq s$, the edge orientation term prefers highly asymmetric inter-block connections. Since $\hat{\rho}_{rr} = 1/2$ for any block r, it also prefers disassortative mixing, with as few connections as possible within blocks.

Substituting the MLEs for ρ and combining (6) with (7), the total log-likelihood is

$$\log P(G \mid g) = \sum_{r,s=1}^{K} m_{rs} \log \frac{m_{rs}}{\kappa_r \kappa_s}\,. \tag{9}$$

As we show in the online version, we can also view the ODC as a special case of the DDC, where we add the constrain $\theta_u^{\text{in}} = \theta_u^{\text{out}}$ for all u. Moreover, if we set $\theta_u = 1$ for all u, we obtain the original block model, or rather its Poisson multigraph version where each A_{uv} is Poisson-distributed with mean ω_{g_u, g_v}. Thus

$$\text{SBM} \leq \text{ODC} \leq \text{DDC}\,,$$

where $A \leq B$ means that model A is a special case of model B, or that B is an elaboration of A.

2.3 Degree-generated block model

Another way to utilize the degree information for community detection is through the degree distributions in each community. In all block models, the degree of each vertex is asymptotically Poisson-distributed. But in the SBM, all the vertices in each community have the same expected degree, while DC or DDC fully specify each vertex's expected degree using θ.

A natural way to make explicit use of the degree information is to force the model to generate the vertex degrees in each community according to some distribution whose parameters differ from one community to another. To maintain the tractability of the random multigraph, we generate the parameters θ, and thus the expected degree of each vertex, rather than the degrees themselves. Given a vertex u, we first generate its expected degree θ_u from a prior degree distribution according to its block membership, then use it as the θ parameter in the degree-corrected block model to generate edges. The *degree-generated block model* is thus a *hierarchical model*, which extends previous degree-corrected block models by adding a degree generation (DG) stage on top.

Likelihood functions and parameter estimation
Given the number of vertices N and number of blocks K, the generative process for undirected networks is described in Table. 1.

Table 1. Degree-generated block model (undirected)

For each vertex $u = 1, ..., N$
Generate $\theta_u \mid g_u, \psi_{1..K}, \sim \mathcal{F}_{g_u}(\cdot \mid \psi_{g_u})$
For each pair of vertices (u, v)
Generate $A_{uv} \sim \mathrm{Poi}(\theta_u \theta_v \omega_{g_u g_v})$

In the table, ψ_r are the hyper parameters of the degree distributions for block r. The form (cdf) of the degree distributions are denoted as $\mathcal{F}_r(\cdot|\psi_r)$, which are given using domain knowledge. If such prior knowledge does not exist, we shall pick distributions which best fit the observed degree sequence. Note that the second stage is just the DC model (1), and we call the whole model DG-DC.

Table 2. Degree-generated block model (directed)

For each vertex $u = 1, ..., N$
Generate $\theta_u^{\mathrm{in}} \mid g_u, \psi_{1..K}^{\mathrm{in}}, \sim \mathcal{F}_{g_u}^{\mathrm{in}}(\cdot \mid \psi_{g_u}^{\mathrm{in}})$
Generate $\theta_u^{\mathrm{out}} \mid g_u, \psi_{1..K}^{\mathrm{out}}, \sim \mathcal{F}_{g_u}^{\mathrm{out}}(\cdot \mid \psi_{g_u}^{\mathrm{out}})$
For each pair of vertices (u, v)
Generate $A_{uv} \sim \mathrm{Poi}(\theta_{uv} \omega_{g_u g_v})$
Generate $A_{vu} \sim \mathrm{Poi}(\theta_{vu} \omega_{g_v g_u})$

For directed graphs, we have ψ_r^{in} and ψ_r^{out} as the hyper parameters of the in- and out-degree distributions for block r respectively. Their degree distributions are denoted as $\mathcal{F}_{g_u}^{\text{in}}(\cdot|\psi_{g_u}^{\text{in}})$ and $\mathcal{F}_{g_u}^{\text{out}}(\cdot|\psi_{g_u}^{\text{out}})$. Both of them can be specified by domain knowledge. For directed graphs, DG stage can precede either DDC or ODC. This flexibility comes from the following options available for the degree-correction term θ_{uv}, which we should name accordingly:

$$\theta_{uv} = \begin{cases} \theta_u^{\text{out}}\theta_v^{\text{in}} & \text{DG-DDC} \\ \theta_u\theta_v & \text{DG-ODC} \end{cases}.$$

Here $\theta_u = \theta_u^{\text{out}} + \theta_u^{\text{in}}$ is the total degree of vertex u. We also have the option to generate total degrees instead, then generate directed networks with orientation parameters ρ as we did in (7). But for the rest of the paper, we use the above directed version for DG-ODC.

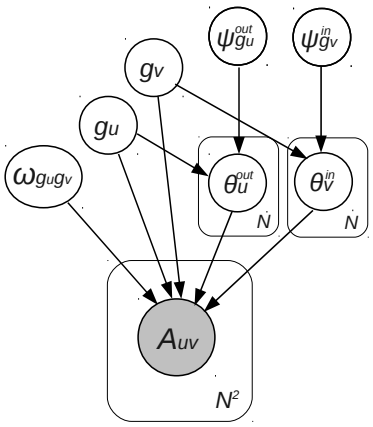


Fig. 1. Graphical model of DG-DDC

Under DG, θ parameters θ are now generated by hyper parameters ψ, as demonstrated by the graphical model in Fig. 1. The log-likelihood function is thus composed of two terms, for DG-DC:

$$\log P(G \mid g, \theta, \omega, \psi) = \log P(\theta \mid g, \psi) + \log P(G \mid \theta, g, \omega)\,. \tag{10}$$

where the first terms is for degree-generation stage, with the second term for edge-generation stage. Similarly, we have DG-DDC:

$$\log P(\boldsymbol{G} \mid g, \theta^{\text{out}}, \theta^{\text{in}}\omega, \psi) = \log P(\theta^{\text{out}}, \theta^{\text{in}} \mid g, \psi) + \log P(\boldsymbol{G} \mid \theta^{\text{out}}, \theta^{\text{in}}, g, \omega)\,. \tag{11}$$

And DG-ODC:

$$\log P(\boldsymbol{G} \mid g, \theta, \omega, \psi) = \log P(\theta^{\text{out}}, \theta^{\text{in}} \mid g, \psi) + \log P(\boldsymbol{G} \mid \theta, g, \omega)\,. \tag{12}$$

The inference of all 3 DG models are very similar. For simplicity, we shall focus on the parameter estimation of DG-DC. Readers can easily generalize the result to DG-DDC and DG-ODC.

In hierarchical models like DG-DC, estimating MLEs for both θ and ψ is usually difficult, as their joint distribution is often intractable. Here we shall propose an simple approximation that is intuitive and efficient.

We first estimate θ to maximize only the second term in (10). Notice that if we only consider the edge-generation term, we have the original DC model, and the MLEs for θ are the observed degrees. Substituting θ with these degrees, we can then estimate the hyper-parameters ψ to maximize the first term. As a result, we simply reuse the degree-corrected block models and the degree-generation term becomes the log-likelihood of generating the observed degrees.

As all the θ parameters are generated independently, for undirected networks the degree generation term is

$$\log P(\theta \mid g, \psi) = \sum_u \log \left(\mathcal{F}_{g_u}(d_u|\psi_{g_u})\right) . \tag{13}$$

Notice that if we use a uninformative piror, i.e., assuming $P(\theta \mid \psi)$ is a uniform distribution, this approximation is exact. With any other prior, the MLEs would be different. Under this approximation, the degree generation term acts merely as a penalty term rather than an intergral part of the likelihood function. However, it is enough to achieve our goal of leveraging the degree information for community detection.

Degree independent community structures favored by the DC model are no longer necessarily the most likely of the whole likelihood function. If they are too far off the prior degree distribution, they will be properly penalized for their poor fit. This would leave the door open for other community structures that might not be as a good fit to the edges, but compensates with a much better fit to the degrees.

Next, we shall show one of the most popular form for $\mathcal{F}_{g_u}(\cdot|\psi_{g_u})$, and the corresponding estimation of the hyper parameter ψ under the approximation $\hat{\theta_u} = d_u$.

Power-law degree generation

We present a power-law DG here to illustrate how we can use DG to handle degree sequences with power-law tail. Degree-generated block model for networks with other degree distributions can be established in the same way. We focus on power-law in this paper because it is prevalent in all kinds of real world networks. First popularized by the Preferential attachment model [4], power-law degree distributions has been hot topic in networks across different disciplines [10, 19, 2, 5].

With a power-law distribution, the degrees are highly skewed and degree-correction becomes necessary. Thus, power-law tail is an idea test bed for our degree-generated block model where degree-correction is crucial but it alone cannot achieve satisfying performance.

Although the degrees are discrete values, the θ parameters can be continuous. We fit the observed degrees to both discrete and continuous power-law distri-

butions. Empirically we find both of them perform very well. The only notable difference between them is the running time. Fitting degrees to a continuous power-law distribution can be faster because the MLEs of its parameters can be calculated analytically.

In the discrete case, for any vertex u in block r, the probability that it has degree d_u (here d_u can be in-, out- or total-degree of vertex u) is the following

$$p(d_u) = \begin{cases} \beta_r & d_u < d_{\min} \\ \frac{(1-\beta_r) d_u^{-\alpha_r}}{\zeta(\alpha_r, d_{\min})} & d_u \geq d_{\min} \end{cases} \tag{14}$$

Here we have the Riemann zeta function $\zeta(\alpha, x) = \sum_{n=x}^{\infty} n^{-\alpha}$. And $d_{\min}$ is the minimal degree of the power-law tail. The hyper parameters of the power-law degree distribution are $\psi_r = \{\alpha_r, \beta_r\}$.

The MLEs for β is straightforward, that is

$$\beta_r = \phi_r / n_r \,. \tag{15}$$

Here ϕ_r is the number of vertices in block r that have degree less than $d_{\min}$. The MLEs for α is a little more complicated. A detailed description about the MLEs for power-law distribution can be found in [6]. There is no analytical solution, but α can be estimated numerically. Given a power-law degree sequence in any block r, that is $d = \{d_1, d_2, ..., d_n\}$, here $d_i \geq d_{\min}$ are integers larger than or equal to $d_{\min}$, the log-likelihood function for the power-law parameter α is

$$\log P(\alpha) = -n \ln \zeta(\alpha, d_{\min}) - \alpha \sum_{i=1}^{n} \ln d_i \,. \tag{16}$$

Then, we can simply search α to maximize (16).

In the continuous case, for any vertex u, we have

$$p(d_u) = \begin{cases} \beta_r & d_u < d_{\min} \\ \frac{(1-\beta_r)(\alpha_r - 1)}{d_{\min}} \left(\frac{d_u}{d_{\min}} \right)^{-\alpha_r} & d_u \geq d_{\min} \end{cases} \tag{17}$$

The MLEs for β is same as (15). Now the MLEs for α can be solved analytically. Given a power-law degree sequence $d = \{d_1, d_2, ..., d_n\}$, $d_i \geq d_{\min}$ in any block r, the MLEs for α_r is

$$\hat{\alpha}_r = 1 + n \left[\sum_{i=1}^{n} \ln \frac{d_i}{d_{\min}} \right]^{-1} \tag{18}$$

3 Experiments

3.1 Generation of synthetic networks

Undirected networks with specific degree distributions in each community can be generated by DG-DC. For each vertex u, we first generate θ_u following the

distribution $\mathcal{F}_{g_u}(\cdot|\psi_{g_u})$. After generating θ parameters, we need to choose a symmetric ω matrix, which satisfies $\sum_s \omega_{rs} = \kappa_r$ for each block r. Here κ_r is the summation of the θ values in block r, that is $\kappa_r = \sum_{u:g_u=r} \theta_u$. Then the number of edges connecting each pair of vertices u, v is drawn from a Poisson distribution with mean $\theta_u\theta_v\omega_{g_ug_v}/(\kappa_{g_u}\kappa_{g_v})$ or $\theta_u\theta_v\omega_{g_ug_v}/(2\kappa_{g_u}\kappa_{g_v})$ if $u = v$. We can also generate the edges by first determining the number of edges for each air of blocks and then assigning the edge ends to vertices, which is described in [12].

Directed networks can be generated by DG-DDC or DG-ODC in the way illustrated in Table 2. For each vertex u, we first generate θ_u^{in} and θ_u^{out}, each of them follows a specific distribution: $\mathcal{F}_{g_u}^{\text{in}}(\cdot|\psi_{g_u}^{\text{in}})$ and $\mathcal{F}_{g_u}^{\text{out}}(\cdot|\psi_{g_u}^{\text{out}})$ respectively.

For DG-DDC, given a block assignment g, we have $\kappa_r^{\text{out}} = \sum_{u:g_u=r} \theta_u^{\text{out}}$ and $\kappa_r^{\text{in}} = \sum_{u:g_u=r} \theta_u^{\text{in}}$. Now ω_{rs} can be chosen to specify the community structure, and it should satisfy the constraints $\kappa_r^{\text{out}} = \sum_s \omega_{rs}$ and $\kappa_r^{\text{in}} = \sum_s \omega_{sr}$. After choosing ω_{rs}, the number of directed edges from vertex u to v is Poisson distributed with mean equal to $\theta_u^{\text{out}}\theta_v^{\text{in}}\omega_{g_ug_v}/(\kappa_{g_u}^{\text{out}}\kappa_{g_v}^{\text{in}})$.

As to DG-ODC, for each vertex u, we combine the parameters θ_u^{in} and θ_u^{out} into one by setting $\theta_u = \theta_u^{\text{out}} + \theta_u^{\text{in}}$. Given a block assignment g, we have $\kappa_r = \sum_{u:g_u=r} \theta_u$. Now ω_{rs} can be chosen to specify the community structure, and it should satisfy the constraints $\kappa_r = \sum_s(\omega_{rs} + \omega_{sr})$. After choosing ω_{rs}, the edges can be generated. The number of directed edges from vertex u to v is Poisson distributed with mean equal to $\theta_u\theta_v\omega_{g_ug_v}/(\kappa_{g_u}\kappa_{g_v})$.

3.2 Synthetic test on undirected networks

We generate undirected networks using DG-DC. The degree sequence in the first block follows a power-law tail with $\alpha = 2.5$ and $d_{\min} = 1$. In the second block, the degree sequence is Poisson-distributed with mean 20. We place about 1200 vertices in each block.

Just like in [12], we use a parameter λ to interpolate linearly between a fully random network with no community structure to some planted one as the following

$$\omega_{rs} = \lambda\omega_{rs}^{\text{planted}} + (1-\lambda)\omega_{rs}^{\text{random}}. \tag{19}$$

Here $\omega_{rs}^{\text{random}} = \kappa_r\kappa_s/2M$, and we set $\omega_{rs}^{\text{planted}}$ to be the following

$$\omega^{\text{planted}} = \begin{pmatrix} \kappa_1 & 0 \\ 0 & \kappa_2 \end{pmatrix}. \tag{20}$$

Thus, all edges are placed within communities.

We plotted the result in Fig. 2. Each point on the graph is based on 30 randomly generated networks. For each network, we choose the best result from 10 initials. For each initial, 1 million MCMC steps are executed. The green points are obtained from DG-DC and the red points are from the original DC without degree-generation. For each color, the square ones are from initials with true block assignment, and the circle ones are from random initials. DG-DC works very well even for very small λ values. DG-DC can classify most of the

vertices correctly even when there is no community structure for DC because the block memberships for most of the vertices can be well determined only based on the degree sequence information. True block assignment initialization cannot help DG-DC. It improves DC when λ is close to the phase transition point. We checked the likelihood values found by DC-T (DC initialized with true block assignment) and mboxDC-R (DC initialized randomly) at $\lambda = 0.4$, and found DC-R achieved higher likelihood value. That means DC-T actually got stuck into a local optima.

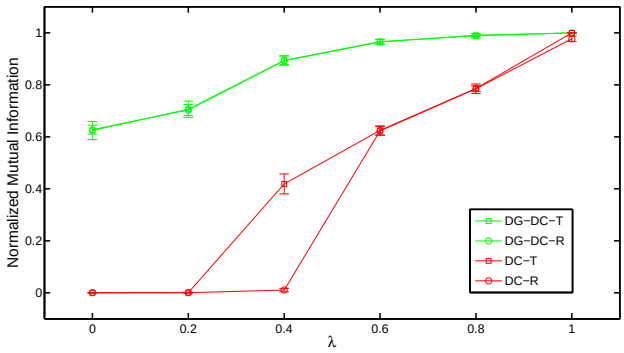


Fig. 2. Tests on networks generated by DG-DC

3.3 Synthetic test on directed networks

In our following synthetic test, we generate directed networks using DG-ODC. We place the vertices into two blocks and each block contains about 1200 vertices. In the first block, both out- and in-degrees are Poisson distributed with mean 20. In the second block, both out- and in-degrees are power-law distributed with $\alpha = 2.5$ and $d_{\min} = 1$. For ODC, $\omega_{rs}^{\text{random}}$ is the one that makes all ρ_{rs} to be equal, i.e., $1/2$. In that case all edges are oriented randomly, and $\omega_{rs} = \omega_{sr}$. On the other hand, the corresponding undirected network should also be fully random with respect to DC, namely $\omega_{rs} + \omega_{sr} = \kappa_r\kappa_s/2M$. Thus, we have $\omega_{rs}^{\text{random}} = \kappa_r\kappa_s/4M$. We set $\omega_{rs}^{\text{planted}}$ to be totally asymmetric. For $K = 2$, we have

$$\omega^{\text{planted}} = \begin{pmatrix} (\kappa_1 - \omega_{12})/2 & \omega_{12} \\ 0 & (\kappa_2 - \omega_{12})/2 \end{pmatrix}, \tag{21}$$

where $\omega_{12} \leq \min(\kappa_1, \kappa_2)$. In this test, we choose $\omega_{12} = \frac{1}{2}\min(\kappa_1, \kappa_2)$.

We can see in Fig. 3, DG-ODC and DG-DDC have very similar performance all the way and both of them can achieve much better performance than the original block models. This meets our expectation very well. As now the degree sequences contain so much information about the block memberships, both

DG-ODC and DG-DDC works perfectly. DG-ODC doesn't outperform DG-DDC because the orientation information cannot help more if the degree sequence information is already used. Without degree generation, ODC outperforms DDC for large lambda values, this is because without leveraging the degree sequence information, edge orientations can still help.

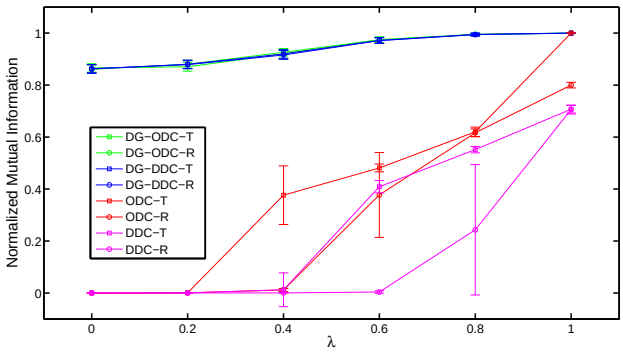


Fig. 3. Tests on networks generated by DG-ODC

In our following test, we generate directed networks using DG-DDC. We place the vertices into two blocks and each block contains about 1200 vertices. The out-degrees of block 1 and in-degrees of block 2 are Poisson-distributed with mean 20. The in-degrees of block 1 and out-degrees of block 2 are power-law distributed with $\alpha = 1.8$ and $d_{\min} = 1$. The power-law degree distributions are upper bounded to make sure the average degree is also 20 (same as the Poisson mean). For DDC, we have

$$\omega_{rs}^{\text{random}} = \kappa_r^{\text{out}} \kappa_s^{\text{in}} / M \,. \tag{22}$$

We set the planted ω matrix as the following one with only off-diagonal entries. Thus, the planted community structure is disassortative. For any λ, the connections between block 1 and block 2 are almost symmetric.

$$\omega^{\text{planted}} = \begin{pmatrix} 0 & \kappa_1^{\text{out}} \\ \kappa_2^{\text{out}} & 0 \end{pmatrix} . \tag{23}$$

As presented in Fig. 4, one interesting thing is ODC-R works equally well for all λ. Although $\omega_{rs}^{\text{random}}$ for DDC is also a random ω for ODC, for small λ, ODC can achieve very good performance due to the degree distributions we used. As both of the out-degrees in block 1 and the in-degrees in block 2 are power-law distributed, some vertices in block 1 will have very high out-degrees and some vertices in block 2 will have very high in-degrees. We can imagine a lot of edges from block 1 to block 2 are connecting these high degree vertices.

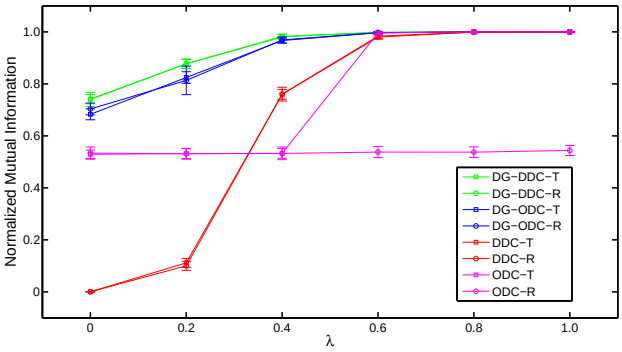


Fig. 4. Tests on networks generated by DG-DDC

If we exchange the block memberships of these vertices, the orientations of all those edges are reversed. This is a community structure preferred by ODC as now most of the inter-block edges are oriented to block 1. As the number of the high degree vertices is very small, mislabeling them does not matter much. That's why ODC can find a block assignment very close to the true one although mboxODC doesn't like the true block assignment either. When λ is large enough, ODC-T can find the planted community structure. We checked the log-likelihood values it found. For $\lambda \leq 0.8$, ODC-R actually found higher log-likelihood values than the one of the true block assignment. For $\lambda = 1$, ODC-R can only find lower log-likelihood values. That means when $\lambda = 0.6$ and 0.8, the true block assignment is actually a local optima for the ODC model. ODC-T got stuck there. But when $\lambda = 1$, namely the network is fully disassortative, then the true block assignment is a better, but ODC cannot find it within 1M MCMC steps if random initialization is adopted.

This phenomena is also found when we were trying to apply ODC on other symmetric connected networks, for example the political-blog network. Recall that the likelihood function for ODC, which is (5), has two parts. The first part generates the undirected network, which is DC. The second part generates the edge orientations. If the orientations are symmetric, there should be two modes in the landscape. One mode is the community structure preferred by the original DC, another mode is the one with highly asymmetric orientations. ODC will stuck into the second mode easily even if the network is highly assortative or disassortative and the true block assignment is the one with higher likelihood. ODC will also stuck into the first mode when true block assignment initials are used although the true block assignment may not necessary to be the one with maximum likelihood.

3.4 Empirical networks

In this section, we test the models on three word adjacency networks in which vertices are separated into two blocks: noun and adjective. "David" is the adjacency network of common adjectives and nouns in the novel David Copperfield by Charles Dickens [18]. "News" is the adjacency network of common (degree is larger than or equal to 10) adjectives and nouns obtained from News corpus. "Brown" is the giant component in the adjacency network of adjective and nouns in the Brown's corpus. All of them are asymmetric networks as adjectives are very likely to be followed by nouns, however nouns are not common to be followed by adjectives. Let's assume adjectives are in block 1 and nouns are in block 2, then we have $p_{12} > p_{21}$. For each network, we consider both multigraph (M) where multi-edges are included and simple graph (S) where multi-edges are ignored. Table. 3 revealed the basic information of these networks. The connection probability matrices are listed in Table. 4.

Table 3. Word adjacency networks

Network	#words	#adjective	#noun	#edges (S)	#edges (M)
David	112	57	55	569	1494
News	376	91	285	1389	2411
Brown	23258	6235	17023	66734	88930

Table 4. Connection probability matrices

David(S)		David(M)		News(S)		News(M)		Brown(S)		Brown(M)	
0.039	0.118	0.080	0.358	0.010	0.015	0.012	0.028	9.1e-05	3.4e-04	1.1e-04	4.4e-04
0.018	0.006	0.025	0.011	0.002	0.010	0.003	0.019	2.0e-05	8.8e-05	2.4e-05	1.2e-04

3.5 Comparison of degree-corrected models

Table. 5 compares the clustering performance for different block models, including SBM, DC, ODC and DDC. When applying DC to these directed networks, we simply ignore the edge orientations (the resulted network may contain multi-edges even though the original directed one doesn't). Both the percentage of correctly labeled vertices and the NMI value are listed for each model on each network. The results for "David" and "News" are based on 100 initials; for "Brown", 50 initials are used. All the initials are chosen randomly. For each model and each network, we take the best result among those initials. For each

initial we run the KL-heuristic [12] followed by 1 million MCMC steps. We also tested a naive heuristic (NH) algorithm which simply labels a vertex as adjective if its out-degree is larger than in-degree or noun if its out-degree is less than in-degree. For vertices with equal in- and out-degree, NH assigns its label randomly. We present the average performance of NH in Table. 5 based on 100 runs for "David" and "News" and 50 runs for "Brown". NH represents the performance we can achieve if we only use the edge orientations for community detection.

Table 5. Clustering results with random initialization

	David(S)	David(M)	News(S)	News(M)	Brown(S)	Brown(M)
SBM	84.8/0.423	57.1/0.051	62.2/0.006	57.2/0.018	70.0/0.001	70.1/7e-04
DC	91.1/0.566	91.1/0.568	56.4/0.084	56.6/0.083	55.6/0.020	55.5/0.015
ODC	87.5/0.462	87.5/0.470	56.4/0.084	56.4/0.029	75.3/0.311	80.3/0.318
DDC	70.5/0.128	51.8/8e-04	56.4/0.084	55.1/0.091	55.8/0.016	53.8/0.012
NH	84.4/0.395	86.6/0.449	72.8/0.215	73.8/0.233	78.0/0.309	78.1/0.314

a little better. ODC mislabeled 2731 while DC mislabeled 2621. But overall, ODC works much better than DDC on "News" and "Brown" networks.

For "David", DC works best and ODC also works pretty well. Both of them performs better than NH. After examining "David" more carefully, we found in this small network, three adjectives only have in-degree. They are "full", "glad" and "alone". ODC will mislabel them while DC labeled them correctly by just ignoring the edge orientations. In such a situation, we really cannot criticize ODC. As to SBM, it works well on "David(S)" but fails on "David(M)", this is because the degrees in the multigraph are more skewed than those in the simple one. It is a little surprising that DDC performs worst, and even worse than SBM. We learn a lesson here that a full degree-correction may make things worse even when the degrees in each community are quite inhomogeneous.

For "News", all the block models fail, even ODC doesn't work. We found there is a highly assortative community structure in the "News" network where about 90% edges are within commuities. All the block models will finally return a community structure very close to that one. As we mentioned earlier, ODC needs to make trade-off between the community structure preferred by DC and the one that is disassortative mixing and have highly asymmetric inter-block connections. Here, ODC sacrifices the second one. However, after checking the results returned by ODC for all initials, we found most of the times ODC works. Table. 6 presents the average clustering performances over 100 initials. The decimals outside the parentheses are the average percentages of correctly labelled vertices and the average NMI values between the best clustering the algorithm found in each

Table 6. Average clustering performances with random initialization

	David(S)	News(S)	Brown(S)
SBM	84.8(0)/0.423(0)	62.3(0.3)/0.005(0.001)	70.0(0)/0.001(0)
DC	91.1(0)/0.566(0)	56.4(0.08)/0.084(4e-04)	59.8(5.3)/0.052(0.037)
ODC	87.5(0)/0.462(0)	74.7(2.1)/0.237(0.020)	75.3(0)/0.311(0)
DDC	63.9(10.7)/0.096(0.108)	56.4(0.07)/0.084(4e-04)	53.5(2.0)/0.008(0.007)
NH	84.4(1.0)/0.395(0.023)	72.8(0.7)/0.215(0.012)	78.0(0.09)/0.309(0.003)

	David(M)	News(M)	Brown(M)
SBM	57.1(0)/0.051(0)	57.5(2.3)/0.017(0.006)	70.1(0)/7e-04(0)
DC	76.4(18.3)/0.344(0.264)	52.5(1.9)/0.025(0.030)	57.6(4.6)/0.032(0.028)
ODC	87.5(0)/0.470(0)	74.3(6.0)/0.232(0.058)	79.7(0.2)/0.317(6e-04)
DDC	59.4(7.6)/0.044(0.055)	52.4(1.9)/0.028(0.031)	52.5(1.6)/0.005(0.006)
NH	86.6(0)/0.449(0)	73.8(0.6)/0.233(0.009)	78.1(0.1)/0.314(0.002)

initial and the true clustering. The decimals in parentheses are the standard deviations. The average performance of ODC is pretty good. ODC returns that highly assortative community structure very occasionally during those initials, although that community structure has the highest likelihood value. On the other hand, if we really know something about the community structure we are looking for, finding out that one based on the results of those initials can be simple.

For "Brown", every block model fails except ODC. It seems SBM works, as it labels 70% vertices correctly. However SBM achieves this by simply putting almost all vertices in one block. That's why the NMI values are very low. ODC has very close performance to NH on both the simple and multigraph.

All the block model tests discussed so far are based on random initializations. However, NH is actually a perfect block assignment initializer for the block models on these word adjacency networks. The following tests are based on NH initializations, namely the NH results are used for block membership initializations.

Table 7. Clustering results with NH initialization

	David(S)	David(M)	News(S)	News(M)	Brown(S)	Brown(M)
SBM	84.8/0.423	57.1/0.051	62.2/0.006	56.1/0.021	70.0/0.001	70.1/7e-04
DC	91.1/0.566	91.1/0.568	56.4/0.084	55.3/0.015	70.6/0.160	70.2/0.155
ODC	87.5/0.462	87.5/0.470	75.3/0.247	77.9/0.270	75.3/0.311	80.3/0.318
DDC	57.1/0.015	64.3/0.060	56.4/0.084	52.9/0.005	54.0/0.005	64.0/0.070
NH	84.4/0.395	86.6/0.449	72.8/0.215	73.8/0.233	78.0/0.309	78.1/0.314

Table 8. Average clustering performances with NH initialization

	David(S)	News(S)	Brown(S)
SBM	84.8(0)/0.423(0)	62.2(0)/0.006(0)	70.0(0)/0.001(0)
DC	91.1(0)/0.566(0)	56.4(0.08)/0.084(4e-04)	71.1(0.2)/0.171(0.003)
ODC	87.5(0)/0.462(0)	75.3(0)/0.247(0)	75.3(0)/0.311(0)
DDC	84.2(6.1)/0.395(0.108)	56.4(0.06)/0.084(3e-04)	62.1(3.5)/0.060(0.023)
NH	84.4(1.0)/0.395(0.023)	72.8(0.7)/0.215(0.012)	78.0(0.09)/0.309(0.003)

	David(M)	News(M)	Brown(M)
SBM	57.1(0)/0.051(0)	58.7(2.0)/0.014(0.005)	70.1(0)/7e-04(0)
DC	91.1(0)/0.568(0)	52.5(1.9)/0.025(0.030)	70.2(0.3)/0.156(0.004)
ODC	87.5(0)/0.470(0)	75.3(1.2)/0.240(0.014)	79.7(0.08)/0.317(3e-04)
DDC	75.6(2.6)/0.204(0.037)	52.4(1.9)/0.028(0.031)	64.3(0.2)/0.073(0.002)
NH	86.6(0)/0.449(0)	73.8(0.6)/0.233(0.009)	78.1(0.1)/0.314(0.002)

In Table. 7, we found with NH initialization ODC works pretty well on "News". It returns the desired comunity structure in all 100 initials. DC also works much better on "Brown", and 70% vertices are correctly labelled. With random initialization, it's only around 55%. With NH initialization, the performances are more stable now. Table. 8 lists the average clustering performances and we can see the standard deviations are much lower than the ones in Table. 6. For "David", now DC works perfectly on "David(M)", but previously, it's average performance on "David(M)" is quite fair and not stable either. DDC also improves itself on both "David(S)" and "David(M)". ODC now works better on "News" with higher average percentages and NMI values and much lower deviations. DC also works much better on average while having much lower deviations on "David".

3.6 Results on degree generated models

In "Brown", the real data show that both the out- and in-degree distributions have heavy tails close to power-law. In Fig. 5 we plotted the complementary CDF of the out- and in-degrees for Brown(S) and Brown(M). We estimate the power-law θ parameters of the real data using both discrete (labeled with [D]) and continuous (labeled with [C]) methods, which are introduced in section 2.3. Setting $d_{\min} = 1$, the MLEs of the θ parameters are listed in Table 9.

We compare the performances of the power-law degree-generated block models with those original degree-corrected models. In Table 10 11, the first row lists the results of the original degree-corrected models without degree-generation, and the second row shows the results of the block models with power-law degree generation. We use KL-heuristic plus MCMC to infer the parameters and the community structures. All the results are obtained from 50 initials, each of them

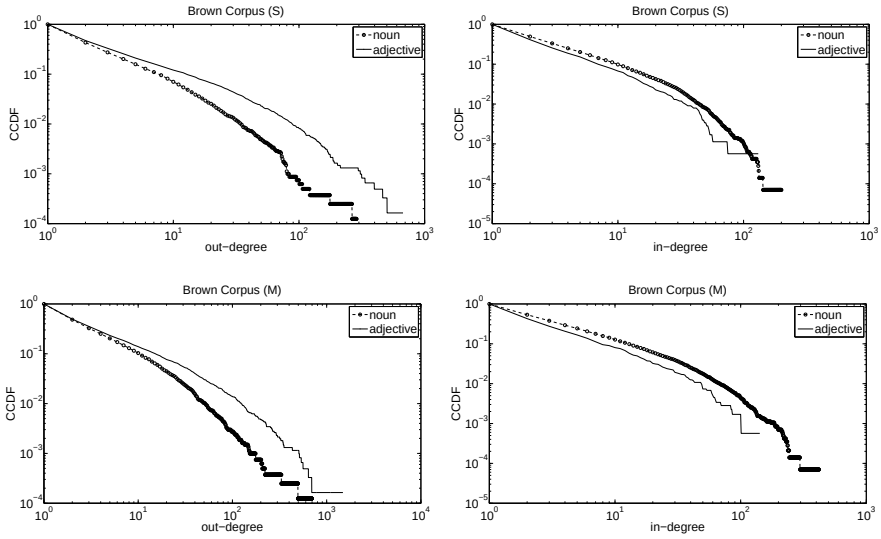


Fig. 5. Degree distributions in Brown network

Table 9. Degree parameter MLEs in Brown

	block	α_{in}[D]	α_{in}[C]	α_{out}[D]	α_{out}[C]	β_{in}	β_{out}
Brown(S)	adj	1.829	2.329	1.952	2.629	0.161	0.527
	noun	1.987	2.721	1.793	2.248	0.716	0.021
Brown(M)	adj	1.741	2.136	1.828	2.326	0.161	0.527
	noun	1.931	2.576	1.740	2.134	0.716	0.021

is initialized randomly. For each model and each network, we take the best result among those initials. In each initial we run the KL-heuristic followed by 1 million MCMC steps. Both the percentage of correctly labelled vertices and the NMI value are listed for each model on each network. We can see in Table 10, degree generation do improves the performance of DDC and DC. As to ODC, it already works pretty well by itself and degree generation doesn't help. However we also found power-law degree generation can actually speed up the searching performance. KL-heuristic has high complexity, which is $O(N^2 \log N)$. Although each MCMC step only takes $O(K^2)$, for large networks, MCMC converges slowly. If we give up KL-heuristic (this may be unavoidable for handling large networks), in other words, each initial only runs 1 million MCMC steps, we get the results in Table 11. We can see in Table 11, DDC, DC and even ODC becomes worse when there is no degree generation. With degree generation, the performances of all these models are very stable compared to the results with KL-heuristic in Table 10.

Table 10. Clustering results on Brown with KL-heuristic

	DC	ODC	DDC	DG-DC	DG-ODC	DG-DDC
Brown(S)	55.6/0.020	75.3/0.311	55.8/0.016	74.4/0.283	75.3/0.310	73.3/0.224
Brown(M)	55.5/0.015	80.3/0.318	53.8/0.012	75.6/0.290	76.0/0.320	72.2/0.210

Table 11. Clustering results on Brown without KL-heuristic

	DC	ODC	DDC	DG-DC	DG-ODC	DG-DDC
Brown(S)	54.3/0.010	72.0/0.188	53.7/0.008	75.1/0.275	76.5/0.297	71.4/0.224
Brown(M)	52.6/0.007	73.4/0.203	53.8/0.011	75.4/0.276	77.3/0.308	70.9/0.194

4 Conclusions

Degree-correction in stochastic block models provides a powerful approach to dealing with networks with inhomogeneous degree distributions. Partial degree-correction is a new idea proposed in this paper which tolerates highly skewed degrees within community while still be able to utilize degree information for community detection purpose. We demonstrated two ways to achieve this. One is the oriented degree-corrected block model which prefers highly asymmetric inter-block connections. Another is the degree-generated block model which fits the degree sequence in each community to a family of distributions before examining the edge connecting patterns. Thus, block assignments in which degree sequences are poorly fitted is thought to be unlikely.

We have shown that in certain networks these new block models can apply appropriate partial degree-correction, achieving higher accuracy. The oriented degree-corrected block model usually works well on asymmetric networks. However, when the misinformation in the edge connecting patterns is overwhelmingly strong, the orientation information might not be enough to compensate. There are also situations when the block connecting pattern looks fairly symmetric, OCD is still able to leverage highly asymmetric connections occurred between only a few vertices. Further study of different degree corrections is required to better understand their effectiveness for different networks.

The degree generated block models showed the most promise with their excellent performance across the board, especially with noisy data. Nonetheless, their effectiveness depends heavily on knowing the true form of the degree distribution in each community. Without the ground truth of block assignments, one would have to guess an appropriate form first, making it a much more difficult problem.

With multiple degree corrected models, and multiple ways to do degree generation for each of them, we now have a tough choice to make whenever we meet new networks. Better understanding of each model could help but real-world networks may exhibit structures too complicated to comprehend. A much better alternative is a model selection criterion which can predict relative performance of each model automatically based on the observed data.

5 Acknowledgments

We are grateful to Terran Lane, Ben Edwards, Aaron Clauset, and Mark Newman for helpful conversations. This work was supported by the McDonnell Foundation.

Bibliography

[1] L. Adamic and N. Glance. The political blogosphere and the 2004 US Election: Divided They Blog. In *Proc 3rd Intl Workshop on Link Discovery.*, 2005.

[2] W. Aiello, F. Chung, and L. Lu. A random graph model for power law graphs. *Experimental Mathematics*, 10(1):53–66, 2001.

[3] E.M. Airoldi, D.M. Blei, S.E. Fienberg, and E.P. Xing. Mixed membership stochastic blockmodels. *The Journal of Machine Learning Research*, 9:1981–2014, 2008.

[4] Réka Albert and Albert-László Barabási. Statistical mechanics of complex networks. *Rev. Mod. Phys.*, 74:47–97, Jan 2002.

[5] Fan Chung and Linyuan Lu. The average distances in random graphs with given expected degrees. *Proceedings of the National Academy of Sciences of the United States of America*, 99(25):15879–82, December 2002.

[6] A. Clauset, C.R. Shalizi, and M.E.J. Newman. Power-law distributions in empirical data. June 2007.

[7] Aurelien Decelle, Florent Krzakala, Cristopher Moore, and Lenka Zdeborová. Asymptotic analysis of the stochastic block model for modular networks and its algorithmic applications. *Physical Review E*, 84(6), December 2011.

[8] Aurelien Decelle, Florent Krzakala, Cristopher Moore, and Lenka Zdeborová. Inference and Phase Transitions in the Detection of Modules in Sparse Networks. *Physical Review Letters*, 107(6), August 2011.

[9] S.E. Fienberg and S. Wasserman. Categorical data analysis of single sociometric relations. *sociological Methodology*, pages 156–192, 1981.

[10] S. Fortunato. Community detection in graphs. *Physics Reports*, 2009.

[11] P.W. Holland, K.B. Laskey, and S. Leinhardt. Stochastic blockmodels: First steps. *Social Networks*, 5(2):109–137, 1983.

[12] B. Karrer and M. Newman. Stochastic blockmodels and community structure in networks. *Physical Review E*, 83(1), 2011.

[13] Cristopher Moore, Xiaoran Yan, Yaojia Zhu, Jean-Baptiste Rouquier, and Terran Lane. Active learning for node classification in assortative and disassortative networks. page 841. ACM Press, 2011.

[14] M. Mørup and L.K. Hansen. Learning latent structure in complex networks. *NIPS Workshop on Analyzing Networks and Learning with Graphs*, 2009.

[15] M. Newman and E.A. Leicht. Mixture models and exploratory analysis in networks. *Proceedings of the National Academy of Sciences*, 104(23):9564–9569, 2007.

[16] M.E. Newman. Assortative mixing in networks. *Physical Review Letters*, 89(20):208701, 2002.

[17] M.E. Newman. Mixing patterns in networks. *Physical Review E*, 67(2):026126, 2003.

[18] M.E. Newman. Finding community structure in networks using the eigenvectors of matrices. *Physical Review E*, 74(3):036104, 2006.
[19] M. A Porter, J. P Onnela, and P. J Mucha. Communities in networks. *Notices of the American Mathematical Society*, 56(9):1082–1097, 2009.
[20] T.A. Snijders and K. Nowicki. Estimation and prediction for stochastic blockmodels for graphs with latent block structure. *Journal of Classification*, 14(1):75–100, 1997.
[21] S. Wasserman and C. Anderson. Stochastic a posteriori blockmodels: Construction and assessment. *Social Networks*, 9(1):1–36, 1987.
[22] W. W Zachary. An information flow model for conflict and fission in small groups. *Journal of Anthropological Research*, 33(4):452–473, 1977.