CSAIL Student Workshop 2005

Workshop Proceedings

Preface

1st Annual CSAIL Student Workshop

CSW 2005
Gloucester, MA
September 19, 2005

The CSAIL Student Workshop (CSW) is a meeting of students, by the students, and for the students. It brings together student researchers in the disparate fields, offering a venue for interaction and the exchange of ideas. The workshop provides an opportunity for participants to gain an overview of ongoing research in CSAIL, to meet other researchers, and to initiate collaboration among different research groups. The scope of the meeting is broad, and the primary audience is students themselves. CSW continues where the Student Oxygen Workshop left off and is a forum for all the research that is occurring at CSAIL.

Many thanks to this year's sponsors, without whom the workshop would not have been possible:
ITA Software
CSAIL Student Committee
Larry Rudolph

Acknowledgements

Conference Chair:
Aaron Adler

Program Chairs:
Albert Huang
Angelina Lee
Rose Liu

Web Masters:
Jeremy Fineman
Ali Mohammad
Yuan Shen

Panel Chairs:
Harold Fox
Gary Look

Organizing Committee:
Kunal Agrawal
Mario Christoudias
Tracy Hammond
Sally Lee
Robert Wang

Steering Committee:
Larry Rudolph
Vineet Sinha

Program Committee:
Aaron Adler
Kunal Agrawal
Meg Aycinena
Jake Beal
Rob Beverly
Mario Christoudias
Brooke Cowan
Jacob Eisenstein
Harold Fox
Seth Gilbert
Albert Huang
Viktor Kuncak
Angelina Lee
George J. Lee
Ben Leong
Rose Liu
Gary Look
Gregory Marton
Sayan Mitra
Evodokia Nikolova
Mike Oltmans
Alexandru Salcianu
Metin Sezgin
Yuan Shen
Sara Su
Bill Thies
Max Van Kleek
Robert Wang
Kevin Wilson
Hanson Zhou

CSW 2005 Workshop Schedule

Gloucester, MA   September 19, 2005

8:45a:	Bus ride to Ocean View Inn and Resort, breakfast on bus	
10:25a:	**Opening remarks** (Adler, Rudolph, Wertheimer)	
10:30a:	**Technical Session I** (Chair: Liu)	
	10:30a:	Analyzing provider and user incentives under congestion pricing on the Internet (Bauer and Faratin)
	10:50a:	JCilk's exception semantics with a dynamic threading model (Lee)
	11:10a:	The Effect of Neighborhood Boundaries on Nearness (Look)
	11:30a:	Proovy: A Simple Proof Checker for Higher Order Procedures in Agent Planning (Fox)
	11:50a:	6 short talks/elevator pitches
12:02p:	**Break**	
12:10p:	**Technical Session II** (Chair: Eisenstein)	
	12:10p:	Hypernyms as Answer Types (Marton)
	12:30p:	A Scalable Mixed-Level Framework for Dynamic Analysis of C/C++ Programs, Dynamic Variable Comparability Analysis for C and C++ Programs (Guo and McCamant)
	12:42p:	Reducing Configuration Overhead with Goal-oriented Programming (Paluska)
	12:54p:	A Come-from-Behind Win or a Blown--Save Loss: Perspectives in Baseball (Oh)
1:06p:	**Lunch**, Discussion, Strolling	
2:30p:	**ITA Talk** (Greg Galperin, Justin Boyan)	
3:00p:	**Technical Session III** (Chair: Marton)	
	3:00p:	Engineering transcription-based logic (Shetty)
	3:20p:	Re-engineering Enzyme Catalysis Using Computer Modeling and Combinatorial Libraries (Armstrong)
	3:40p:	5 short talks/elevator pitches
3:50p:	**Break**	
4:00p:	**Technical Session IV** (Chair: Wilson)	
	4:00p:	The Worst Page-Replacement Policy (Fineman)
	4:20p:	Incremental Optimization of Large Robot-Acquired Maps (Olson)
	4:40p:	A Scalable Architecture for Network Fault Diagnosis in the Knowledge Plane (Lee and Faratin)
	5:00p:	A Usability Evaluation of Two Computer Vision-Based Selection Techniques (Eisenstein)
5:20p:	**Break**	
6:00p:	**Panel**: Finding a Thesis Topic, Session Chairs: Gary Look, Harold Fox Panel Members: Brooke Cowan, George Lee, Ozlem Uzuner, and Hanson Zhou	
7:00p:	**Dinner**	
8:00p:	**Talk** (Charles Leiserson)	
8:30p:	**Awards Presentation**	
9:00p:	Return to Cambridge	

Table of Contents

Full Papers

The Worst Page-Replacement Policy

Kunal Agrawal KUNAL_AG@MIT.EDU
Jeremy T. Fineman JFINEMAN@MIT.EDU
MIT Computer Science and Artificial Intelligence Laboratory, 32 Vassar Street, Cambridge, MA 02139

1. Introduction

Consider a computer system with a two-level memory hierarchy consisting of a small fast memory of size k and a large slow memory. Memory is divided into fixed-size pages. Each memory access indicates an access into a particular page of memory. If the page is located in fast memory, the access has no cost. If the page is located only in slow memory, the access induces a ***page fault***, whereby the page must be moved from slow memory into fast memory (possibly evicting a page that is currently stored in fast memory). A page fault has a cost of one.

Research in the area of page-replacement strategies focuses on strategies that reduce the number of page faults. If all the page requests are known a priori (***offline***), the optimal strategy is to replace the page whose next request occurs furthest in the future [Belady, 1966]. An ***online*** strategy must make its decisions at the time that each page request arrives, without any knowledge of the future accesses.

Since the performance of any online strategy depends on the input sequence, Sleator and Tarjan introduce ***competitive analysis*** [Sleator & Tarjan, 1985] to analyze these strategies. An online strategy A is c-competitive if there exists a constant β such that for every input sequence σ,

$$A(\sigma) \leq c \cdot \mathrm{OPT}(\sigma) + \beta \ ,$$

where $A(\sigma)$ is the cost incurred by the algorithm A on the input sequence σ, and $\mathrm{OPT}(\sigma)$ is the cost incurred by the optimal offline strategy for the sequence σ. Sleator and Tarjan prove that there is no online strategy for page replacement that is better than k-competitive, where k is the memory size. Moreover, the ***least-recently-used (LRU)*** heuristic, whereby the page evicted is always the one least recently used, is k-competitive. If the offline strategy operates on a memory that is twice the size of that used by the online strategy, LRU is 2-competitive.

In this paper we are interested in finding a "reasonable" online strategy that causes as many page faults as possible. We assume that the fast memory is initially empty. A ***reasonable*** strategy[1] follows two rules:

1. It is only allowed to evict a page from fast memory when the fast memory is full.
2. It is only allowed to evict a page from fast memory when that page is being replaced by the currently requested page, and the currently requested page does not already reside in fast memory.

Although this problem has no practical motivation, it is fun and theoretically interesting.

The optimal offline strategy OPT for the problem of maximizing page faults discards the page that will be requested next. An online strategy A is c-competitive if there exists a constant β such that for every input sequence σ,

$$A(\sigma) \geq \mathrm{OPT}(\sigma)/c - \beta \ ,$$

where $A(\sigma)$ is the number of page faults incurred by the algorithm A on the input sequence σ, and $\mathrm{OPT}(\sigma)$ is the number of page faults incurred by the optimal offline strategy on the sequence σ.

Throughout this paper, when we talk about strategies being competitive, we mean with respect to the offline strategy that maximizes page faults. An optimal strategy is therefore the "worst" page-replacement policy.

The rest of this paper is organized as follows. Section 2 proves that there is no deterministic, competitive, online algorithm to maximize page faults, and that no (randomized) algorithm is better than k-competitive. Section 3 gives an algorithm that is k-competitive, and hence optimal. Section 4 proves that a direct-mapping strategy is also the worst possible strategy under the assumption that page locations are random. Most proofs are omitted for space reasons.

2. Lower bounds

This section gives lower bounds on the competitiveness of online strategies for maximizing page faults.

The following lemma states that there is no deterministic online strategy that is competitive with the offline strategy.

[1]Once unreasonable strategies are allowed, one could design a strategy that uses only one location on the fast memory. This strategy will perform optimally for maximizing page faults, but it doesn't make much sense in a real system.

Lemma 1 *Consider any deterministic strategy A with a fast-memory size $k \geq 2$. For any $\varepsilon > 0$ and constant β, there exists an input σ such that $A(\sigma) < \varepsilon \cdot \mathrm{OPT}(\sigma) - \beta$.*

Proof. Consider a sequence σ that begins by requesting pages $v_1, v_2, \ldots, v_{k+1}$. After page v_k is requested, all strategies have a fast memory containing pages $v_1, \ldots, v_k$. At the time v_{k+1} is requested, one of the pages must be evicted from the fast memory. Suppose that the deterministic strategy A chooses to evict page v_i. Then consider the sequence $\sigma = v_1, v_2, \ldots, v_k, v_{k+1}, v_j, v_{k+1}, v_j, v_{k+1}, \ldots$, that alternates between v_{k+1} and v_j for some j with $1 \leq j \leq k$ and $i \neq j$. After v_{k+1} is requested, both v_j and v_{k+1} are in A's fast memory. Thus, A incurs only the first $k+1$ page faults. The offline strategy OPT incurs a page fault on every request (by evicting page v_j when v_{k+1} is requested and vice versa). Extending the length of the sequence proves the lemma. □

Lemma 1 also holds even if we introduce resource augmentation. That is, even if the deterministic strategy is allowed a smaller fast memory of size $k_{on} \geq 2$ than the fast memory $k_{off} \geq k_{on}$ used by the offline strategy, there is still no competitive deterministic strategy.

The following lemma states that no randomized strategy is better than expected k-competitive when both the online and offline strategies have the same fast-memory size k. Moreover, when the offline strategy uses a fast memory of size k_{off} and the online strategy has a fast memory of size $k_{on} \leq k_{off}$, no online strategy is better than $k_{off}/(k_{off} - k_{on} + 1)$. We omit the proof due to space limitations.

Lemma 2 *Let k_{off} be the fast-memory size of the offline strategy and $k_{on} \leq k_{off}$ be the fast-memory size of the online strategy. Consider any (randomized) online strategy A. For any $c < k_{off}/(k_{off} - k_{on} + 1)$ and constant β, there exists an input σ such that $E[A(\sigma)] < \mathrm{OPT}(\sigma)/c - \beta$.*

3. Most-recently used

This section describes two k-competitive strategies. The first strategy uses one step of randomization followed by the deterministic "most-recently-used" (MRU) heuristic. The second strategy uses more randomization to achieve the optimal result even when the offline and online strategies have different fast-memory sizes.

Since least-recently-used (LRU) is optimal with respect to an offline strategy that minimizes page faults, it is reasonable to expect MRU to be optimal for maximizing page faults. This strategy, however, is deterministic, and Lemma 1 states that no deterministic strategy is competitive. Instead we consider a ***randomized MRU*** strategy, where the first page evicted (when the $(k+1)$th distinct page is requested) is chosen at random. All subsequent requests follow the MRU strategy. This strategy avoids the alternating-request problem from the proof of Lemma 1.

Theorem 3 *Randomized MRU is expected k-competitive, where k is the fast-memory size.*

This result does not match the lower bound from Lemma 2. In particular, it does not generalize to the case in which the online and offline strategies have different fast-memory sizes. We have a strategy called "reservoir MRU" that uses more randomization. The main idea behind our ***reservoir MRU*** strategy is to keep a reservoir of $k_{off} - 1$ page, where each previously requested page is in the reservoir with equal probability.[2] The reservoir MRU strategy works as follows. For the first k_{on} distinct requests, the fast memory is not full, thus there are no evictions. After this time, if there is a request for a previously requested page v_i, and the page is not in fast memory, then the most recently requested page is evicted. When the nth new page is requested, for any $n > k_{on}$, the most recently requested page is evicted with probability $1 - (k_{off} - 1)/(n-1)$. Otherwise, a fast-memory location (other than the most-recently-used page's) is chosen uniformly at random, and the page at that location is evicted.

The following theorem matches the lower bound given by Lemma 2, and hence reservoir MRU is optimal.

Theorem 4 *Reservoir MRU is expected $k_{off}/(k_{off} - k_{on} + 1)$-competitive, where k_{off} is the fast-memory size of the offline strategy, and $k_{on} \leq k_{off}$ is the fast-memory size for reservoir MRU.*

This theorem means that when the offline strategy and reservoir MRU have the same fast-memory size k, reservoir MRU is k-competitive. When reservoir MRU has fast-memory size k and the offline strategy has fast-memory size $(1 + 1/c)k_{on}$, reservoir MRU is $(c+1)$-competitive, which is analogous to Sleator and Tarjan's [Sleator & Tarjan, 1985] result for LRU.

4. Direct Mapping

This section considers a particular page-replacement strategy used in real systems called "direct mapping" and proves that under some assumption of randomness, the direct-mapping strategy is k-competitive.

In a direct-mapping strategy [Patterson & Hennessy, 1998], each page is mapped to a particular location on the fast memory. If the page is requested and it is not on the fast memory, it evicts the page in that location. This strategy does not follow the rules of a reasonable strategy setup

[2]This technique is inspired by reservoir sampling [Vitter, 1985], which is where we came up with the name.

in Section 1. In particular, it may evict a page from the fast memory before the fast memory is full. We still think it is interesting to consider this strategy in the context of maximizing page faults as it is a strategy commonly implemented for the normal page-replacement problem (where the goal is reversed).

The direct-mapping strategy we consider is as follows. Each time a new page is requested, that page is mapped to a random location on the fast memory uniformly at random. Every time that page is requested again, it maps to the same location. The following theorem states that this strategy is k-competitive.

Theorem 5 *The direct-mapping strategy is k-competitive, where k is the fast-memory size of the online and offline strategies.*

In real direct-mapping strategies, pages are not typically mapped at random. Instead, they are mapped based on the lower order bits in the address. Theorem 5 states that as long as the pages are located at random locations in memory, then (deterministic) direct-mapping is the worst possible strategy. This assumption of random locations may seem a bit pessimistic in a real program. On a machine with some time sharing, however, each application may be located at a random offset in memory, so as long as the applications don't access too many pages during a time slice, the theorem still applies.

References

Belady, L. A. (1966). A study of replacement algorithms for virtual storage computers. *IBM Systems Journal*, *5*, 78–101.

Patterson, D. A., & Hennessy, J. L. (1998). *Computer organization & design: The hardware / software interface*. San Francisco, CA: Morgan Kaufmann. Second edition.

Sleator, D. D., & Tarjan, R. E. (1985). Amortized efficiency of list update and paging rules. *Communications of the ACM*, *28*, 202–208.

Vitter, J. S. (1985). Random sampling with a reservoir. *ACM Transactions on Mathematical Software*, *11*, 37–57.

Re-engineering Enzyme Catalysis Using Computer Modeling and Combinatorial Libraries

Kathryn A. Armstrong KATHRYN@MIT.EDU
Bruce Tidor TIDOR@MIT.EDU
MIT Computer Science and Artificial Intelligence Laboratory, 32 Vassar Street, Cambridge MA, 02139 USA

1. Introduction

Enzymes are proteins that act as molecular machines to efficiently catalyze chemical reactions. An enzyme is made of a chain of amino acids whose sequence is determined by the information in DNA/RNA. The enzymes existing in nature catalyze only a subset of the possible chemical reactions and so the development of new enzymes is of interest to those doing chemical synthesis. New enzymes have been created in the past by making random mutations to the protein sequence of a known enzyme and then screening for new function (Farinas et al., 2001; Glieder et al., 2002). However, the number of sequences available to an enzyme is phenomenally large, and such searches can only examine a tiny fraction of them. Efforts to computationally model and predict new catalytic sequences directly are underway but have yet to create new catalytic activity. Our strategy is more broad; we computationally analyze the sequences available to an enzyme's structure and eliminate those that are incompatible. A catalytically active enzyme must first fold correctly, and so screening this reduced list of sequences for function should be more effective than screening random mutants.

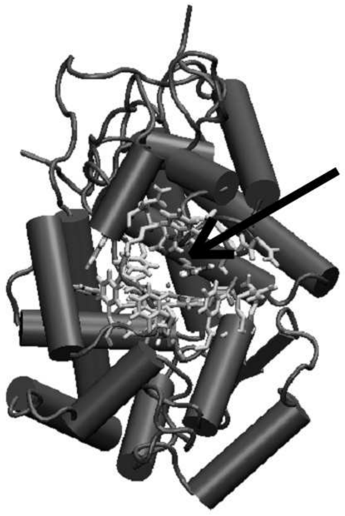


Figure 1. Horseradish peroxidase, a plant enzyme that synthesizes hormones. The enzyme active site is indicated by an arrow.

2. Reducing the Search Space

It is thought that mutations in an enzyme active site (the enzyme section most directly involved with catalysis) are most often detrimental but have a large impact on catalysis (Heering et al., 2002). Conversely, mutations far from the active site have subtle effects but are more easily tolerated (Morawski & Arnold, 2001). We would like to substantially modify catalytic activity, so we have decided to analyze the active site of the enzyme Horseradish peroxidase. The three-dimensional structure of this enzyme (produced by X-ray crystallography, shown in Figure 1) was used to start our search of the allowed active site amino acid sequences. The dead-end elimination and A* algorithms allow us to search all possible sequences of amino acids and their structures in the enzyme active site. By fixing the enzyme's backbone conformation and allowing only discrete conformations of each amino acid, dead-end elimination guarantees that we will find the global minimum energy sequence and structure in this discrete space (Desmet et al., 1992). Then A*, a branch-and-bound search algorithm, is used to create a list of sequences and structures ranked by energy (Leach & Lemon, 1998). We assume that any sequence more than 15 kcal/mol worse in energy than the native sequence will not fold into the correct enzyme structure, and so we eliminate all sequences with energies above this cutoff. The A* algorithm efficiently generates these feasible sequences by pruning the branch-and-bound tree of sequences using our energy cutoff along with estimation of the lowest possible energy required to make each sequence and structure. For the five positions we chose in the enzyme active site, this restricts the feasible list of sequences from 20^5 (3.2^6) to about 50000. This substantially reduced list of sequences can now be searched exhaustively using high-throughput experimental techniques.

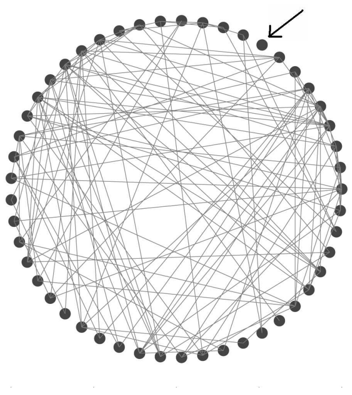


Figure 2. Evolutionary map of the feasible sequences for two active site positions. Nodes represent sequences and each edge represents a single DNA mutation. The natural sequence is indicated by an arrow.

3. Creating the Feasible Sequences

While 50000 sequences is a reasonable number of new enzymes to screen once they exist, it is beyond the limit of the number of enzymes a person would be willing to experimentally build one by one. Therefore, we have developed two techniques to create all of the feasible enzymes at once. The first method uses the list of 50000 active site sequences to create a larger combinatorial library of sequences that includes most of the sequences we desire. The most popular amino acids at each of the five active site positions in our original list are made experimentally in all possible combinations. While this technique creates some unwanted sequences, the resulting combinatorial library is still much smaller than the original search space and about 70% of the library sequences are from the original list.

The second method for experimentally creating the feasible sequences uses an evolutionary map of the feasible sequences to suggest better starting points for the random mutation experiments mentioned in the Introduction. The evolutionary map of all 50000 sequences is too large to print, but to give an idea of the graph structure we show an evolutionary map in Figure 2 of the 49 compatible sequences for only 2 positions in the active site. In this case, we can see that some sequences have many neighbors, and therefore could easily mutate to other feasible sequences and might be good starting points for mutation experiments. The natural sequence, on the other hand, has no neighbors, indicating that a random mutation experiment starting from this sequence might be less successful.

4. Results and Future Work

Our computational analysis merges with experimental efforts underway in the Wittrup and Klibanov laboratories at MIT. The Wittrup lab uses yeast-display and high-throughput screening to completely search our combinatorial libraries, and the Klibanov laboratory uses advanced enzymology to thoroughly study the properties of variant enzymes. The first experimental screen of our Horseradish peroxidase sequence library was for specific catalysis of the L over the D conformation of tyrosine, both natural substrates of this enzyme. One mutant enzyme with increased specificity was found in our designed library, though the full enzymological characterization of the new enzyme has not been completed. That only one new sequence showed this modified catalytic activity indicates the rarity of catalytic function in sequence space. Further computational work must also be done to better characterize our evolutionary maps. For example, clusters in these graphs could yield other suggestions for mutation experiments. We hope that this joint theoretical and experimental approach will produce understanding of the inner workings of this enzyme through iterative analysis, design, and experimental testing. The techniques developed for this problem may also be generally applicable to other hard protein design problems, involving other functions, in the future.

References

Desmet, J., Demaeyer, M., Hayes, B., & Lasters, I. (1992). The dead-end elimination theorem and its use in protein side-chain positioning. *Nature*, *356*, 539–542.

Farinas, E. T., Bulter, E., & Arnold, F. H. (2001). Directed enzyme evolution. *Current Opinion in Biotechnology*, *12*, 545–551.

Glieder, A., Farinas, E. T., & Arnold, F. H. (2002). Laboratory evolution of a soluble, self-sufficient, highly active alkane hydroxylase. *Nature Biotechnology*, *20*, 1135–1139.

Heering, H. A., Smith, A. T., & Smulevich, G. (2002). Spectroscopic characterization of mutations at the phe(41) position in the distal haem pocket of horseradish peroxidase c: structural and functional consequences. *Biochemical Journal*, *363*, 571–579.

Leach, A. R., & Lemon, A. P. (1998). Exploring the conformational space of protein side chains using dead-end elimination and the a* algorithm. *Proteins: Structure, Function, and Genetics*, *33*, 227–239.

Morawski, B., & Arnold, F. H. (2001). Functional expression and stabilization of horseradish peroxidase by directed evolution in saccharomyces cerevisiae. *Biotechnology and bioengineering*, *76*, 99–107.

Analyzing Provider and User Incentives Under Congestion Pricing on the Internet

Steven Bauer BAUER@CSAIL.MIT.EDU
Peyman Faratin PEYMAN@CSAIL.MIT.EDU
MIT Computer Science and Artificial Intelligence Laboratory, 32 Vassar Street, Cambridge MA, 02139 USA

1. Overview

The networking community has a long history of research investigating ways to price network traffic. The academically favored approach often involves some form of congestion pricing (B. Briscoe, 2005; B. Briscoe, 2003). However, congestion pricing has been criticized for creating a potentially perverse incentive for providers to cause artificial congestion by under-provisioning their networks since increased congestion will increase their revenue (Shenker et al., 1996). In our work we demonstrate that even if network providers, disciplined perhaps by market competition, do not have an incentive to cause artificial congestion, that users actually have surprising, perverse incentives to cause artificial congestion under congestion pricing. We analyze these incentives using a repeated game theoretic approach. This result is important because it suggests that even if congestion pricing were technically feasible, market acceptable and cost effective to implement, that it still may not be a desirable pricing mechanism on the Internet.

2. Background

The argument for congestion pricing is based on the theory that it is the only economically efficient usage-based pricing mechanism (MacKie-Mason & Varian, 1995). As the argument often goes, the "marginal cost of bandwidth is zero" if there is no congestion, therefore, under a competitive marketplace, the usage-price during those periods should also be zero.

However, congestion pricing is not without its critics. Both economic and technical arguments (Shenker et al., 1996), and arguments based on historical evidence (Odlyzko, 2001) have pointed out the shortcomings of congestion pricing. Some of the critiques of congestion pricing coupled with the traditional responses are presented in table 1.

Congestion-pricing problems	Traditional responses
1. Congestion pricing potentially creates an incentive for providers to under-provision, creating artificial congestion	1. Competition disciplines the market so that providers will have an incentive to expand capacity when congestion occurs
2. Recovering large sunk costs may become infeasible under marginal cost pricing	2. Two-part tariffs allow the recovery of fixed costs with the fixed access charge plus congestion charges that account for the congestion costs
3. Technically difficult to implement in a cost-effective manner	3. Clever engineering solutions are possible (e.g. leveraging ECN bits)
4. Congestion pricing may be unacceptable to users	4. Congestion pricing is the only economically efficient approach so it is the only approach that will survive long-term

Table 1. Traditional congestion pricing critiques and responses

3. User incentives under congestion pricing

In our work we explore what would happen if the first three problems in table 1 could be adequately addressed by a congestion pricing mechanism. We assume that providers expand capacity once they have received sufficient congestion revenue (a capacity expansion assumption illustrated in figure 1). We also assume that access charges cover provider's fixed costs and that congestion charging is technically implementable.

The capacity expansion assumption implies that providers do not have an incentive to cause artificial congestion (for instance by intentionally under-provisioning their networks.) Rather after a provider-determined sufficient amount of revenue is collected, the provider will expand capacity. Providers may for instance use the congestion charges to retire debts from a previous capacity expansion before incurring new expansion debts, or they may use the congestion charges to directly purchase additional routers

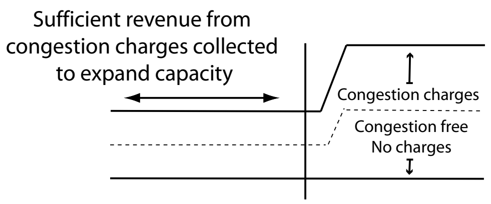

Sufficient revenue from congestion charges collected to expand capacity
Congestion charges
Congestion free
No charges

Figure 1. Capacity expansion assumption: revenues from congestion charges provide the financial basis for expanding a network.

or network links. In other words, under the capacity expansion assumption, congestion charges represent more than pure profit for a provider – they represent the financial basis for growing a network.

4. User incentives under congestion pricing

To formalize the analysis of user incentives and the strategy space we employ a repeated game model. Game theory provides the right analytic tools and also allows us to demonstrate that while congestion pricing in a single shot game induces efficient user behavior, surprisingly congestion pricing in a *repeated game* introduces incentives for some users to create inefficient artificial congestion. This is consistent with the recent work of Afergan (Afergan, 2005) that illustrates the importance of considering the repeated context of games that model networking – a problem area that inherently involves many repeated processes.

The intuition behind this perverse incentive is that users can lower their own overall long-term contribution to a capacity expansion cost by paying smaller penalties (i.e. smaller congestion charges) earlier (discounting the time value of money appropriately), thereby enabling their later and larger amounts of traffic to enjoy the benefit (i.e. a congestion free expanded network capacity). By causing congestion in earlier time periods a selfish user can induce other players that would have been "free riders" – sending traffic while there was no congestion – to now contribute to the capacity expansion cost.

Informally stated, a subgame perfect equilibrium of this repeated game is for users to adopt a congestion-dependent strategy. The user initially sends their normal traffic (delaying as long as tolerable to avoid congested periods). However, once the user forms an expectation that they cannot avoid incurring congestion charges for a large portion of their normal traffic, they have an incentive to prevent free riders during uncongested periods. The user sends artificial traffic during some of these uncongested periods to try to cause artificial congestion. The additional artificial congestion benefits the user by causing the revenue required to expand the network to be extracted sooner and from users that would not have normally paid for their traffic.

5. Summary

Practically speaking, individual users are unlikely to be sophisticated enough to strategize in such a way. However, were congestion pricing to be deployed on the Internet, an inevitable result would be that applications (or operating systems) would become strategic in when and how they generate traffic on the network. (We refer to this as *congestion-strategic traffic behaviors.*) Indeed one of the goals of congestion pricing is to create incentives for network traffic to be modified given the level of congestion. So while individual users would likely not make this a practical problem, the capability of classes of applications to exhibit strategic behaviors makes understanding the incentives created by congestion pricing a very relevant issue.

To summarize, the main contributions of this work are:

- A set of new models, Congestion-Pricing Traffic Games, for analyzing the incentives of congestion pricing on the Internet.
- Demonstration of subgame perfect equilibriums of the Congestion-Pricing Traffic Games.
- An analysis of congestion-strategic traffic behaviors and the resulting implications for the viability of congestion pricing on the Internet.

References

Afergan, M. (2005). *Applying the repeated game framework to multiparty networked applications*. Doctoral dissertation, Massachusetts Institute of Technology.

B. Briscoe, A. Jacquet, C. D. C.-G. e. (2005). Policing congestion response in an internetwork using re-feedback. *Proceedings of the ACM SIGCOMM '05 Conference*. Philadelphia, Pennsylvania.

B. Briscoe, V. Darlagiannis, O. H. (2003). A market managed multiservice internet. *Computer Communications, 26(4):404–414, 2003.).*

MacKie-Mason, J., & Varian, H. (1995). Pricing congestible network resources. *IEEE Journal on Selected Areas in Communications*, *13*, 1141–1149.

Odlyzko, A. (2001). Internet pricing and the history of communications. *Computer Networks (Amsterdam, Netherlands: 1999)*, *36*, 493–517.

Shenker, S., Clark, D., Estrin, D., & Herzog, S. (1996). Pricing in computer networks: Reshaping the research agenda. *ACM Computer Communication Review*, *26*, 19–43.

Integrating on a Spatial Network Without Coordinates

Jacob Beal JAKEBEAL@MIT.EDU
MIT Computer Science and Artificial Intelligence Laboratory, 32 Vassar Street, Cambridge MA, 02139 USA

1. Introduction

Calculating surface and volume integrals is a useful operation for space-approximating networks. For example a sensor network might estimate crowd size by integrating the density of people in an area, or estimate the severity of a pollutant spill by calculating the area of the affected region. More exotically, a smart material might integrate force to find the weight it is supporting, or a bio-film computer might monitor its activity by integrating the flow of nutrients through a volume.

Calculating such an integral involves interpolating the collection of values measured at network nodes across space, a problem which is non-trivial if the distribution of nodes through space is not known in advance. Spatial interpolation is a well-studied problem with a large family of applicable methods.[1] These methods, however, generally depend on knowing the spatial coordinates at which values are measured, and coordinates for nodes may not be reliably available to a space-approximating network.

I present a method for finding surface integrals without coordinates. My method assumes that nodes communicate only with their nearby neighbors, then integrates by summation using Thiessen weights estimated from expected communication range and number of neighbors.

2. Method

Thiessen weighting(Thiessen, 1911) is a simple discrete approximation of spatial integrals commonly used in GIS problems. A Voronoi diagram is calculated for the network and each node is assigned a weight equal to the area of its cell. The integral is then estimated as the weighted sum of the measurements.

Without coordinates we cannot calculate a Voronoi diagram, but we do not actually need to know which space belongs to a particular cell, only the total area of the cell. For interior nodes, this can be estimated from the density of the network: if there are ρ nodes per unit area, then the expected area occupied by each node is $1/\rho$.

[1]See, for example, (Lam, 1983), (Burrough & McDonnell, 2000), (Meyers, 1994), or (Cressie, 2003) for a thorough review.

To estimate the Thiessen weight of an interior node i, we assume that nodes are distributed uniformly through space at an unknown density.[2] Thus, given an expected communication range r, if i has n_i neighbors then its estimated Thiessen weight is

$$w_i = \frac{\pi r^2}{1 + n_i}.$$

Note that since this estimate uses local information, it is adaptive for regions with differing local density, but may fail for regular distributions of nodes (e.g. grids).

Nodes at the edge of the network are more difficult to handle, since their Voronoi cells may be infinite. Thiessen weighting clips these cells against a boundary, which cannot be done without coordinates. Moreover, without coordinates or distribution priors, it is not even possible to distinguish between edge nodes and interior nodes.

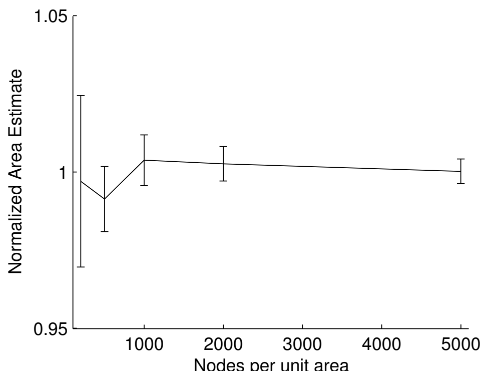
Since we cannot distinguish between edge nodes and interior nodes, we will instead examine the behavior of our weight estimation w_i at the edge. As we approach the edge of a uniform density space-approximating network, the apparent density decreases, effectively causing edge nodes to represent some of the empty space within their communication range.

We can calculate the effective increase in area beyond the edge by finding the expected increase in weight for edge nodes. For simplicity, we will ignore corners and consider a square rather than circular neighborhood. If x is distance from the edge, then the effective area is multiplied by $\frac{2r}{x+r}$. The expected multiplier is thus

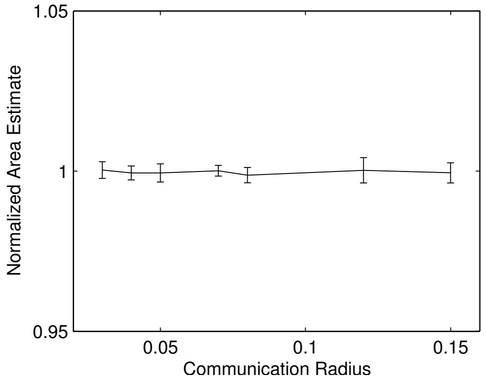
$$\int_0^r \frac{1}{r}\frac{2r}{x+r}dx = 2\ln(r+x)|_{x=0}^{r} = (2\ln 2)$$

so we should expect integrals to extend the edge of a space-approximating network by about 1/3 of a communication radius. This can be compensated for in deployment or simply factored into expected error.

[2]This is not a probability distribution, because the measure is not well defined.



(a) Accuracy vs. Density



(b) Accuracy vs. Radius

Figure 1. Normalized area of the interior of a square. Graph (a) shows small error and decreasing variance as node density ranges from 200 to 5000 per unit area and communication radius is fixed at 0.12. Graph (b) shows minimal error when communication radius ranges from 0.018 to 0.15 units and node density is fixed at 5000.

3. Experimental Results

I tested these predictions in simulation, using 100 to 10,000 nodes uniformly randomly distributed through a rectangle with unit area and an aspect ratio ranging from 1 to 16. Nodes were connected to all neighbors within the communication radius, which ranged from 0.018 to 0.15 units distance in different experiments. I then integrated the constant 1 through regions of the rectangle to calculate area.

Integration in the interior yielded answers deviating from correct by less than 3% independent of communication radius and with variance decreasing with node density (Figure 1). Integration of the entire rectangle produced deviations close to the predicted values (Figure 2): a least squares fit yields an equation for estimated area $A = 0.32 \cdot c \cdot r + 1.00$, where c is the circumference of the rectangle and r is the communication radius.

4. Conclusions

Estimating Thiessen weights from number of neighbors is an effective method for calculating approximate integrals in space-approximating networks. When calculating integrals, the boundary of the network is effectively extended beyond the outermost nodes by approximately 1/3 of a communication radius.

Further investigation is needed to verify that this method is effective for integrating other functions and for networks with variable density and more realistic connectivity, particularly networks with poor connectivity. Finally, Thiessen weights are crude compared to other spatial interpolation methods; perhaps more sophisticated methods can also be adapted for an environment without coordinates.

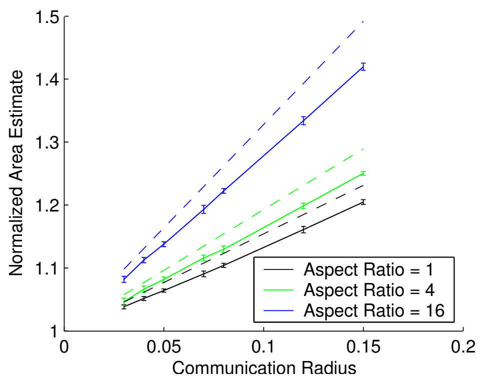


Figure 2. Estimated area versus radius for rectangles of various aspect ratios in a network with 5000 nodes (dashed lines are predicted values). The extra area is proportional to communication radius and circumference, but independent of node density.

References

Burrough, P., & McDonnell, R. (2000). *Principles of geographical information systems*. Oxford University Press.

Cressie, N. (2003). *Statistics for spatial data, revised edition*. John Wiley & Sons.

Lam, S. (1983). Spatial interpolation methods: a review. *American Cartographer*, *10*, 129–49.

Meyers, D. (1994). Spatial interpolation: an overview. *Geoderma*, *62*, 17–28.

Thiessen, A. H. (1911). Precipitation for large areas. *Monthly Weather Review*, *39*, 1082–1084.

Morphogenesis on an Amorphous Computer

Arnab Bhattacharyya abhatt@csail.mit.edu
MIT Computer Science and Artificial Intelligence Laboratory, 32 Vassar Street, Cambridge, MA 02139 USA

1. Introduction

Embryological development is a magnificent demonstration of how complexity can arise from initial simplicity. A single egg cell contains most of the information needed to position the millions of cells in a human body; moreover, the construction process is remarkably robust in that it can recover from a large number of cell deaths and malfunctions. Our goal is to understand the major principles involved in the developmental process so that they can be usefully applied to areas in computer science. Specifically, in this paper, I use motivation from morphogenesis and developmental biology to describe a scheme for creating systems that dynamically and robustly self-assemble into desired shapes. What follows will describe this scheme as well as the computational framework used to implement it.

2. The Computational Framework

The central challenge in imitating biology is managing the huge amount of concurrent, largely decentralized, behavior exhibited by the cells. How do you program a massively distributed, noisy system of components? This problem is formalized in the study of *amorphous computing*. An amorphous computing medium (Abelson et al., 2000) is a system of tiny, computationally-limited elements scattered irregularly across a surface or a volume. These elements, which we shall call *cells*, have a limited range of communication, can retain some local state, are identically programmed, and can replicate and kill themselves. These cells, thus, are analogous to biological cells. The goal of the amorphous system is to attain some desired global state. So, while the cells are only locally interacting, their collective behavior should result in complex global behavior. Moreover, we want the amorphous computation to be robust; that is, the computation should still proceed in the face of random cell death/failure.

An individual cell has no *a priori* knowledge of where it is located with respect to the global structure. However, eventually, each cell must get some idea of where it is located in order to be differentiated properly from its neighboring cells. In biology, regional specification of the embryo is accomplished mainly through the use of *gradient fields* (Nüsslein-Volhard, 1996; Slack, 1991). Embryonic cells respond to a particular chemical, called a *morphogen*, whose concentration increases in some direction, forming a gradient; thus, cells differentiate based on the concentration of morphogen at the positions they are located. We use a similar strategy for regional specification in the amorphous computing medium. A cell can release a morphogen that is uniquely identified. In our model, the concentration of the morphogen decreases linearly with the distance from the source. Cells in the amorphous medium have receptors with which they can query the local morphogen concentration; thus, they can adapt their behavior based on the inferred distance from some morphogen source. In addition to the morphogens released by the cells, there could also exist background gradient fields. These fields serve to orient the very initial cells in development; so, functionally, they are similar to the Bicoid morphogen in *Drosophila* which establishes the fruit-fly embryo's anterior-posterior axis at the earliest stages.

3. The Problem and Related Work

Here, we apply the above computational paradigm to the creation of arbitrary (two- or three-dimensional) shapes from amorphous elements. The amorphous scheme should have the following properties:

- shape generation should resemble biological development in that an initially small group of cells should replicate repeatedly until the desired shape is attained
- cells should be allowed to move during the course of development
- the shape development should be regenerative, meaning that if some portion of the shape is suddenly cut off during any part of the development process, the rest of the shape should be able to regrow the missing part
- the number of gradient fields used should be minimized

Amorphous shape generation has already been discussed in some previous works. Attila Kondacs (Kondacs, 2003) presents a scheme that compiles an input global two-dimensional shape into a program that provides local instructions to cells in an amorphous medium to grow into the desired global configuration via replication and local interaction. Kondacs' scheme is regenerative, especially with respect to random cell death. Clement and Nagpal (Clement & Nagpal, 2003) also develop an algorithm to create regenerating spatial patterns. They introduce the concept of active gradients to implement self-repairing topologies.

The ideas described here address problems a bit different from those treated in both the above cited works. Firstly, the proposed regenerative scheme will work for both two and three dimensional shapes. Secondly, the shapes generated will be dynamic since the cells are mobile and regeneration takes place while the shape is developing. And thirdly, the development scheme is intended to allow a more accurate modeling of the morphogenesis patterns found in nature. A closer modeling will provide deeper insight into some puzzling questions in theoretical morphogenesis and pattern formation, such as the question of how deuterostomes evolved from protostomes (Sussman, 2005).

4. Growing and regenerating shapes

The central idea in our scheme is that cells can denote developmental processes in addition to points in space. We will call such cells *organizers*. The state of an organizer is a recursive program that creates other organizers. Thus, one could think of our system as a layer of amorphous programs with each organizer in one layer containing the embryo for some of the organizers in the next layer, with the final layer consisting of the cells that make up the desired shape.

In the actual implementation, all the organizers have a physical presence. In the beginning of the self-assembly process, there exists a single organizer at stage-one. This organizer is the embryo for stage-one. There also exists an external gradient field, and the stage-one embryo moves toward higher concentrations of this morphogen. When the embryo has reached the maximum, it stops moving and starts replicating to produce other stage-one organizers. (The children of an organizer might be placed in the direction of a gradient field; thus, a colony of organizers can grow toward an increasing gradient.) At some point determined by its state, a stage-one organizer starts producing stage-two organizers. Thus, every stage-one organizer becomes an embryo for stage-two. This process recurses with each stage-n organizer replicating into other stage-n organizers and then, at some point, becoming embryos for stage-$(n+1)$. At any point, an organizer can secrete morphogens to set up gradient fields, query local morphogen values, move toward higher values of a morphogen, or kill itself (mimicking apoptosis in biological cells). Organizers also have a physical property called *adhesiveness* which determines how an organizer physically adheres to its neighbors.

Regeneration is made easier because organizers can regenerate all the organizers in the stages subsequent to it. Growth is modular instead of incremental. At each stage, regeneration might take place by the active gradient scheme of (Clement & Nagpal, 2003) or by the reference-point competition scheme of (Kondacs, 2003). These schemes have to be modified to take into fact that the cells are mobile. The multiple growth stages can account for a wide range of types of cell deaths. Since the organizers operate asynchronously, regeneration on multiple stages could be occurring at the same time. Also, note that if after development, the capabilities of the lower stages are permanently discarded, then some gross region deaths cannot be repaired while most minor ones can. This is similar to what happens in newts, for example.

5. Future Work and Acknowledgments

The ideas described above are in the process of being implemented. So, they will surely undergo some revisions as new problems and issues are discovered. Also, much as in (Clement & Nagpal, 2003), the regeneration process fails if the embryo of the very first stage is removed. A way to replace the stage-one embryo would make the regeneration process more complete.

I would like to thank Gerry Sussman and Hal Abelson greatly for helping me understand the issues in amorphous computing and for encouraging me to work on this problem.

References

Abelson, H., Allen, D., Coore, D., Hanson, C., Homsy, G., Knight, T., Nagpal, R., Rauch, E., Sussman, G., & Weiss, R. (2000). Amorphous computing. *Communications of the ACM*, *43*.

Clement, L., & Nagpal, R. (2003). Self-assembly and self-repairing topologies. *Workshop on Adaptability in Multi-Agent Systems, RoboCup Australian Open*.

Kondacs, A. (2003). Biologically-inspired self-assembly of two-dimensional shapes using global-to-local compilation. *International Joint Conference on Artificial Intelligence*.

Nüsslein-Volhard, C. (1996). Gradients that organize embryo development. *Scientific American*.

Slack, J. (1991). *From egg to embryo*. Cambridge, UK: Cambridge University Press.

Sussman, G. (2005). Two Ways to Make a Tube from a Bag. Unpublished manuscript.

A Prototype Web User Interface for Scalable Medical Alert and Response Technology

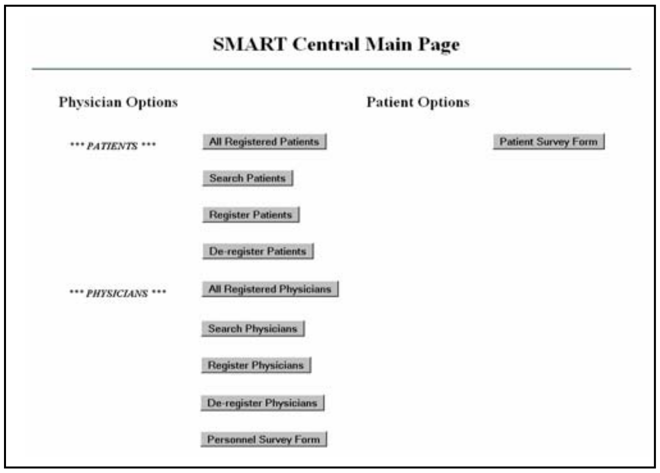
Sharon H. Chou CNORAHS@MIT.EDU
MIT Computer Science and Artificial Intelligence Laboratory, 32 Vassar St., Cambridge MA, 02139 USA

We present a prototype web application for the Scalable Medical Alert and Response Technology (SMART) [1]. The project aims to wirelessly monitor vital signs and locations of otherwise unattended patients in the waiting area of an emergency department. Waiting patients would feel secure that their condition is being monitored even when a caregiver is unavailable. Caregivers will be alerted in real-time to problems occurring in the waiting room, while threshold values for patient alerts and priorities can be dynamically adjusted. We hope to gain some insights from deployment in the emergency room settings, which could be extended to disaster situations where patients drastically outnumber available caregivers. This application is currently in development and will be tested in actual emergency rooms in the near future.

SMART Central system description

The system links caregivers, patients, a streaming database that continuously updates patient status, and a processor that analyzes the patient data for alarm conditions. Alarm conditions include high heart rate, low heart rate and low SpO_2 (blood oxygenation level). When an alarm condition is detected, it is dispatched to an available caregiver. Caregivers each have a PDA (HP iPAQ) that allows them to see the roster of patients and to click through to see a patient's vital signs in real-time. Patients wear a similar PDA as an interface for vital signs monitoring (EKG leads and oximetry sensors to measure SpO_2). A cricket location system tracks the patients and caregivers. [2]

Web User Interface Implementation



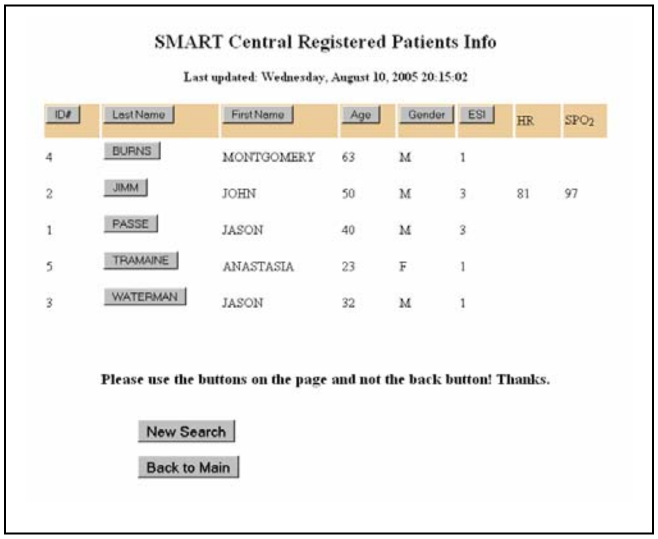
The web application facilitates patient monitoring for caregivers by providing a GUI to display relevant information of registered patients and physicians. The main page at http://nms.lcs.mit.edu/~cnorahs/smart-main.cgi is the portal for relevant information on physicians and patients registered with SMART Central, and it:

1. Display a roster of all the patients with their names, ID numbers, age, gender, ESI (Emergency Severity Level), heart rates, and SpO_2.
2. Search the patient database by last name, age range, or ESI.
3. Register new patients and store their information in the PostgreSQL database.
4. De-register patients and remove them from the database.
5. Display all the physicians with their names, ID, and locations

6. ~ 8. Search, register and de-register physicians

9. ~ 10. User surveys for both physicians and patients.

Clicking on the "All Registered Patients" link displays as below:



SMART Central Registered Patients Info

Last updated: Wednesday, August 10, 2005 20:15:02

ID#	Last Name	First Name	Age	Gender	ESI	HR	SPO_2
4	BURNS	MONTGOMERY	63	M	1		
2	JIMM	JOHN	50	M	3	81	97
1	PASSE	JASON	40	M	3		
5	TRAMAINE	ANASTASIA	23	F	1		
3	WATERMAN	JASON	32	M	1		

Please use the buttons on the page and not the back button! Thanks.

The text, tables and buttons were coded in HTML. The CGI scripts for sorting, searching, and communicating with the PostgreSQL database were written in Python.

Clicking on the column header buttons sorts the roster by that column. The text to be sorted is passed as a hidden string variable to the reloaded HTML page, where the CGI script would parse the string and display it in a table format. The figure above comes up after clicking on the "Last Name" button.

The columns for heart rate (HR) and SpO_2 automatically update at preset intervals (i.e. 30 seconds). The data arrives in continuous streams from the patient's sensor leads (or the PostgreSQL database files in the simulated case). To enable communication between the data processor and the monitoring device (or database), we import the Python *socket* module into the CGI script. The CGI for the webpage sends a query to the streaming database or a patient's iPAQ, which returns data streams packaged and interpreted by the Python *struct* module. The resulting data is displayed on the patient information webpage. In this example, all the patients are simulated. Patient #2 (John Jimm) has streaming data set up (87 beats/min and 97% SpO_2). Software development for the data stream processing was based on source code by Jason Waterman.

The automatic update function is coded by JavaScript embedded in the Python CGI. The JS also codes the time counter for the last update.

Clicking on one of the last names opens a page with more detailed information about the patient's alarm conditions, allergies, and current medications.

A login script [3] provides restricted access for SMART Central. Only the registered physicians can access the main page and all the links within. The login script is an open-source Python utility written by Michael Foord.

Lastly, there are two online user surveys with free-response and Likert scale questions to gather user feedback from patients and caregivers. The questions were written by the clinical decision group at Brigham's Women Hospital.

Benefits

This system provides an easy-to-use interface for emergency room caregivers to access patient information from any web terminal. The automatic page refresh ensures that medical data is updated frequently and reliably, essential for an emergency room setting. With the location tracking functionality implemented, physicians could locate patients and other physicians quickly.

Challenges

The first major design challenge encountered was implementing the sorting function. Since HTML is a stateless language, all the information from the previous unsorted page must be passed to the sorted page as a hidden variable. The syntax for hidden variable passing took some exhaustive search and debugging. Next, parsing the hidden variable string for display on the sorted page called for modularized subroutines tested sequentially for robustness.

Secondly, meticulous proofreading was required to combine CGI code in Python with the JavaScript for automatic page refresh and time stamps. This was equivalent to making a Python program generate JavaScript, and involved much quotation matching for displaying strings.

Another challenge was getting the streaming data from the Python *socket* module to work together with web page refreshes. When the browser sends a query to the monitoring device, the device would return data to be interpreted by the Python *struct* module and displayed on the browser. This data transfer would result in a few seconds of delay, making the automatic page reload sometimes cumbersome. Since speed and accuracy are crucial for the web application to work in clinical settings, work is underway to speed up communication between the browser and the monitoring devices. Simulations had shown the information to be accurate yet not as efficient as desired.

Alternative Implementation Methods

Instead of Python, Perl, C, or Java could also be used. Python is a powerful language with versatile string manipulation functions, ideal for parsing strings in this web application. Furthermore, its clean syntax was easier to debug and decreased development time. However, Java or C might be more compatible with the embedded JavaScript.

Further Developments

- Incorporate location tracking of caregivers as a search query for patients who need specialized assistance
- Increase performance efficiency of the streaming PostgreSQL database

Acknowledgements

This prototype was developed under the supervision of Dorothy Curtis and Prof. John Guttag.

References

[1] SMART Project Overview web page
http://smart.csail.mit.edu/

[2] MIT subcontract technical proposal
http://smart.csail.mit.edu/publications/MIT-tech.doc

[3] Python reference web page
http://www.voidspace.org.uk/python/index.shtml

A Usability Evaluation of Two Computer Vision-Based Selection Techniques

Jacob Eisenstein JACOBE@CSAIL.MIT.EDU
MIT Computer Science and Artificial Intelligence Laboratory, 32 Vassar Street, Cambridge MA, 02139 USA

1. Introduction

Conventional human-computer interaction techniques have been optimized for one specific usage case: a stationary individual whose full attention is devoted to the computer. The standard input hardware – a mouse and keyboard – are well-suited for this kind of interaction, but they function poorly in other usage cases. Desktop computers do not mesh well with other activities, such as (non-virtual) socialization, performing household chores, or watching TV (Norman, 1999).

Ongoing research in *Communication Appliances* focuses on designing technology for remote communication that can be seamlessly integrated into other daily activities. An interaction scheme that treats a computer as a workstation is not likely to be successful in this domain. New interaction techniques must be developed and evaluated.

The problem of selecting from a menu of choices is ubiquitous, not only in desktop computer user interfaces, but also in appliances such as microwave ovens, stereos, and thermostats. Such household items typically provide a custom interface of physical buttons, either on the device itself or on a remote control. But anecdotal experience tells us that as the functionality of these devices becomes more complex, such physical interfaces become increasingly difficult to use (Nielsen, 2004).

Computer-vision based interaction has been proposed as a solution (Freeman et al., 2000). Using a camera to track either an object or the user's body, there is no need to type or use a mouse; the user need not even approach the computer at all. At the same time, there is no complicated hardware interface to learn, and no remote control to find buried under the couch cushions. While vision-based interaction offers promising solutions for this usage case, our understanding of the usability principles of such interfaces is still at an early stage. This paper describes an empirical evaluation of two different computer vision-based interaction techniques for the problem of making a selection from a menu.

2. Selection Techniques

Two selection techniques have been implemented, using the OpenCV computer vision library. The first – *motion selectors* – is modeled after the Sony EyeToy. The user triggers a selector by waving a hand or otherwise creating motion within the selector. A meter increases with the amount of motion detected, and the selector is triggered when a threshold is reached. While the Sony EyeToy uses simple image differencing to detect motion, the system implemented for this experiment uses optical flow detection, which was thought to be more accurate.

Figure 1 demonstrates the motion selectors. This set of images are taken from the video that the participant himself actually observed while performing this experiment. As discussed above, the visual feedback of the system state is considered an important part of this interaction technique. In part (a), the desired target selector flashes, indicating the participant should select it. In part (b), the participant moves his hand towards the selector. Once the participant has reached the selector, he moves his hand within it (part (c)), and the activation level rises. When the activation level reaches the top in part (d), the selector is activated, and flashes to indicate this.

The second selection technique – *tracking selectors* – requires that the user place a real color ball within the selector. The ball is tracked using the camshift algorithm,with the backprojection computed by a set of histograms at several levels of precision in the YCrCb color space.

The system's guess of the location of the tracked object is indicated by an ellipse. The tracker is enable to assess its level of confidence in its own estimate, using goodness-of-fit metrics based on the size, shape, and color of the estimated location and boundary of the tracked object. A high level of certainty is indicated to the user by coloring the ellipse green; a low level is indicated by coloring the ellipse yellow.

Figure 2 shows several images of the tracker selectors, again taken directly from the video feed seen by the participant. In part (a), the system is not yet sure of the location of the tracked ball, as indicated by the large ellipse, which is colored yellow. As the participant moves towards

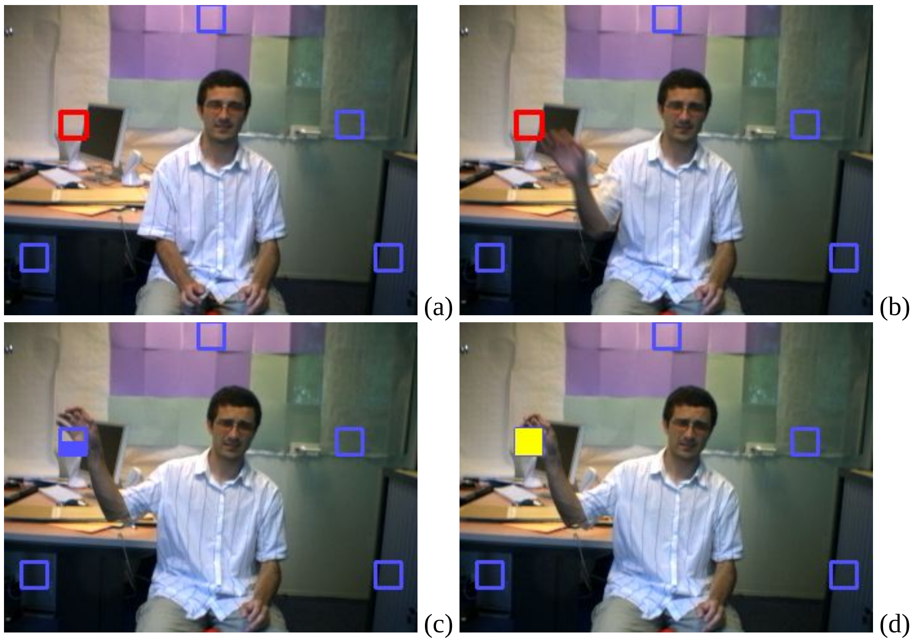


Figure 1. Motion selectors

the target, the tracker focuses in on the location, and ellipse becomes green (part (b)). In part (c), the system momentarily becomes confused, possibly because the tracked object is being moved very quickly, and the ellipse again turns yellow to indicate this. Without being instructed to do so, the user holds the ball still for an instant, enabling the system to recover. The selector is triggered instantaneously when the user moves the tracked object into it in part (d).

3. Experiment

An experiment was conducted to compare the speed, accuracy, and likability of the two techniques. Twelve people participated in this study, and none was previously familiar with either selection technique.

Participants were told that the goal of the experiment was to see how fast they could select a targeted button. The targeted button was indicated by making it blink. Participants were told to go as fast as possible, as long as errors were kept within reason. Participants underwent an initial training period in which they were allowed as many trials as they felt they needed to learn how to use each selection technique.

For both selection techniques, the buttons were laid out in an semi-ellipse, such that no button would come closer than 15 pixels from the edge of the screen. The entire experimental procedure was automated. Participants were instructed to keep their hands in their laps until one of the buttons started blinking; then they were to activate that button as quickly as possible. Afterwards, they were to return their hands to their lap. The system also enforced rest periods to prevent fatigue.

After this initial training period, six experimental blocks were conducted. Each block consisted of twenty trials of a single selection technique. Alternating blocks of each interaction technique were used, and the ordering was counterbalanced across participants. Within each block, the number of selectors was varied between 2,5,11, and 21; an equal number of each type was given within each block, and the order was determined randomly. Similarly, the location of the target button was varied, and each participant experienced an equal distribution across the semi-ellipse of buttons. The following results were logged: intended target, actual selection, and elapsed time for the trial.

4. Results

A two-factor within-participant analysis was used to analyze the effects of the selection technique and the number of selectors on two dependent variables: error rate, and selection time. Results are reported as statistically significant when $p < .05$.

The choice between motion selectors and tracking selectors did not significantly impact on the error rate, but the number of selectors did. The interaction between the number of selectors and the selection technique was also observed to have a statistically significant effect on error rate. These results are described in Table 1.

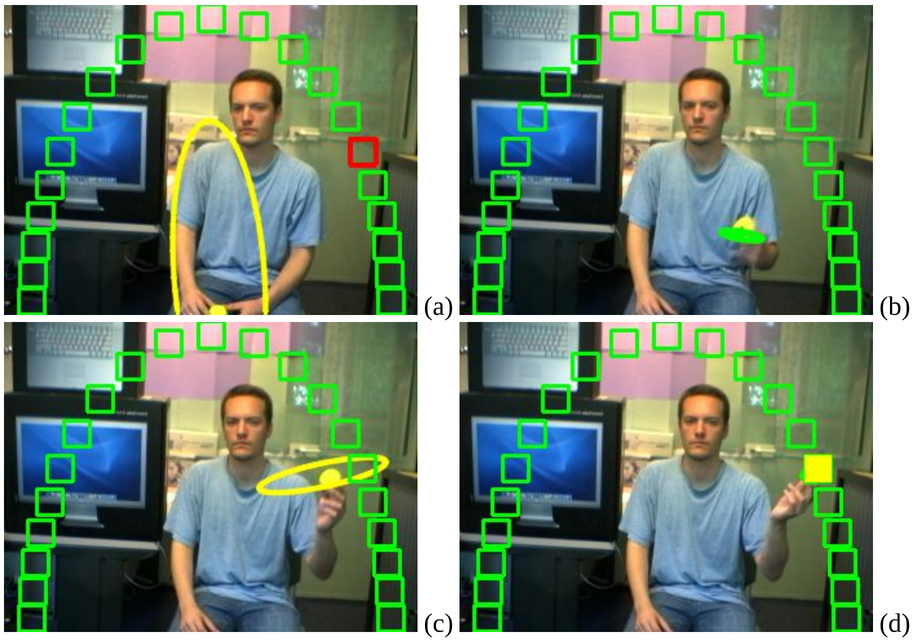


Figure 2. Tracking selectors

num. selectors →	2	5	11	21
motion selectors	0	0	7	37
tracking selectors	0	0	1	20

Table 1. distribution of errors

	tracking	motion	no pref
Which is faster?	10	1	1
Which is more accurate?	9	3	0
Which did you prefer?	6	5	1

Table 2. qualitative results

Users were significantly faster when using the tracking selectors, compared to the motion selectors. The average selection time was 1.94 seconds with the motion selectors, compared to 1.63 seconds with the tracking selectors. Although the average selection time increased monotonically with the number of selectors, this effect was not found to be significant. There were also no significant interactions between the two factors.

As described in Table 2, most of the participants believed the tracking selectors were faster and more accurate, but were more evenly divided as to which method they preferred.

5. Discussion

The tracking-based selection technique was not observed to be worse on any measure than the motion-based selection technique. It was observed to be faster, and to produce fewer errors when the number of selectors was very large.

As suggested by the qualitative results, speed and accuracy are not the only considerations that shape the participants' preferences. Both techniques have strengths and weaknesses which are more difficult to quantify in a controlled experiment. The tracking-based technique is robust to background movement, but was found to be not particularly robust to lighting changes. The motion-based technique does not require to user to keep a tracked object, but we had to go to great lengths to eliminate background motion that might confuse the system. A longer-term longitudinal study of how each system is actually used is necessary to assess the role of these factors.

6. Acknowledgements

This research was performed under the supervision of Wendy Mackay at the In Situ research group of INRIA.

References

Freeman, W. T., Beardsley, P. A., Kage, H., Tanaka, K.-I., Kyuma, K., & Weissman, C. D. (2000). Computer vision for computer interaction. *SIGGRAPH Comput. Graph.*, *33*, 65–68.

Nielsen, J. (2004). Remote control anarchy. http://www.useit.com/alertbox/20040607.html.

Norman, D. (1999). *The invisible computer*. The MIT Press.

Proovy: A Simple Proof Checker for Higher Order Procedures in Agent Planning

Harold Fox HFOX@CSAIL.MIT.EDU
MIT Computer Science and Artificial Intelligence Laboratory, 32 Vassar Street, Cambridge MA, 02139 USA

1. Introduction

When designing an intelligent agent, flexibility and introspection are critical attributes. That is, an agent's behavior should be dynamic and capable of change. Moreover, it should also be able to analyze its own behavior, so that it can diagnose failures and possibly even reprogram itself.

We have developed a simple, functional language, Proo, which can describe a wide range of agent behavior. Additionally, it allows the specification of theorems with checkable proofs. These theorems can formally prove that a given agent's behavior meets a given set of requirements in the context of a given environment.

The particular environments we are studying are deterministic, fully observable, and higher-order. We are motivated by the problem of getting an agent to learn how to operate a human computer interface such as a program for coordinating mobile robots. The assumptions of determinism and full observability are reasonable as a first approximation, since user interfaces strive to be predictable without hidden state. However, the agent's perception of the environment needs to be higher order, because it needs to model a software program, which is logically complex.

To be precise, the environment consists of a set of states $\mathcal{E}$. An agent consists of a set of actions $\mathcal{A}$. Each action A_i, is a function from $\mathcal{E}$ to $\mathcal{E}$. It manipulates the state of the world, producing a new state. An agent behavior, B, is a mapping from states E to action sequences in $\mathcal{A}^*$. A goal state of a behavior, g, is a state such that $B(g) = ()$. Proo allows us to define these actions and behaviors and prove properties about them.

Proovy is a first step of a research agenda to develop agents that can learn their own behaviors to act in a virtual world. Because we want our agents to be introspective, we want them to produce proofs that their behaviors are correct. Such behaviors and proofs would be represented internally using Proo constructs. Furthermore, the background knowledge that the agent would need to perform effectively in different domains can also be encoded and verified using Proo and Proovy.

2. Related Work

Proof assistants or interactive theorem provers are a well-established area of computer science used to prove mathematical theorems and verify the logical correctness of critical systems like computer chips (Kaufmann et al., 2000).

We are interested in a proof language where facts and implications can be written down clearly in a logical sequence as they would be in a mathematics text. Because proving theorems is not the final goal of this research, we need a system that can concisely represent proofs produced by an agent problem solver and check them in batch. The most advanced work in this area is the proof language Athena (Arkoudas, 2000). However, unlike Athena and the other denotational proof languages, Proovy is not concerned with the formal semantics of what makes one statement provable and another not. So, Proovy can apply multiple tactics at a time, searching a deep array of possibilities to match a given statement to TRUE. Its proofs can be shorter and more succinct.

Theorem provers have been used as the foundation of classical artificial intelligence planning (Fikes & Nilsson, 1971) from the beginning. However, planning has always been considered as a problem in first order logic. Little work has been done in worlds where an agent's actions have consequences that cannot be represented in first order logic.

3. The Proo Language

Proo is a simple language for defining functions and writing theorems. Its function notation is derived from Scheme, although the top level organization has its own syntax. It is a pure functional language with no side effects, so there is no notion of time or program execution. Objects consist of primitives, conses, and lambda functions.

Functions do not need to specify the types of their arguments, so any expression is syntactically legal and equality is determined by simple symbol comparison. Because of the unsoundness of the untyped lambda calculus, Proo is also not logically sound. Through paradoxical examples,

Figure 1. Proof of a lemma allowing proof by contradiction

```
theorem ctrdict (all (a) (-> (-> a F) (not a)))
  (lambda (a) {
    ctrdict.1 : (if (-> a F) {
      ctrdict.2 : (-> (not F) (not a)) [ctrpos ctrdict.1];
      (not a) [ctrdict.2];
    });
  });
```

one can prove that true is false. However, this is not as serious a problem as it would be for an official mathematical theorem prover. Since the human knowledge encoding and the automated reasoning are under our control, we can socially enforce that they not produce lambda functions that lead to paradoxes.

Proo does not allow recursive functions. To get the same behavior, there is a special operator called suchthat, which allows a prover to refer to an object that uniquely satisfies a particular property. For example, factorial would be defined in the following way:

```
Factorial := (suchthat (lambda (f)
  (= (f n)
     (if (= n 0) 0 (f (- n 1))))))
```

`suchthat` allows us to define a wider class of logical functions than traditional programming languages.

Theorems are defined with a claim and a proof (see figure 1). The proof consists of a sequence of facts with each fact deducible from previous facts in the proof as well as prior theorems, axioms, and definitions. The proof writer must specifically state the prior facts needed to prove a given fact. This enables the proof checker to constrain its search, so proofs can be shorter with fewer intermediate steps explicitly written down.

4. The Proovy Proof Checker

At its core, the proof checker uses a matching engine which takes two expressions and a list of free variables and attempts to find a binding to make the two expressions equal. It uses general rewrite rules that specify how a pair of expressions can be made to equal each other. When proving a fact, the proof checker attempts to match the fact with T, the truth primitive.

Whenever a match fails, the proof checker will use its current implication rules and attempt to match the predicate of each implication to `T`. When a binding matches successfully, the checker adds the consequence of the implication as a new rule to the matcher.

5. Using Proo To Specify Agent Behavior

With suitable lemmas and definitions, we can define an agent's actions. As a basic example, consider a 1x5 one-dimensional world of bits. All of the bits are set to `0` except for an active bit set to `1`. The agent has two actions, `LEFT` and `RIGHT`, which move the active bit coordinate down and up respectively. We can define the functions thus:

```
out := (lambda (n)
          (lambda (i) (if (= i n) 1 0)));
down := (lambda (n) (if (= n 0) 0 (- n 1)));
up := (lambda (n) (if (= n 4) 4 (+ n 1)));
inv := (lambda (f) (lambda (y)
  (suchthat (lambda (x) (= y (f x))))));
LEFT := (lambda (s)
             (out (down ((inv out) s))));
RIGHT := (lambda (s)
             (out (up ((inv out) s))));
```

We can similarly define agent behaviors to produce sequences of actions given a particular goal state. With these definitions, we can prove that the action sequence produced by a behavior will correctly reach any goal state from any input state.

The meta-knowledge provided by Proo lets programmers formally describe and analyze agent behavior. In the future, we hope to be able to build agents that could discover their own behaviors from such action descriptions. Agents could then know why they were acting in a certain way instead of just instinctively reacting to the world.

References

Arkoudas, K. (2000). *Denotational proof languages*. Doctoral dissertation, MIT.

Fikes, R. E., & Nilsson, N. J. (1971). STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence* (pp. 189–208).

Kaufmann, M., Manolios, P., & Moore, J. S. (2000). *Computer-aided reasoning: An approach*. Kluwer Academic Publishers.

Hierarchical Recursive Feature Elimination: A Proposed Method for Reducing the Set of Features Used in an EEG-based Epileptic Seizure Detector

Elena Leah Glassman ELG@CSAIL.MIT.EDU
Networks and Mobile Systems Group, MIT Computer Science and Artificial Intelligence Laboratory, 32 Vassar Street, Cambridge, MA 02139 USA

1. Introduction

This research is concerned with reducing the number of channels and features computed from each channel of an EEG-based, patient-specific epileptic seizure detector (Shoeb, 2003), while still maintaining performance. The current Recursive Feature Elimination (RFE) (Guyon et al., 2002) implementation we employ can significantly reduce the number of channels (and therefore electrodes) within practical time constraints. Yet RFE, an SVM-based greedy feature elimination algorithm that has become an established method for feature selection, can be computationally expensive. Numerous methods have been introduced to make RFE run within practical time constraints, such as E-RFE (Furlanello et al., 2003b), SQRT-RFE, and 1-RFE (Furlanello et al., 2003a).

The proposed hierarchical RFE method is another approach to speeding up the RFE process, which it is hoped will enable the removal of not only channels but individual features computed from the channels as well. Such a reduction will decrease overall computational complexity and power consumption, which will be especially important for mobile seizure detector systems

2. Background

The electroencephalogram (EEG) is an electrical record of brain activity that is collected using an array of electrodes uniformly distributed on a subject's scalp. The seizure detector (Shoeb, 2003) declares onsets based on a collective examination of features extracted from 21 EEG channels. A channel is defined as the difference between a pair of (typically adjacent) electrodes. The detector is trained and tested on the dataset collected by Shoeb.

RFE uses the internal workings of a Support Vector Machine (SVM) to rank features. An SVM is trained on feature vectors derived from examples of two classes. The apparent importance of each feature is derived from the orientation of the class-separating hyperplane. The feature(s) with the least apparent importance are removed. The remaining features are used to retrain the SVM for the next iteration. The process is repeated recursively until some stopping criterion is met (Guyon et al., 2002).

3. Motivation and Proposed Method

Since the number of features (4 per channel, 84 total) is large, the existing system does elimination on a channel-by-channel basis (Lal et al., 2004) At each iteration, the channel whose features contribute least to discrimination between the seizure and non-seizure classes is removed. If one begins with 21 channels and stops when all but one have been eliminated, the process takes one to three hours for each patient on a state-of-the-art desktop computer but can take as many as seven hours for patients with a relatively large amount of recorded data or whose seizure and normal data are particularly hard to discriminate between Using this process to eliminate features individually would increase the processing time by a factor of four. Posed as a search space problem, how can this space, which is too large, be effectively narrowed?

Because the patient-specific detector is trained on only one patient's data, the number of data points is small; the ratio of abnormal (seizure) class data points to features is roughly 1:1. The recommended ratio of data points to features is at least 10:1 (Jain et al., 2000) Whittling down the channels to the best subset is, consequently, vulnerable to random variation, possibly causing unimportant channels to be kept while important channels are eliminated. This would be evident if a patient's seizures were localized to one side of the head, and the final selected subset included an electrode from the opposite hemisphere.

To prevent this and integrate a priori knowledge, one can consider meta-features: groups of spatially similar channels. Constraining the selection of channels by grouping them based on anatomical information could make the process more robust.

This paper proposes an algorithm for finding a robust subset of channels, disregarding the unnecessary features from those channels, and completing the process within practical time constraints. The algorithm takes advantage of the data's hierarchical structure by first performing recursive group elimination until a stopping criterion (such as a threshold of performance) is met. Then recursive channel elimination is performed on the channels remaining after eliminating unnecessary groups. Finally, recursive feature elimination is performed on the features from the remaining channels. The same stopping criterion is used at each level. This algorithm will be referred to in the rest of this paper as Hierarchical Recursive Feature Elimination or H-RFE.

4. Related Work

Though a literature search for hierarchical feature selection retrieves many papers using this terminology, it was only after this paper's submission that a manuscript (Dijck et al., 2005) was found, still in press, proposing hierarchical feature selection in a manner similar to H-RFE. The main difference is that the method proposed by Dijck et al. bundles correlated features instead of using a priori knowledge to bundle features (which will not necessarily be correlated).

Since H-RFE is a multiresolution feature selection method by which it is possible to zoom in on a subset of features and potentially narrow the search space rapidly, it is similar to methods that remove an exponentially decreasing number of electrodes at each iteration. Yet H-RFE incorporates scale: it transitions from eliminating groups of channels (low resolution) to individual features (high resolution).

This proposed method is similar to Entropy-based RFE (E-RFE) in its ability to automatically adapt to the data (Furlanello et al., 2003b). With E-RFE, the number of features removed in any one iteration is based upon the percentage of features determined to be unimportant. If a data-based stopping criterion is used, H-RFE will adapt to the data as well, determining at each scale when no further elements should be eliminated and it is necessary to recurse down to a finer scale.

5. Initial Results and Future Work

Though H-RFE has not yet been completely implemented, its operation on a single (channel) scale is functional and has been applied to twenty patients' data. The stopping criterion is based on the margin, or distance between classes' support vectors, in the feature space of the SVM used to rank the features, as well as cross-validation accuracy.

For the twenty patients analyzed so far, the number of channels was reduced, on average, from 21 to 8.35. Compared to the original 21-channel seizure detectors, the reduced-channel detectors had a 12 percent increase in latency (0.9 seconds), with an 11 percent decrease in false alarms and 12 percent increase in misses, on average (results pooled across all patients). Eventually, using the full multiscale implementation of H-RFE, it may be possible, as a result of narrowing the search space through the use of multiple scales (such as groups of channels), to eliminate superfluous features from the remaining channels within a practical amount of time. Though the concept of hierarchical feature selection has already been recently proposed, this method of bundling features–not based on statistical correlations, which are vulnerable to random variation when analyzing small datasets, but on a priori knowledge–may be particularly suited to problems in which the data lends itself to hierarchical organization.

6. Acknowledgements

I gratefully acknowledge many helpful discussions with Dorothy Curtis, Martin Glassman, Prof. John Guttag, Daniel Leeds, Eugene Shih, Ali Shoeb, and Zeeshan Syed.

References

Dijck, G. V., Hulle, M. M. V., & Wevers, M. (2005). Hierarchical feature subset selection for features computed from the continuous wavelet transform. *Proc. 2005 IEEE Workshop on Machine Learning for Signal Processing*. Mystic, Connecticut, USA. In press.

Furlanello, C., Serafini, M., Merler, S., & Jurman, G. (2003a). An accelerated procedure for recursive feature ranking on microarray data. *Neural Networks*, *16*, 641–648.

Furlanello, C., Serafini, M., Merler, S., & Jurman, G. (2003b). Entropy-Based Gene Ranking without Selection Bias for the Predictive Classification of Microarray Data. *BMC Bioinformatics*, 54.

Guyon, I., Weston, J., Barnhill, S., & Vapnik, V. (2002). Gene selection for cancer classification using support vector machines. *Machine Learning*, *46*, 389–422.

Jain, A. K., Duin, R. P. W., & Mao, J. (2000). Statistical pattern recognition: A review. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *22*, 4–37.

Lal, T. N., Schroder, M., Hinterberger, T., Weston, J., Bogdan, M., Birbaumer, N., & Scholkopf, B. (2004). Support vector channel selection in BCI. *IEEE Transactions on Biomedical Engineering*, *51*, 1003–1010.

Shoeb, A. H. (2003). Patient-specific seizure onset detection. Master's thesis, Massachusetts Institute of Technology.

Dynamic Variable Comparability Analysis for C and C++ Programs

Philip J. Guo PGBOVINE@CSAIL.MIT.EDU
Stephen McCamant SMCC@CSAIL.MIT.EDU
MIT Computer Science and Artificial Intelligence Laboratory, 32 Vassar Street, Cambridge MA, 02139 USA

1. Introduction

Languages like C and C++ provide programmers with only a few basic types (e.g., `int`, `float`). Programmers often use these types to hold semantically unrelated values, so types typically capture only a portion of the programmer's intent. For example, a programmer may use the `int` type to represent array indices, sensor measurements, the current time, or other unrelated quantities. `pair<int,int>` can represent the coordinates of a point, a quotient and remainder returned from a division procedure, etc. The use of a single programming language representation type for these conceptually distinct values obscures the differences among the values.

```
int main() {
  int year = 2005;
  int winterDays = 58;
  int summerDays = 307;
  compute(year, winterDays, summerDays);
  return 0;
}

int compute(int yr, int d1, int d2) {
  if (yr % 4)
    return d1 + d2;
  else
    return d1 + d2 + 1;
}
```

In the program above, the three variables in `main` all have the same type, `int`, but two of them hold related quantities (numbers of days), as can be determined by the fact that they interact when the program adds them, whereas the other contains a conceptually distinct quantity (a year). `day` and `year`, the *abstract types* that the programmer most likely intended to convey in the program, are both represented as `int`.

A variable comparability analysis aims to automatically infer when sets of variables with the same representation type actually belong to the same abstract type. Sets of variables with the same abstract type are said to be *comparable*. This analysis could make the code's intention clearer, prevent errors, ease understanding, and assist automated program analysis tools. In the past, it has been performed statically using type inference, but we propose to perform a dynamic comparability analysis by observing interactions of values at run-time. We believe that a dynamic analysis can yield more precise results with greater scalability, given an execution which provides adequate coverage. We have implemented a tool called DynComp which performs this analysis for C and C++ programs.

2. Application to Invariant Detection

One specific application of a comparability analysis is to improve the performance and results of the Daikon invariant detector (Ernst, 2000). Daikon analyzes program value traces to infer properties that hold over all observed executions. Without comparability information, Daikon attempts to infer invariants over all sets of variables with the same representation type, which is expensive and likely to produce invariants that are not meaningful. For the above example, Daikon may state that `winterDays < year`. While this invariant is true, it is most likely not meaningful because the two variables belong to different abstract types (they are not comparable). Comparability information indicates which pairs of variables should be analyzed for potential invariants, which both improves Daikon's performance and helps it produce more meaningful results.

3. Static Analysis: Lackwit

The Lackwit tool (O'Callahan & Jackson, 1997) performs a static source code analysis on C programs to determine when two variables have the same abstract type. It performs type inference to give two variables the same abstract type if their values may interact at any time during execution via a program operation such as + or =. Because it does not actually execute the program, it must make conservative estimates regarding whether variables may interact, which may lead to imprecise results with fewer abstract types than actually present in the program. Also, though it is sound with respect to a large subset of C, this subset does not cover all the features used in real programs: it may miss interactions that result from some kinds of pointer arithmetic, and it does not track control flow through function pointers.

4. Proposed Dynamic Analysis

We propose a dynamic approach for computing whether two variables are comparable at program points such as procedure entries and exits. The analysis conceptually computes abstract types for values, then converts the information into sets of comparable variables at each program point (called *comparability sets*). It consists of a value analysis which occurs throughout execution and a variable analysis which occurs during each program point.

The value analysis maintains, for each value in memory and registers, a tag representing its abstract type. It associates a fresh abstract type with each new value created during execution. For a primitive representation type such as `int`, new values are instances of literals and values read from a file. Only values of primitive types receive tags; structs and arrays are treated as collections of primitive types. Two values have the same abstract type if they interact by being arguments to the same program operation such as `+` or `=`. This is a transitive notion; in the code `a+b; b+c`, the values of `a` and `c` have the same abstract type. Each program operation on two values unifies their abstract types, using an efficient union-find data structure, and gives the result the same abstract type.

The variable analysis is intended to report, for any pair of variables at a given program point, whether those variables ever held values of the same abstract type at that program point. The abstract type information that is maintained for values must be converted into abstract types for variables each time a program point is executed. In order to accommodate this, the analysis keeps a second variable-based set of abstract type information (independently for each program point) and merges the value-based information into that data structure at each execution of the program point. We are currently experimenting with several algorithms for this operation, each with different degrees of precision versus performance.

4.1 Implementation: DynComp

We have implemented a tool called DynComp for performing dynamic comparability analysis of C and C++ programs. It is built upon a framework based on dynamic binary instrumentation using Valgrind (Nethercote & Seward, 2003). It maintains a numeric tag along with each byte of memory and each register which represents the abstract type of the value stored in that location. The value analysis is performed by instrumenting every machine instruction in which values interact to unify their tags in the union-find data structure. The variable analysis is performed by pausing the program's normal execution during program points, reading the tags of the values held by relevant variables, and translating the abstract types represented by these tags to the abstract types of the variables.

4.2 Advantages of Our Dynamic Approach

Our dynamic approach has the potential to produce more precise results than static analysis because it need not apply approximations of run-time behavior but can observe actual behavior. Whereas a static tool must infer whether two variables could ever possibly interact and become comparable on any possible execution (usually by making conservative estimates), our dynamic analysis (given a test suite with adequate coverage) can tell exactly whether the two variables are comparable during actual executions.

Furthermore, our use of dynamic binary instrumentation results in a tool that can be more robust than Lackwit's source-based static approach because it only needs to deal with value interactions in memory and registers, which have relatively simple semantics. We do not need to handle complex source code constructs (such as pointer arithmetic, function pointers, or type casts) or analyze the source of or make hand-written summaries for library code (which often includes difficult-to-analyze constructs), requirements which are often difficult to implement robustly.

5. Experimental Results and Future Work

DynComp has been tested to work on moderately-sized C and C++ programs (around 10,000 lines of code). In quantitative evaluations, DynComp usually produces smaller comparability sets than Lackwit and allows Daikon to run faster and generate fewer invariants. In qualitative evaluations, the sets that DynComp produces more closely match programmer-intended *abstract types* because it does not have to make approximations about run-time behavior.

DynComp's scalability is currently limited by the memory overhead of maintaining tags, but we are currently working on garbage collection and various optimizations to overcome this limitation. In the meantime, another member of our research group is working on a Java implementation of dynamic comparability analysis.

References

Ernst, M. D. (2000). *Dynamically discovering likely program invariants*. Doctoral dissertation, University of Washington Department of Computer Science and Engineering, Seattle, Washington.

Nethercote, N., & Seward, J. (2003). Valgrind: A program supervision framework. *Proceedings of the Third Workshop on Runtime Verification*. Boulder, Colorado, USA.

O'Callahan, R., & Jackson, D. (1997). Lackwit: A program understanding tool based on type inference. *Proceedings of the 19th International Conference on Software Engineering* (pp. 338–348). Boston, MA.

A Scalable Mixed-Level Framework for Dynamic Analysis of C/C++ Programs

Philip J. Guo PGBOVINE@CSAIL.MIT.EDU
Stephen McCamant SMCC@CSAIL.MIT.EDU
MIT Computer Science and Artificial Intelligence Laboratory, 32 Vassar Street, Cambridge MA, 02139 USA

1. Introduction

Many kinds of dynamic program analyses, such as run-time data structure repair (Demsky et al., 2004) and invariant detection (Ernst, 2000), are interested in observing the contents of data structures during execution. There is a large number and variety of programs written in C and C++ which make compelling and practical real-world test subjects for these analyses. However, it is often difficult to create robust and scalable implementations of these analyses because the traditionally-preferred approach of source-code modification suffers from many limitations. Its complement, a binary-based approach, is often inadequate as well because it cannot access source-level information about data structures which these analyses require.

To overcome the limitations of previous approaches, we have built a framework based on a *mixed-level* approach which uses both binary and source-level information at run time. Our framework enables the implementation of dynamic analyses that are *easy to use* — working directly on a program's binary without the need to deal with source code, *robust* — avoiding output of garbage data and crashes caused by unsafe memory operations, and *scalable* — operating on programs as large as 1 million lines of code.

We have used this framework to build a value profiling tool named Kvasir, which outputs run-time contents of data structures to a trace file for applications such as invariant detection, and compared it against its source-based predecessor to show the advantages of our mixed-level approach.

2. Limitations of a Source-Based Approach

A source-based approach works by inserting extra statements into the target program's source code so that when the instrumented code is compiled and executed, the program runs normally and outputs the desired data, usually to a trace file for further analysis. Here are some limitations:

1. **Usability**: A source-based tool is often difficult to use on real-world programs because a user must instrument every source file and then compile the instrumented source into a binary. It can be time-consuming to figure out exactly which files to instrument and compile because many real-world programs have their source code spread throughout multiple directories and utilize sophisticated compilation configurations.

2. **Robustness & Scalability**: It is difficult to make a robust and scalable source-based tool for programs written in a memory-unsafe language like C or C++ due to dynamic memory allocation and pointer manipulation. It is impossible to determine run-time properties of dynamically-allocated data at the time the source code is instrumented. For instance, an `int*` pointer may be uninitialized, initialized to point to one integer, or initialized to point to an array of unknown size. In order for a source-based tool to insert in statements to observe the values which this pointer refers to at run time, it must maintain metadata such as initialization state and array size. This can be difficult to correctly implement, as it involves transforming all code related to pointer operations to also update their metadata.

   The complexities of parsing C (and especially C++) code presents another barrier to scalability. The parser must properly account for various dialects and standards (e.g., K&R, ANSI) and complex source-code constructs (e.g., function pointers, C++ templates) which are likely to arise in many real-world programs.

Previous research in our group used a source-based approach to implement the Dfec value profiling tool (Morse, 2002). Our experiences with using and debugging Dfec confirm the aforementioned limitations.

3. Mixed-Level Approach and Framework

When implementing Kvasir, the successor to Dfec, we adopted a new mixed-level approach which combines binary-level instrumentation with source-level constructs extracted from a program's debugging information. The framework we developed can be utilized to build a wide range of dynamic analyses, of which Kvasir is the first. It consists of three parts: dynamic binary instrumentation using Valgrind (Nethercote & Seward, 2003), memory safety checking using the Valgrind Memcheck tool, and data structure traversal using debugging information. These components work together to provide a framework which overcomes limitations of source-based approaches:

1. **Usability**: A user of a tool built upon this framework simply needs to run it on a binary file compiled with debugging information. There are no worries about which source files to instrument or compile. We utilize Valgrind to re-write the target program's binary to insert the appropriate instrumentation instructions at run time. In order to gather source-level information about variable names, types, and locations, we use the debugging information compiled into the binary.

2. **Robustness & Scalability**: In order to ensure that an analysis does not crash the target program or output garbage values, we utilize the Valgrind Memcheck tool to keep run-time metadata about which bytes of memory have been allocated and/or initialized. This metadata is updated during every machine instruction that operates on memory. Memcheck's memory checking is far more robust than the source-based metadata approach because it works directly on the binary level, thus avoiding source-level complexities.

   Furthermore, working on the binary level greatly improves robustness and scalability because the semantics of machine instructions are much simpler than that of source code. The source-level description of data has a complex structure in terms of pointers, arrays, and structures, but the machine-level representation is as a flat memory with load and store operations.

   When source-level information is required, though, instead of gathering it from the source code, we utilize debugging information, which is much easier to parse and more dialect and language-independent. With such type, size, and location information, our framework allows tools to observe the run-time values of variables, arrays, and recursively traverse inside of structs. Memcheck allows tools to discover array sizes at run time and determine which values are initialized, thus preventing the output of garbage values.

4. Evaluation of Kvasir Value Profiling Tool

Kvasir, built upon our mixed-level framework, works by instrumenting the program's binary to pause execution at function entrances and exits. At those times, it traverses through data structures to output their contents to a trace file. It works on many large C and C++ programs such as `povray`, `perl`, `xemacs`, and `gcc`. Kvasir provides selective tracing functionality to control which variables to observe and at which times the observations should be made, which is useful for tracing selected parts of large programs. For example, a CSAIL research project on data structure repair (Demsky et al., 2004) used Kvasir to trace the evolution of flight information data structures in the air traffic control program CTAS (1 million lines of code) and a map in the server for the strategy game Freeciv (50,000 lines).

Kvasir has two main limitations: First, Kvasir only works on x86/Linux programs because Valgrind only supports x86/Linux. However, most open-source C/C++ programs can be compiled for this platform. Secondly, programs running under Kvasir suffer a performance slowdown of around 80 times during normal execution without any output. There is an additional slowdown proportional to the amount of trace data outputted. However, by selecting a small subset of functions and variables to trace, it is possible to run Kvasir on large, interactive GUI programs such as CTAS at reasonable levels of speed and responsiveness.

Kvasir's usability, robustness, and scalability far surpass that of its source-based predecessor Dfec, which could only work without crashing on small to mid-sized programs, and usually only after hours of work massaging the source code so that Dfec could accept it as input. The largest program that Dfec worked on was `flex` (12,000 lines), but only after weeks of effort to modify its source code. In contrast, we can download `flex`, compile it with debugging information, and run Kvasir on it, all within half an hour.

5. Future Work

Both our mixed-level framework and the Kvasir tool are under active development, especially to improve C++ support. If you have a research project that requires a tool to observe or modify data structures within large C or C++ programs at run time and would like to use our framework or the Kvasir tool, please contact us. We plan to add new features based on user interests and requirements.

References

Demsky, B., Ernst, M. D., & Rinard, M. (2004). Automatic inference and enforcement of data structure consistency constraints (in preparation).

Ernst, M. D. (2000). *Dynamically discovering likely program invariants*. Doctoral dissertation, University of Washington Department of Computer Science and Engineering, Seattle, Washington.

Morse, B. (2002). A C/C++ front end for the Daikon dynamic invariant detection system. Master's thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA.

Nethercote, N., & Seward, J. (2003). Valgrind: A program supervision framework. *Proceedings of the Third Workshop on Runtime Verification*. Boulder, Colorado, USA.

A Scalable Architecture for Network Fault Diagnosis in the Knowledge Plane

George J. Lee GJL@MIT.EDU
Peyman Faratin PEYMAN@MIT.EDU
Steven Bauer BAUER@MIT.EDU
MIT Computer Science and Artificial Intelligence Laboratory, 32 Vassar Street, Cambridge MA, 02139 USA

1. Introduction

The Internet enables a wide variety of devices to communicate with one another, but both the distributed administration of the Internet and the complex interactions among failures make problems difficult to diagnose. To address this problem, we propose an architecture for distributed network fault diagnosis that allows data providers, diagnosis providers, and users to exchange information in a standard way. We see this diagnostic architecture as one aspect of the Knowledge Plane (KP)(Clark et al., 2003), a platform which will enable automated network diagnosis, management, configuration, and repair. A general architecture for distributed diagnosis will allow different diagnosis providers, such as intrusion detection systems and domain-specific failure diagnosis systems, to interoperate and exchange data from different sources such as network monitors and Internet tomography systems. Designing such an architecture is difficult because it must be able to support a wide range of data and diagnostic methods as well as scale to large networks with unpredictable network faults potentially affecting millions of users. In this paper we describe an approach for addressing these challenges using an extensible ontology and a scalable routing protocol and present the results of some preliminary experiments.

There has been some related work in networks for exchanging diagnostic information. Wawrzoniak et al. developed Sophia, a system for distributed storage, querying, and processing of data about networks (Wawrzoniak et al., 2004). Thaler and Ravishankar describe an architecture for diagnosing faults using a network of experts (Thaler & Ravishankar, 2004). Gruschke describes how an event management system can use dependency graphs for diagnosis (Gruschke, 1998). Unlike previous work, we consider how to handle large volumes of diagnostic requests by reasoning about the effects of individual network failures.

2. An Agent Architecture for Fault Diagnosis

Our goal is to design a network architecture that allows distributed diagnosis for a wide range of faults. As a starting point, we consider how to diagnose reachability faults in which a user cannot complete a network task because they cannot reach some destination. Reachability faults may result from a variety of causes, including physical network cable disconnection, network misconfiguration, access provider failures, routing failures, and software failures.

In order to automatically collect data about faults, perform diagnosis, and convey diagnoses to users affected by faults, we developed a diagnostic architecture comprising a network of intelligent agents. Each agent may request information from other agents and respond to such requests. In this network architecture, **user agents** make requests for diagnosis, and **diagnosis agents** request data from **data agents** in order to respond to user agent requests.

3. Scalable Routing Using Aggregation

Our architecture must address the issue of scalability. In large networks such as the Internet, a single serious network fault may affect millions of users. To successfully diagnose such faults, our architecture uses aggregation to greatly reduce the number of messages required for diagnosis. Instead of performing a full diagnosis for every request, an agent may determine that multiple requests may be aggregated and answered using existing knowledge.

In order to maximize the effectiveness of aggregation, agents route requests and responses according to the Internet autonomous system (AS) topology. For each AS, there exists a diagnostic agent that knows about failures within that AS. A diagnostic request from a user agent first travels to the diagnostic agent for the user's access provider AS. If the diagnostic agent does not have enough information to respond, then it forwards the request to the agent for the next AS along the AS path towards the unreachable destination. When an agent has enough information to produce a diagnosis, it sends a response that travels along the reverse path of the request, allowing each agent along the path to store the data in the response. Routing along the AS path ensures that if a fault occurs somewhere along the

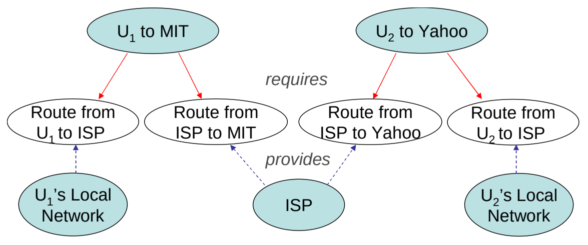


Figure 1. Inferring failures using a dependency graph

AS path, data is collected from every agent with information about a fault while minimizing the average number of requests each agent handles.

In order to aggregate effectively, however, an agent must be able to determine whether it can satisfy multiple requests using existing knowledge. To address this problem, we propose an ontology that allows agents to describe the dependencies between different network components and infer all the network failures that may result from a failed component. This ontology defines **components** and **dependencies**, where components may *require* multiple dependencies. A dependency is satisfied if and only if there exists a component that *provides* that dependency. A component functions if and only if it has not failed and all the dependencies it requires are satisfied. This dependency graph is distributed: each diagnostic agent may define its own local dependency model and incorporate additional dependency information from other agents.

Figure 1 depicts a simple example of how an agent might use a dependency graph for aggregation. The shaded and unshaded ellipses represent components and dependencies, and the solid and dotted lines indicate required and provided dependencies, respectively. To reach their destinations, users U_1 and U_2 must be able to reach their ISP, and their ISP must be able to reach their respective destinations. Since both users share the same ISP, if their ISP fails, then a diagnostic agent can infer that neither user can reach their desired destinations. An actual dependency graph would also model dependencies among many other types of networking components, including DNS, network applications, and link-layer components.

To illustrate the potential benefit of aggregation, we conducted simulations of our routing protocol using an actual AS topology consisting of 8504 nodes using data from Skitter[1]. Figure 2 plots the average number of requests an agent receives based on its distance from the agent of the AS responsible for the failure, where the point "0 hops" represents the responsible agent. In this simulation, the failure affects 1,000,000 users, each of whom makes a diagnostic request. Without aggregation, the responsible agent

[1]http://www.caida.org/tools/measurement/skitter/

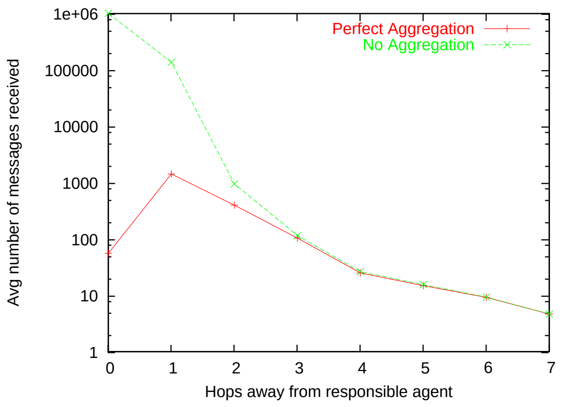


Figure 2. Message Distribution

receives one request from every user agent. With perfect aggregation, if agents can accurately respond to requests using existing knowledge, the agent for the responsible AS only receives at most one request from each of its neighboring agents. The benefit of aggregation diminishes farther away from the responsible agent because the distant agents receive fewer requests and hence have fewer opportunities for aggregation. This plot shows that an effective ontology and intelligent aggregation can greatly reduce the average number of requests agents process.

4. Conclusion and Future Work

In this paper we proposed an architecture for distributed network fault diagnosis that represents data using an extensible ontology and routes requests and responses using aggregation to reduce the average number of requests an agent must process. We are currently developing a prototype of this architecture and plan to conduct more experiments to determine how well it performs under a variety of realistic failure cases.

References

Clark, D. D., Partridge, C., Ramming, J. C., & Wroclawski, J. T. (2003). A knowledge plane for the internet. *Proceedings of SIGCOMM '03*.

Gruschke, B. (1998). Integrated event management: Event correlation using dependency graphs. *Proceedings of DSOM '98*.

Thaler, D. G., & Ravishankar, C. V. (2004). An architecture for inter-domain troubleshooting. *Journal of Network and Systems Management*, *12*.

Wawrzoniak, M., Peterson, L., & Roscoe, T. (2004). Sophia: an information plane for networked systems. *SIGCOMM Comput. Commun. Rev.*, *34*, 15–20.

JCilk's Support for Speculative Computation

I-Ting Angelina Lee ANGELEE@MIT.EDU
MIT Computer Science and Artificial Intelligence Laboratory, 32 Vassar Street, Cambridge MA, 02139 USA

1. Introduction

JCilk is a Java-based multithreaded language that extends the semantics of Java [Gosling et al., 2000] by introducing "Cilk-like" [Supercomputing, 2001, Frigo et al., 1998] parallel programming primitives. JCilk supplies Java with procedure-call semantics for concurrent subcomputations and, more importantly, integrates exception handling with multithreading by defining semantics consistent with Java's existing semantics of exception handling.

JCilk's strategy of integrating multithreading with Java's exception semantics yields some surprising semantic synergies. In particular, JCilk supports an implicit abort mechanism as part of JCilk's exception semantics: when multiple subcomputations are executed in parallel, an exception thrown by one subcomputation signals its sibling computations to abort. This implicit abort yields a clean semantics in which only a single exception from the enclosing `try` block is handled.

In ordinary Java, an exception causes a nonlocal transfer of control to the `catch` clause of the nearest dynamically enclosing `try` statement whose `catch` clause handles the exception. The *Java Language Specification* [Gosling et al., 2000, pp. 219–220] states,

> "During the process of throwing an exception, the Java virtual machine abruptly completes, one by one, any expressions, statements, method and constructor invocations, initializers, and field initialization expressions that have begun but not completed execution in the current thread. This process continues until a handler is found that indicates that it handles that particular exception by naming the class of the exception or a superclass of the class of the exception."

In JCilk, we have striven to preserve these semantics while extending them to cope gracefully with the parallelism provided by the Cilk primitives. Specifically, JCilk extends the notion of "abruptly completes" to encompass the implicit aborting of any side computations that have been spawned off and on which the "abrupt completion" semantics of the Java exception-handling mechanism depends.

This implicit abort mechanism allows speculative computation, which is required by some parallel search algorithms, such as branch-and-bound and heuristic search [Dailey & Leiserson, 2002]. In these algorithms, some computations may be spawned off speculatively, but are later found to be unnecessary. In such cases, one wishes to terminate these extraneous computations as soon as possible so that they do not consume system resources.

It turns out that the exception-based abort mechanism provided by JCilk is cleaner to code with than the inlet-based one provided by Cilk. Specifically, JCilk extends Java's exception semantics to allow exceptions to be passed from a spawned method to its parent in a natural way so that the need for Cilk's aborting constructs, `inlet` and `abort`, is obviated.

This paper presents how JCilk supports speculative computation with its exception-based abort mechanism. The paper first summarizes the basic syntax and semantics in JCilk, and then gives a motivational example of how the Queens puzzle with speculative computation can be implemented in JCilk.

The work described in this paper represents joint research with John S. Danaher and Charles E. Leiserson.

2. The JCilk language

The philosophy behind our JCilk extension to Java follows that of the Cilk extension to C: the multithreaded language should be a true semantic parallel extension of the base language. JCilk extends Java[1] by adding new keywords that allow the program to execute in parallel. If the JCilk keywords for parallel control are elided from a JCilk program, however, a syntactically correct Java program results, which we call the ***serial elision*** of the JCilk program. JCilk is a ***faithful*** extension of Java, because the serial elision of a JCilk program is a correct (but not necessarily sole) interpretation of the program's parallel semantics.

To be specific, JCilk introduces three new keywords —

[1]Actually, JCilk extends the serial portion of the Java language, and it omits entirely Java's multithreaded support as provided by the `Thread` class.

```
 1   cilk int f1() {
 2      int w, x, y, z;
 3      cilk try {
 4         w = spawn A();
 5         x = spawn B();
 6      } catch(Exception e) {
 7         w = x = 0;
 8      }
 9      y = spawn C();
10      z = D();
11      sync;
12      return w + x + y + z;
13   }
```

Figure 1: A simple JCilk program.

cilk, spawn, and sync — which are the same keywords used to extend C into Cilk, and they have essentially the same meaning in JCilk as they do in Cilk.

Analogous to Cilk, in JCilk the keyword cilk is used as a method modifier to declare the method to be a ***cilk method***, which is analogous to a regular Java method except that it can be spawned off to execute in parallel. When a parent method spawns a child method, which is accomplished by preceding the method call with the spawn keyword, the parent can continue to execute in parallel with its spawned child. The sync keyword acts as a local barrier. The JCilk runtime system ensures that program control cannot go beyond a sync statement until all previously spawned children have terminated. In general, until a cilk method executes a sync statement, it cannot safely use results returned by previously spawned children.

Different from Cilk, however, in JCilk the cilk keyword can also be used as a modifier for a try statement. JCilk enforces the constraint that spawn and sync keywords can only be used within a cilk try block, but not within an ordinary try block. Placing spawn or sync keywords within a catch or finally clause is illegal in JCilk, whether the catch or finally clause belongs to a cilk try statement or to an ordinary try statement. The reason try blocks containing spawn and sync must be declared cilk is that when an exception occurs, these try statements may contain multiple threads of control during exception handling. Although the JCilk compiler could detect and automatically insert a cilk keyword before a try statement containing spawn or sync, we feel the programmer should be explicitly aware of the inherent parallelism. We disallow spawn and sync within catch or finally clauses for implementation simplicity, but we might consider revisiting this decision if a need arises.

To illustrate how we have introduced these Cilk primitives into Java, first consider the JCilk program in Figure 1 and assume that no exception happens during execution. The method f1 spawns off methods A and B to run in parallel in lines 4 and 5. After spawning A and B, the execution of f1 continues to spawn off C in parallel in line 9 and call method D normally in line 10. Then the execution waits at the sync in line 11 until all the subcomputations A, B, and C have completed. When they all complete, assuming that no exception occurs, f1 computes the sum of their returned values as its returned value in line 12.

What happens, however, when an exception occurs during execution? When an exception is thrown, JCilk "semisynchronously" aborts the ***side computations*** of the exception, which include any method that is dynamically enclosed by the catch clause of the cilk try statement that handles the exception. Consider the same program in Figure 1 again and assume that an exceptions occurs during execution. If A throws an exception after f1 has spawned B, B and all its subcomputations are automatically aborted. Similarly, if B throws an exception while A is still executing, A and all its subcomputations are automatically aborted. In either cases, once all computations dynamically enclosed by the catching cilk try have terminated (either by throwing exception of by aborting), the corresponding catch clause or finally clause (if any) is then executed. On the other hand, if C throws an exception, which is not caught anywhere within method f1, all A, B, and their subcomputations are aborted, and the exception is propagated up to f1's parent, as the reason why f1 is terminated.

This implicit abort is performed ***semisynchronously***, that is, when a computation is aborted, the abort happens only after each spawn, sync, or cilk try statement. JCilk's semantics for semisynchronous aborts simplify the reasoning about program behavior when an abort occurs, limiting the reasoning to those points where parallel control must be understood anyway. In addition, JCilk provides for aborts themselves to be caught by defining a new subclass of Throwable, called CilkAbort, thereby allowing programmers to clean up an aborted subcomputation.

3. The Queens problem

Figure 2 illustrates how the so-called Queens puzzle can be programmed in JCilk. The program would be an ordinary Java program if the three keywords cilk, spawn, and sync were elided, but the JCilk semantics make this code a highly parallel program. This JCilk program serves as an example of how speculative applications can be coded in JCilk in a relatively simple fashion compared to in Cilk or in Java.

This implementation uses a speculative parallel search and exploits JCilk's implicit abort semantics to avoid extraneous computation. The program spawns many branches in the hopes of finding a safe configuration of the n queens. When such a configuration is discovered, some outstand-

```
 1  public class Queens {
 2    private int n;
      :
 3    private cilk void
 4    q(int[] cfg, int r) throws Result {
 5      if(r == n) {
 6        throw new Result(cfg);
 7      }

 8      for(int c=0; c<n; c++) {
 9        int[] ncfg = new int[n];
10        System.arraycopy(cfg, 0,
11                          ncfg, 0, n);
12        ncfg[r] = c;

13        if(safe(r, c, ncfg)) {
14          spawn q(ncfg, r+1);
15        }
16      }
17      sync;
18    }

19    public static cilk void
20    main(String argv[]) {
        :
21      int n = Integer.parseInt(argv[0]);
22      int[] cfg = new int[n];
23      int[] ans = null;

24      cilk try {
25        spawn (new Queens(n)).q(cfg, 0);
26      } catch(Result e) {
27         ans = (int[]) e.getValue();
28      }
29      sync;

30      if(ans != null) {
31        System.out.print("Solution: ");
32        for(int i = 0; i < n; i++) {
33          System.out.print(ans[i] + " ");
34        }
35        System.out.print("\n");
36      }
        ⋮
37    }
38  }
```

Figure 2: The Queens problem coded in JCilk. The program searches in parallel for a single solution to the problem of placing `n` queens on an `n`-by-`n` chessboard so that none attacks another. The search quits when any of its parallel branches finds a safe placement. The method `safe` determines whether it is possible to place a new queen on the board in a particular square. The `Result` exception (which inherits from class `Exception`) is used to notify the `main` method when a result is found.

ing `q` methods might still be executing; those subsearches are now redundant and should be aborted. JCilk's exception mechanism facilitates programming this strategy: an exception thrown by a computation automatically aborts all sibling computations and their children dynamically enclosed in the catching `cilk try` block. In this example, the branch that found the legal placement throws the `Result` exception, which propagates all the way up to the `main` method and aborts all outstanding `q` methods automatically. A `sync` statement (line 29 in the `main` method) is presented before it proceeds to print out the solution to ensure that all side computations have terminated.

4. Project progress

We have developed a solid set of semantics that extends Java's exception mechanism to handle gracefully the parallelism introduced by `spawn` and `sync`. We have also built a working version of the JCilk system, including both the compiler and the runtime system, which supports the semantics that we have designed. The system implementation serves as an important foundation for JCilk's semantic design. Only after implementing the actual system, can we be certain that our semantic design is "reasonable" and not just some semantics on paper.

Several applications are developed in JCilk to demonstrate the expressive of JCilk's features, including Queen's puzzle, parallel alpha-beta search, and LU Decomposition. The next goal of the project is to improve the system performance and to integrate JCilk's threading model with other modern language features in Java.

References

Dailey, D., & Leiserson, C. E. (2002). Using Cilk to write multiprocessor chess programs. *The Journal of the International Computer Chess Association*.

Frigo, M., Leiserson, C. E., & Randall, K. H. (1998). The implementation of the Cilk-5 multithreaded language. *Proceedings of the ACM SIGPLAN '98 Conference on Programming Language Design and Implementation* (pp. 212–223). Montreal, Quebec, Canada. Proceedings published ACM SIGPLAN Notices, Vol. 33, No. 5, May, 1998.

Gosling, J., Joy, B., Steele, G., & Bracha, G. (2000). *The Java language specification second edition*. Boston, Massachusetts: Addison-Wesley.

Supercomputing (2001). *Cilk 5.3.2 reference manual*. Supercomputing Technologies Group,, MIT Laboratory for Computer Science, 545 Technology Square, Cambridge, Massachusetts 02139.

The Effect of Neighborhood Boundaries on Nearness

Gary Look GARYL@CSAIL.MIT.EDU
MIT Computer Science and Artificial Intelligence Laboratory, 32 Vassar Street, Cambridge MA, 02139 USA

1. Motivation

The notion of "nearness" is a key feature used by location-based services, yet it has not received a rigorous treatment in the ubiquitous computing (ubicomp) literature. Usually, location-based services such as MapQuest address the nearness issue by assuming a fixed radius around a location, and treating all places within that radius as near. However, personal experience suggests that each person has their own perception of what places are "near," and that, when mapped out, these places would form a shape that is very different from a circle.

For example, one of the main entrances to the MIT campus is located at 77 Massachusetts Avenue (Mass Ave). My barber is located in Cambridge's Central Square neighborhood, 0.8 miles from 77 Mass Ave. From 77 Mass Ave, I consider my barber near enough that I wouldn't mind walking to his shop. On the other hand, there is a Virgin Music store located in the other direction along Mass Ave, across the Harvard Bridge in Boston. Yet, even though I know this store is also 0.8 miles away from 77 Mass Ave, I do not consider it near enough for me to walk to it.

This research investigates what factors influence a person's perception of nearness. Specifically, I am looking at how nearness is influenced by a person's familiarity with a neighborhood and shaped by the boundaries that define that neighborhood. My goal is to incorporate these factors into a model of an individual's sense of nearness. My claim is that the features that are used as neighborhood boundaries — topography, major streets, demographics — are all identifiable and can be used to objectively explain individual differences in the seemingly subjective assessment of nearness. Moreover, I claim this holds as the scale space under consideration changes.

2. Approach

My research is divided into two major sub-areas. The first area is identifying where different neighborhoods are located and what features of the urban landscape people use to define neighborhood boundaries. I am particularly interested in how local neighborhoods, those without formally dictated boundaries (for example, Boston's Back Bay, or Cambridge's Harvard Square) are defined. The second area of work is to identify which of these features shape the area that a person considers to be nearby. I have just started this line of research, and in this abstract, I will focus the discussion on the second area.

As illustrated in the anecdotal example at the start of this abstract, I believe that nearness is in large part shaped by the number and type of boundaries between two places. A place P that is x miles from reference place R and in the same neighborhood as R may be considered near R. However, another place P' that is also x miles from R but in another neighborhood may not seem near due to the number of boundaries separating P' from R. However, nearness is also influenced by the places we frequent, and this would reduce the cumulative effect of boundary crossings on our perception of nearness and thus explain individual differences in the nearness perception.

As a first step in this research, I conducted a study in which I asked people to imagine they are at different subway stops. Then, without the aid of a map, I presented a list of places and asked "If a person comes up to you on foot and asks you which of these places are near the T stop where you are, how would you respond?" Figure 1 shows a map of one subway stop and some of the places on the list for that subway stop. After participants completed the survey, I then presented them a map of the Boston-Cambridge area and asked them to sketch the boundaries to a number of neighborhoods. The goal of this survey is to provide insight into a number of questions:

- Is there a consistent set of features that both define neighborhood boundaries and influence a person's perception of nearness? Do neighborhood boundaries that are less distinct (such as the boundary between Central and Harvard Squares) have a different effect on our sense of nearness than those boundaries that are more distinct?
- For a given individual, can the criteria used to define what is near one particular location be used to determine what that person would consider to be near another location? Can this criteria be generalized across individuals?

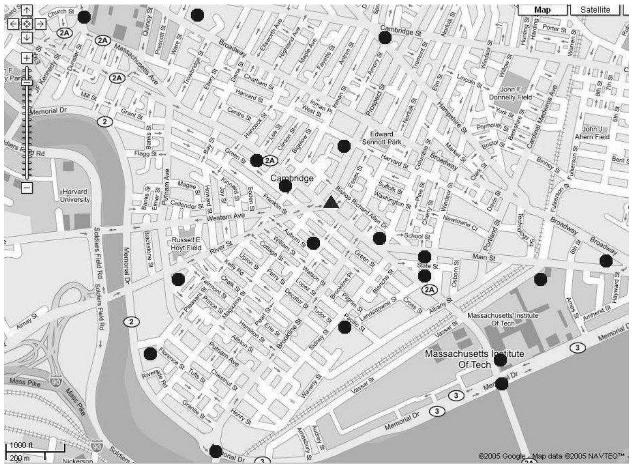


Figure 1. The triangle in the center of this map is the location of the Central T stop. Each of the circles mark a place for which I ask "Is this place near the Central T stop?" This is for the reader's visualization only; study participants were not provided a map.

- How does a person's perception of nearness change from place to place? Does the shape and size of the region considered near remain constant, or does it change? If we centered the nearness region around the reference subway stop, is the region symmetrical? What would explain asymmetries?

I am currently analyzing the data from this study. The first part of my analysis compares how well different models predict which places a person considers to be near a particular subway stop. The models I am considering are:

1. "As the crow flies" models, that consider all places within a certain distance near
2. "Actual distance" models, that predict nearness based on the actual length of the route between two places
3. Individual-defined, neighborhood-based models that use the neighborhood boundary sketches people drew
4. Street-defined, neighborhood-based models that use areas enclosed by major streets

To measure goodness-of-fit of these different models, I will compare the recall and precision of each of these models on the survey data I collected. Recall would be the fraction of all near places (as defined by each individual's survey results) a model is able to identify. Precision is the ratio of correctly identified near places (again, as defined by each individual) to all places the model classified as near.

3. Related Work

Denofsky proposed three different thresholds that could be applied when determining if two items are near one another (Denofsky, 1976). These thresholds are an absolute range threshold, that uses the maximum possible distance between two items; an object size threshold that takes into account the largest dimensions of the items under consideration; and a "standard" threshold that looks at the distance between "adjacent" pairs of items in the same area as the pair of items under consideration. Like Denofsky, I treat nearness as a context-sensitive metric, but instead of looking at relative distance as the influencing factor, I use a person's familiarity with a region and the boundaries between regions as features that influence the sense of nearness.

The neighborhood boundary sketching part of my study is patterned after Montello's study of vague spatial referents (Montello et al., 2003). This study investigated various behavioral methods for identifying informally defined areas such as "downtown." In addition to developing various methodologies for determining the referents of such areas, Montello also found that some regions had a defining core location that was not necessarily located in the centroid of the region. Part of my future work will look into programmatically identifying the boundaries to neighborhoods like these and the core locations around which they are built.

4. Contribution

Despite the various computing infrastructures available for providing accurate and precise location information, ubicomp applications still need to account for a person's perceived notion of nearness. Although these notions may be incorrect or based on incomplete information, having applications simply provide an objective ground truth with regards to distance is not sufficient. In order to provide personalized location-based recommendations, ubicomp applications need to understand the reasoning process a person uses to determine if something is nearby. Understanding this process allows us to do two things. First, it improves the relevance of suggestions produced by location-based applications. Second, by understanding the process, it makes it easier for applications to correct people's spatial misconceptions.

References

Denofsky, M. E. (1976). How near is near? Master's thesis, Massachesetts Institute of Technology. MIT AI/TR 344.

Montello, D. R., Goodchild, M. F., Gottsegen, J., & Fohl, P. (2003). Where's downtown?: Behavioral methods for determining referents of vague spatial queries. *Spatial Cognition and Computation*, *3*, 185–204.

Hypernyms as Answer Types

Gregory Marton GREMIO@CSAIL.MIT.EDU
Stefanie Tellex STEFIE10@ALUM.MIT.EDU
Aaron Fernandes ADFERNAN@ALUM.MIT.EDU
Boris Katz BORIS@CSAIL.MIT.EDU
MIT Computer Science and Artificial Intelligence Laboratory, 32 Vassar Street, Cambridge MA, 02139 USA

1. Introduction

A system that answers questions automatically with just the right information will return answers of the correct type. The question itself often specifies the answer type it expects. When answering a question like "*What flowers did Van Gogh paint?*", we prefer answers that are members of the class of flowers, e.g. tulips, begonias, or sunflowers, over other things Van Gogh might paint, like "watercolors".

The answer type is also a poor search term: for the question "What countries produce coffee?", answers may be of the form "Brazil produces coffee", far from any mention of the term "country".

2. Related Work

The first Answer Type Identification systems used named entity categories to require, for example, `person` or `organization` types for "Who" questions, and `date` or `time` types for "When" questions (Hirschman & Gaizauskas, 2001).

Today's top systems use answer type taxonomies with hundreds to thousands of entries (Voorhees, 2004; Hovy et al., 2001; Hovy et al., 2002; Katz et al., 2003; Katz et al., 2004), allowing systems to answer questions that, for example, start with "Which countries" or "What president".

Separately, others have explored automatic identification of hypernym–hyponym (henceforth "class"–"member") relations in large bodies of text (Hearst, 1992; Carballo, 1999; Gruenstein, 2001; Snow et al., ; Pantel, 2005). Phrases in English like "corn, wheat, and other staple crops" identify corn and wheat as members of the class of "staple crops". For the most part, these techniques have been used to augment WordNet, a machine–readable lexicon of English that encodes, among other things, these class–member relations.

Harabagiu *et al.* (Pasca & Harabagiu, 2001) observed that answer types can be identified with WordNet classes, and their members can be used as a set of possible answers. Fleischman (Fleischman et al., 2003) used automatic class identification to answer "Who is *P*?" with the automatically extracted classes of which *P* is a member. We are the first to put together these two ideas: to automatically identify classes in the same text that we will answer questions on, and to use those classes as answer types. The class information we extract also turns out to be instrumental in a number of related question–answering tasks.

3. Approach

We build on the observation that classes are answer types and their members are good candidate answers, and we use automatic class identification techniques to identify over a million classes and their members. These enable us to accurately answer questions with much more specific answer types than those previously available. Examples include: "Which non–OPEC countries ..." and "Which former Yugoslav President ... ".

3.1 Candidate Classes

Aaron Fernandes extended (Fernandes, 2004) Fleischman *et al.*'s work (Fleischman et al., 2003) on finding definitions from applying only to person names, to applying to most noun phrases. He generated candidate class–member pairs like those in Figure 1.

3.2 Aggregating Classes

Marton and Tellex then aggregated these class–member candidate pairs π, using a probability of correctness $\mathrm{Freq}(\pi)$ based on the number of times a pair occurred in the corpus, and on the precision $p(z)$ of each pattern z: [1]

$$\mathrm{Freq}(\pi) = \frac{\sum_z p(z) * \mathrm{count}(\pi)}{\sum_z (p(z) * \sum_i \mathrm{count}(i))}$$

[1]The numerator is an expectation of times π was correctly seen; i in the denominator iterates over all pairs observed with the pattern z, making the denominator a normalizing constant. $p(z)$ was estimated for each z from at least 100 examples.

Pattern cue	$p(z)$	**Frequency**	**Example**
common noun then name	0.75	2,125,812	*President* `Clinton`
apposition marked by commas	0.89	625,962	`Noemi Sanin`, *a former foreign minister,*
plural then like	0.42	158,167	*immunisable diseases* like `polio`.
such as or such *X* as	0.47	118,684	*stinging insects*, such as `bees`, `wasps`, `hornets` and `red ants`,
called or also called	0.70	14,066	*a game* called `Tightrope walker`
named then proper name	0.64	8,992	*an Armenian* named `Wilhelm Vigen`
known as, also known as	0.68	8,199	*low-tariff trade rights* known as `most-favored-nation status`

Figure 1. Examples of the seven patterns expressing class–member relations. Precision (*p(z)*) and Frequency of each pattern are shown for a million–article body of newspaper text, along with an example *class* and `member` in context.

4. Progress

We have collected class–member pairs from the AQUAINT corpus[2] of English newspaper text using the patterns described above, yielding 2.3 million candidate categories. Precision, measured using human correctness judgements, is around 50%. We measured recall using the NIST TREC 2003 and 2004 Question Answering Track[3] data. The first form of recall that we measured tested the coverage of categories: of the focus–phrases of questions asked ("Which *flowers* did Van Gogh paint"), around 91% were candidate categories in our list (95% for 2005). The second form of recall tested the coverage of members: for each question, about 30% of the known answers were members of a class associated with the question's focus phrase.[4]

5. Future Work

We will integrate this work with our question–answering system in a number of ways:

- If the answer type in a question (between the *Wh*–word and the verb) matches a class, then we will use members of that class as search terms, and we will prefer those members as answers to the question.

- Categories may appear elsewhere in the question, as in the question: "Which *oil companies* drill in *non–OPEC countries*?". When we identify such categories, we will again use members as search terms.

- In a conversation, a question might refer to a contextual topic by one of its classes. For example, if "Conde Nast" is under discussion, one must know that it is a publishing company in order to understand the question: "Who is *the publishing company*'s CEO?"

Automatic Hypernym Extraction and Automatic Question Answering have been areas of intense study since the 1990s. Question answering is most often applied to a particular body of text. This work lets an automatic system read that text to learn the class information it needs to answer questions about the knowledge within.

References

Carballo, S. (1999). Automatic construction of a hypernym-labeled noun hierarchy from text. *ACL1999*.

Fernandes, A. (2004). Answering definitional questions before they are asked. Master's thesis, MIT.

Fleischman, M., Hovy, E., & Echihabi, A. (2003). Offline strategies for online question answering: Answering questions before they are asked. *ACL2003*.

Gruenstein, A. (2001). Learning hypernyms from corpora.

Hearst, M. (1992). Automatic acquisition of hyponyms from large text corpora. *14th COLING conference*.

Hirschman, L., & Gaizauskas, R. (2001). Natural language question answering: The view from here. *Natural Language Engineering*.

Hovy, E., Hermjakob, U., & Lin, C.-Y. (2001). The use of external knowledge in factoid QA. *TREC-10*.

Hovy, E., Hermjakob, U., & Ravichandran, D. (2002). A question/answer typology with surface text patterns. *HLT2002*.

Katz, B., et al. (2003). Integrating web-based and corpus-based techniques for question answering. *TREC-12*.

Katz, B., et al. (2004). Answering multiple questions on a topic from heterogeneous resources. *TREC-13*.

Pantel, P. (2005). Inducing ontological co-occurrence vectors. *ACL2003*.

Pasca, M., & Harabagiu, S. (2001). The informative role of WordNet in open-domain question answering. *NAACL 2001 Workshop on WordNet and Other Lexical Resources* (pp. 138–143).

Snow, R., Jurafsky, D., & Ng, A. Y. Learning syntactic patterns for automatic hypernym discovery. *NIPS2004*.

Voorhees, E. (2004). Overview of the TREC 2004 question answering track.

[2]http://www.ldc.upenn.edu/Catalog/CatalogEntry.jsp?catalogId=LDC2002T31

[3]http://trec.nist.gov/

[4]Answers for 2005 will not become available until November.

English-to-English Statistical Machine Translation: Why and How

Ali Mohammad ALAWI@CSAIL.MIT.EDU
Federico Mora FEDERICO@CSAIL.MIT.EDU
MIT Computer Science and Artificial Intelligence Laboratory, 32 Vassar Street, Cambridge MA, 02139 USA

1. Introduction

Machine translation, the task of generating tools to translate or to help translate text from one natural language into another by means of a computerized system, has been the subject of intense research over the past five decades. It is one of the earliest proposed uses for the computer and, to date, one of the most dramatic failures of the field. From the beginning, computer scientists promised imminent perfect translation and have been consequently punished by grant committees for their failure to deliver (J. Pierce, 1966). In the past, techniques in machine translation followed the most popular techniques in artificial intelligence; it is not surprising, therefore, that for some time machine translation efforts were directed primarily in the construction of monolithic rule-based systems (Jordan & Benoit, 1999; D. J. Arnold, 1994). This effort in the research community continued until the turn-of-the-century, when it was abandoned with the advent of successful machine learning methods based on simple statistical models and large training sets.

The success of simple statistical methods over intricate and massive rule-based systems is ultimately due to our inability to plan for all natural language text in a set of rules. Natural language is an imprecise method of communication; it is ambiguous and largely unstructured, and is therefore the most expressive form of communication extant. Any set of rules tends to be fragile; a small set can usually yield surprisingly good results, but improvement beyond that point is suprisingly difficult (almost impossible). This is our understanding for why statistical methods are more successful in practice.

Despite this, statistical methods have their limitations. To date, state-of-the-art statistical methods have very little linguistic motivation, and, although they are able to handle complex sentences in a more robust (albeit dissatisfactory) fashion, they often make very common errors that could be fixed by very simple linguistic rules if a way could only be found to introduce such rules without introducing the notorious fragility that accompanies them; this is one popular avenue of current research. Rule-based methods, on the other hand, are generally quite heavily linguistically motivated.

Although further development of rule-based systems is considered a misplaced effort in the academic community, commercial systems are still largely based on this paradigm and are in widespread use in the industry. The poor performance of commercial machine translation systems is now a contemptuous by-word of popular culture. At the same time, quite a lot of literature exists due to the more than forty years of concentrated research in this direction. It is natural to ask, then, if some technique might be found to combine the sophisticated linguistic analysis of the rule-based systems with the robustness of the statistical systems. This is the topic of this paper.

2. Methods and Madness

Statistical machine translation systems are based on the following idea (we will make the canonical assumption that we wish to translate French sentences into English): let us assume that, when French people speak French, they are thinking of an English sentence; they perform some random, noisy procedure on the English, and spit out their French sentence. Our goal is to recover the English sentence. So, there is some distribution $P(e|f)$ and we wish find the maximum-likelihood hidden English sentence e given an observed French sentence f. We use Bayes' rule and arrive at the following:

$$\begin{aligned}\arg\max_e P(e|f) &= \arg\max_e \frac{P(f|e)P(e)}{P(f)} \\ &= \arg\max_e P(f|e)P(e),\end{aligned}$$

where we can omit the $P(f)$ since f is fixed. We call $P(f|e)$ the *translation model* and $P(e)$ the *language model*. It is this formulation that allows us to use very simple translation models and still achieve decent translations. This is very intuitive: if one wishes to translate French to English, it is better if they are native speakers of English and know a little French than if they are native speakers of French and know a little English.

There has been much work in developing both good translation models and good language models. The most sophisticated language models, in practice, never do much better than a simple trigram model, where one assumes that English is a Markov process with a history of two words. This is a recurring theme in Natural Language Processing: it's really easy to get a very competitive baseline that is nowhere close to perfect or even useful in practice.

3. Our Technique

Our technique is very simple and intuitive. We will start with the normal input to a statistical model: a large set of corresponding sentences in French and English, called a *parallel corpus*. We translate the French sentences into English using our rule-based system, then train our statistical system on the new parallel corpus of machine-generated vs human-generated English. Our translation model need not be terribly sophisticated here, obviously, as we are translating within one language and expect little reordering to be successful. It is our language model that will do the work here: fixing minor grammatical errors by considering word movement, insertion, and deletion against its "knowledge" of the English language.

4. Results

We tested our technique on the German-English EUROPARL corpus (Kohn, 2003), a collection human-generated parallel text from the European Parliament. Our rule-based system is the popular SYSTRAN, used without permission via google (Google, 2005). The statistical machine translation technique was IBM Model 2 (P. Brown, 1993) with one-dimensional gaussian alignments (Mohammad & Collins, 2005). The results are shown in Table 1.

Technique	BLEU Score
SYSTRAN	0.13
IBM2+1dG	0.20
IBM2+1dG + SYSTRAN	0.19

Table 1. Results from the individual and combined systems. Note that google does very poorly compared to the statistical model; this is almost certainly due to the general domain intended use of google and the (complex, but still) domain-specific training of the statistical models.

The vast success of the two statistical models over the rule-based model is most likely due to the fact that SYSTRAN's system is built with no domain-specific knowledge whatsoever, whereas the entirety of both statistical systems' knowledge of language is based on the EUROPARL corpus. Though the European Parliament is the medium of discussion in a wide array of domains and the origin of a variety of complex sentence structures, it is still a restricted domain and this is an advantage.

5. Future Work

Certainly a promising avenue of future research is in language models improved beyond the trigram model, particularly models that incorporate long-range measures of grammaticality. This could be used, perhaps to improve English produced by those who are not strong in the ways of the pen.

Furthermore, since we believe that the improvement was due primarily to the language model, we can also hope that a highly simplified, static translation model could replace the IBM Model 2, in effect producing better translations in restricted domains *without a parallel corpus* (whence we could not make use of statistical systems).

An interesting possible application of this kind of translation is in reproducing language evolution to, say, recast *Hamlet* in modern English. It is certainly possible that a language's development could be modeled by simply developing a sequence of language models building up to the current version and by allowing limited word movement and word changes even without a parallel corpus.

6. Conclusions

We have demonstrated a way of combining a general-purpose rule-based system with a simple statistical model to garner robustness and significant improvements in translation quality in restricted domains.

References

D. J. Arnold, *et al*. (1994). *Machine translation: An introductory guide*. Blackwells-NCC, London.

Google (2005). Google language tools. *http://google.com/*.

J. Pierce, *et al*. (1966). *Automatic language processing advisory committee*.

Jordan, B. D. P., & Benoit, J. (1999). A suvey of current paradigms in machine translation. *Advances in Computers*, *49*, 1–68.

Kohn, P. (2003). European parliament corpus. *http://www.statmt.org/europarl/*.

Mohammad, A., & Collins, M. J. (2005). Gaussian alignments in statistical translation models. *Thesis*.

P. Brown, *et al*. (1993). The mathematics of statistical machine translation: Parameter estimation. *Association for Computational Linguistics*, 263–311.

Using Structured, Knowledge-Rich Corpora in Question Answering

Federico Mora FEDERICO@CSAIL.MIT.EDU
Jesse Louis-Rosenberg JESSEL@CSAIL.MIT.EDU
Gregory Marton GREMIO@CSAIL.MIT.EDU
Boris Katz BORIS@CSAIL.MIT.EDU
MIT Computer Science and Artificial Intelligence Laboratory, 32 Vassar Street, Cambridge MA, 02139 USA

1. Introduction

"Question answering (QA) is a type of information retrieval. Given a collection of documents (such as the World Wide Web or a local collection) the system should be able to retrieve answers to questions posed in natural language. QA is regarded as requiring more complex natural language processing techniques than other types of information retrieval such as document retrieval, and it is sometimes regarded as the next step beyond search engines" (Wikipedia, 2005).

To answer domain-independent questions precisely, one requires an extensive and accurate corpus from which to extract answers. These two aspects are often at odds with each other: large corpora (like the Web) often contain much incorrect data, while smaller corpora such as encyclopedias and dictionaries are generally too limited in scope.

Wikipedia, an online encyclopedia created by volunteers across the web, provides a well-structured, wide-coverage corpus available for free. It contains over 600,000 articles, which are updated daily. Because it provides such a large domain of knowledge-rich articles, it serves as a excellent corpus for question answering.

2. Approach

We incorporate Wikipedia as a source in our question answering system for the TREC 2005 QA track using various orthogonal methods (Katz et al., 2004). The TREC 2005 QA track divides questions into three categories: list, definition and factoid. These questions are clustered into small groups, each of which is about the same topic (Voorhees, 2004). The answers to these questions must be found with AQUAINT, a corpus of newspaper articles from 1998 to 2000. We integrated Wikipedia into our list and definition answering system. Our factoid system was unaltered, but there has been previous work using Wikipedia for factoid questions (Ahn et al., 2004).

2.1 Finding a Wikipedia Article

The first step in employing the Wikipedia for a question is finding the relevant article for a topic. Topics in previous years were restricted to simple noun phrases, and over 90% of them appeared in the Wikipedia in some form. Of the 75 topics this year, 10 were "event" noun phrases like "1998 Nagano Olympics" and 4 were headline-like events, like "Liberty Bell 7 space capsule recovered from ocean". Fewer of these appear in the Wikipedia. We found the correct Wikipedia article for 87% of the noun phrase topics, for 70% of the noun phrase event topics, and none of the headline topics — 81% accuracy overall.

A topic was often listed as a Wikipedia title. If not, we tried several variants: removing pluralization and allowing non-capitalized words in the body instead of the title. If all of those, and some combinations, failed to find an article, then we took the top Google result when searching Wikipedia, and took the top available article from the main namespace.

Sometimes this best Google article had only a few paragraphs on the desired topic. If fewer than 25% of the paragraphs contained words from the topic, then only those paragraphs that did contain some topic word were selected. We used Google to find 39 of the Wikipedia topics; all 14 misidentifications were among these.

2.2 Lists

List questions ask for a list of entities that satisfy some condition, for example, "What diseases are prions associated with?". We must return members of the class of diseases, and ideally only those that involve prions. Wikipedia helps by providing members of many classes, and by offering synonyms for key terms. Having class-member relations is crucial also because answers often do not have the class name ("disease") near the answer ("Prions are known to cause Creutzfeld-Jacob syndrome").

Wikipedia provides class-member relations in three different forms: First there are entire articles that are just lists, whose titles are the class name (the article, "prizes, medals and awards") (48,412 class-member pairs); second, some articles have lists within them, ("Nirvana (band)" has "discography") (166,263 pairs); third, articles may mention a Category, for example the "Line Feed" article is a member of the category "Control characters", but we deemed these too dirty for immediate use.

Our baseline system used only about 3000 lists — in comparison, over 200,000 manually-compiled lists made many more answer types available.

Wikipedia also provides synonyms in its Redirect structure and subtitles. For instance, we can expand "disease" to "medical condition", "Woodrow Wilson Guthrie" to "Woody Guthrie" and "Woodie Guthrie", and "Nagano" to "Nagano, Japan". These manually constructed synonyms can significantly aid recall.

2.3 Definitions

Definition questions ask for any additional interesting information that can be found on a topic, without repeating any previous answers. Encyclopedia entries aim to have the definitional content we want, so we can recast the problem as trying to find the Wikipedia content in the newspaper articles.

System	Definition Passage
without Wikipedia	"was born 86 years ago in Okemah, Okla., to a couple of staunch Democrats"
synonymy	"folk singer"
Bleu and anaphora	"His most famous folk song – 'This Land is Your Land' – is not the patriotic ditty it appears."

Table 1. Definition answers for the topic "Woodrow Wilson Guthrie": Which answers are most essential and descriptive?

We used Wikipedia synonymy to help match variants of the topic; for example "folk singer" is mentioned with "Woody" Guthrie but not with "Woodrow Wilson Guthrie". We boosted keywords from the Wikipedia article, and in one run we applied Bleu (Papineni et al., 2002) as a similarity metric with sentences from the Wikipedia article to identify definitional information. These strategies helped us find nuggets that are more essential to the topic.

3. Results

Table 1 shows results on the TREC 2004 QA question set.

Lists restrict the space of possible answers, so it is to be expected that our 2005 system with Wikipedia lists should have higher precision and lower recall than the baseline 2005 system. In the disease example from above, having a list keeps us from having to guess any noun phrase, but some diseases may be missing from our list.

Run	Recall	Precision	F-measure ($\beta = 1$)
2004	0.125	0.185	0.132
2005	0.315	0.123	0.155
2005+wiki	0.237	0.163	0.170

Table 2. 2004—last year's results; 2005—improved system, same lists; 2005+wiki—improved system, adding lists from Wikipedia

Why then is the difference so small (non-significant), given such a huge gain in knowledge of answer types? The lists we knew best weren't the lists being asked about. Over the 75 questions, only 83 Wikipedia lists were used at all, and most of these were either augmenting existing lists ("countries by continent" duplicates "countries") or incorrectly used. Only 3 lists contributed meaningfully to the system.

Definition results improved qualitatively with Wikipedia strategies, but reliable quantitative evaluation is elusive.

Our preliminary[1] results indicate that Wikipedia has improved QA recall through synonymy, and QA precision by guiding our search for answers in the corpus.

4. Future Work

The Wikipedia has untapped structure that promises to be useful:

- Use structured information from tables and fields.
- Build class-member relations from Wikipedia categories.
- Use class-member relations from Wikipedia for other tasks like anaphora resolution, duplicate detection, disambiguation, and query expansion.
- Incorporate Wikipedia into our factoid system.

References

Ahn, D., et al. (2004). Using wikipedia at the TREC QA track. *Proceedings of the TREC-13 Conference*.

Katz, B., et al. (2002). Omnibase: Uniform access to heterogeneous data for question answering. *NLDB 2002*.

Katz, B., et al. (2004). Answering multiple questions on a topic from heterogeneous resources. *TREC-13*.

Papineni, K., et al. (2002). Bleu: a method for automatic evaluation of machine translation. *ACL2002*.

Voorhees, E. (2004). Overview of the TREC 2004 question answering track.

Wikipedia (2005). Question answering. *Wikipedia*.

[1]official judgements will not be available until November

A Synthetic Lattice for Structure Determination of Uncharacterized Proteins

Julie E. Norville NORVILLE@CSAIL.MIT.EDU
MIT Computer Science and Artificial Intelligence Laboratory, 32 Vassar Street, Cambridge MA, 02139 USA

1. Introduction

Proteins are constructed from amino acid chains which are folded into complex three-dimensional structures. Structural knowledge provides a molecular snapshot of the protein. Molecular pictures can assist scientists who wish to understand proteins, engineers who wish to modify either the proteins' structures or functions, and drug designers who wish to cause large effects within an organism due to small perturbations involving proteins at the molecular scale. The structure of many proteins remains unexplored. In order to determine a protein's structure it is not sufficient to examine only one molecule because it will be destroyed during the imaging process. Instead, one must examine an ensemble of molecules. One type of easily interpreted ensemble of molecules for structural determination is a crystalline lattice.

Unfortunately, protein crystallization is an art rather than a science. The target protein is screened against a diverse array of solutions until crystals are observed. Usually the first crystals that appear are not suitable for structure determination. To grow larger or more ordered crystals, it is usually necessary to change the pH, ionic strength, or temperature from the starting conditions. The process is painful, unreliable, and time consuming, and membrane proteins, which do not readily dissolve in water, are even more difficult to crystallize.

This paper describes a general method for preparing two-dimensional crystals from either monomeric proteins or symmetric protein complexes. Though formation of protein crystals is challenging, several classes of proteins form two-dimensional crystalline arrays in nature. I propose using such a natural crystalline array as a template; proteins can be bound to this lattice, using the natural structure of the array to form two-dimensional crystals. While common structural determination techniques require three-dimensional crystals, there are standard methods for which two-dimensional crystals are sufficient (Murata et al., 2000), (Saibil, 2000). In each of the following sections, the key parameter is that the three-dimensional structure of the target protein is undetermined.

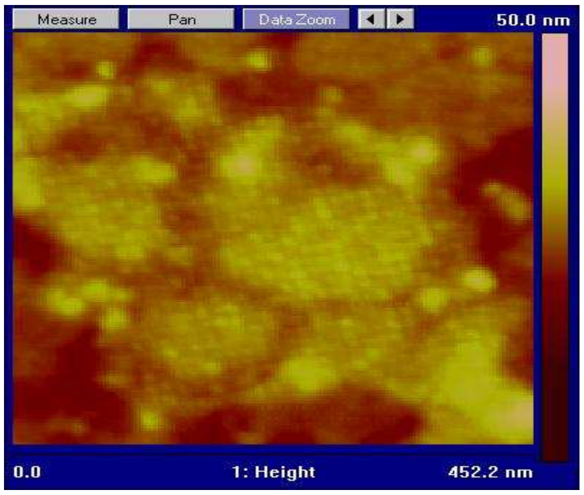


Figure 1. The protein we are using forms a crystalline lattice.

2. Making a Crystal out of a Crystalline Protein

A crystalline lattice is an ordered ensemble of molecules sufficient for structural determination. In this case the natural crystalline lattice will act as a support scaffold for the target molecule. Together they will create a new "synthetic" crystalline lattice. There exist natural proteins that form crystalline lattices. In some cases practical concerns prohibit the use of certain crystalline lattices. For example, a number of membrane proteins form two-dimensional crystalline lattices. However, they are hard to produce. In addition, other proteins require being at the right state before they can be formed into a lattice. Instead it was desirable to choose a protein that could be purified in large quantities and easily forms a lattice. S-layer proteins, or surface layer proteins, are a class of proteins that satisfy many of the criteria for an ideal crystalline lattice scaffold. In nature these proteins exist as a crystalline sheet on the outermost layer of many types of bacteria (Sleytr & Sara, 1997). The s-layer protein that best met these first conditions, *SbpA* did not yet have a determined structure. Thus, now that we have mastered the purification of the protein, we are currently in the process of determining its structure from two-dimensional crystals.

3. Choosing a Target Protein

The choice of the scaffold lattice constrains the choice of target protein for structure determination. It must fit within the lattice of the crystalline scaffold formed by the protein *SbpA*. *SbpA* is a tetrameric protein with a unit cell of size 13nm. Thus the target protein must either be less than 6.5nm X 6.5nm in size or it must be a symmetric tetramer smaller than 13 nm in size. Proteins that do not fit within these constraints would require a different crystalline scaffold. (The class of S-layer proteins provides many possible candidates with different lattice spacing and symmetry groups.) The dimensions of the target protein can be determined by examining single particles with electron microscopy. Symmetry can be determined using a combination of gel electrophoresis and analytical centrifugation.

4. Binding the Target Protein to the Crystalline Lattice

There are many ways in which the target protein can be bound to the crystalline lattice. A suitable binder must attach the target to the lattice so that it assumes one position. Currently it is unclear whether to use a structurally flexible binder that would it allow the target protein to find its own spatial equilibrium or to use a structurally rigid binder that will constrain the target protein in place. I am currently using a strategy in which a small peptide tag which binds to the crystalline lattice is genetically engineered onto the target protein. Ideally the binding tag will bring the target protein down to specific positions on the scaffold lattice. Currently we are in the process of finding a small peptide that binds to the crystalline lattice using the technique called phage display. Phage display allows one to select binding sequences from a library of 10^9 possible sequences. The phage display determined tag could be bound to the target protein in either a structurally loose or constrained fashion and has the possibility of being genetically engineered onto many types of target proteins.

5. Conclusions and Remaining Challenges

This paper provides the starting point for building synthetic crystals for structure determination. Currently, we will need to complete the structural determination of the protein *SbpA* which forms the crystalline lattice. If it does not diffract to high resolution, then another crystalline protein may be more suitable. We will also need to explore experimentally the binders which will connect the target protein and the lattice.

6. Acknowledgements

I thank my advisors Angela Belcher and Tom Knight. For assistance with the AFM imaging of *SbpA*, shown in the figure I thank Dan Solis. The author received graduate fellowships from the National Science Foundation, Merck, and Vinton Hayes. This work is supported by a Packard Fellowship, DARPA/ONR N00014-01-1-1060, and SRC task ID 1267.001.

References

Murata, K., Mitsuoka, K., Hirai, T., Walz, T., Agre, P., Heymann, J., Engel, A., & Fujiyoshi, Y. (2000). Structural determinants of water permeation through aquaporin-1. *Nature*, *407*, 599–605.

Saibil, H. (2000). Macromolecular structure determination by cryo-electron microscopy. *Acta Cryst. D*, *56*, 1215–1222.

Sleytr, U., & Sara, M. (1997). Bacterial and archaeal S-layer proteins: structure-function relationships and their biotechnological applications. *Trends in Biotechnology*, *15*, 20–26.

A Come-from-Behind Win or a Blown-Save Loss: Perspectives in Baseball

Alice Oh AOH@CSAIL.MIT.EDU
MIT Computer Science and Artificial Intelligence Laboratory, 32 Vassar Street, Cambridge MA, 02139 USA

1. Introduction

One can say a baseball game was a come-from-behind win or a blown-save loss. It all depends on who you are and how you view the event. The goal of our HiT (Highlights in Two) Project is to recreate a multimedia highlight summary of a baseball game from different perspectives, such that the users can view the event with their preferred bias. As the first step in automatically generating biased multimedia highlights, this paper presents the results of the analyses of bias in written media.

As an example of bias in the media, let us look at how media describes the fifth game of the American League Championship Series (ALCS) of 2004 between the Boston Red Sox and the New York Yankees. The Boston Globe and the New York Times featured headlines, respectively, “Two wins in hand, two to go”, and “Yankees lead series, 3-2”. The unbiased fact was that the New York Yankees had won the first three games of the best-of-seven series, then the Red Sox won the next two games. The biased interpretations were that, for a New Yorker, the Yankees were still up by one game, but for a Bostonian, the Red Sox had just won two games. The divergence of perspective does not end there. What follow the headlines are two articles that hardly seem to describe the same event. Besides these two game wrap-up articles, the two newspapers, as well as other online sports websites, such as ESPN (www.espn.com) and MLB (www.mlb.com) feature many articles presenting in-depth analyses from a variety of perspectives.

As such, sports fans have many choices for written media, but they have very few choices for visual media. Local and national sports channels broadcast actual games and/or highlights, but they only offer one perspective. The Internet offers more flexibility, such as MLB.com where users may search for short video clips, but those discontinuous and out-of-context clips do not satisfy the user’s needs for a biased analysis or an insightful story of the game. The HiT Project is directly aimed at those needs of the users.

2. Related Work

There have been many recent successes in multimedia analysis and modeling. In the sports video domain, research systems successfully analyze videos of soccer (Xie et al., 2004), baseball (Zhang & Chang, 2002), and other sports. The outputs from these systems vary in the level of domain specificity and semantic detail, but a typical classification algorithm produces an index of all events in a sports video such as play/break, score/batter change/base change. Such successes naturally lead to the next step of developing a system for automatic highlight generation.

Tailoring multimedia summaries to a specific user based on a user preference model has been an active area of research, particularly in the broadcast news domain (cf. (Maybury et al., 2004)) where they retrieve news stories that are of interest to a user. Our problem is different from the broadcast news domain in that the individual events we retrieve for the user need to be connected together to make one continuous story. Babaguchi et al. (Babaguchi et al., 2004) have done extensive research on generating video abstracts based on personal preferences, but our system goes beyond theirs in two ways. First, we infer the significance of the events based on context-dependent semantic features. For example, in a high-scoring baseball game, a one-run home run in the first inning may not be significant, but a two-run walk-off single in the ninth inning is probably much more significant. Secondly, we focus on the overall story that matches a user’s perspective, as opposed to selecting individual events based on user preferences.

3. Biases in Media

Although sports journalists stay true to the facts when they report on baseball games, many local newspapers and television broadcasts spin the stories to cater to the local audience. To understand how they actually do this, we analyzed articles from several local newspapers and compared them among themselves and against the Associated Press (AP) articles to quantify how “bias” is realized. We looked at three articles per game for six games: April 20, 25, 26 between the Boston Red Sox and the Baltimore Orioles, and May 13, 14, 15 between the Red Sox and the Seattle Mariners. We used articles in the Boston Globe for Red Sox coverage, Baltimore Sun for the Orioles coverage, and the Seattle Times for the Mariners coverage. We used the AP articles as the neutral counterpart.

Newspaper (Game)	Bos	Bal	Bos (Norm)	Bal (Norm)
Sun (3 Games)	91	128	0.83	1.52
Globe (3 Games)	145	52	1.33	0.62
AP (3 Games)	109	84	1.00	1.00

Table 1. Word counts of player and coaching staff names.

3.1 Bias in Discussing Players

The articles showed significant differences on three dimensions. First, the local articles covered more content on their home team's players compared to the AP articles. This is evidenced by the difference between the word count of the home team player names and the word count of the other team player names. Table 1 shows the comparison of the word counts between the Baltimore Sun articles and the Boston Globe articles. The first two columns show the raw counts, and the next two columns show the counts normalized to the counts in the AP articles. To save space, we present only the total counts, but all three games display consistent differences. The counts and the normalized counts clearly show that the Baltimore Sun writes more about the Orioles players, and the Boston Globe writes more about the Red Sox players. The Seattle Times articles and the Boston Globe articles between the Mariners and the Red Sox showed similar biases. A chi-square analysis shows significant difference between the Globe and the Sun at $p < 0.001$, also significant differences between the AP and the Globe, AP and the Sun, both at $p < 0.01$.

3.2 Bias in Choosing Events

Second, the local papers discussed more of their home team's successes than the neutral AP articles. To see this, we can dissect the events of each game into three categories: events that are included in all three articles, events in the home team's articles, and events in the visiting team's articles. For a given event, we can say it is a *local-positive* if it is an offensive play that led to, or could have led to scoring (e.g., a hit, a walk), or if it is a defensive play that prevented the other team from scoring, (e.g., a strike out, a double play). On the ten games we analyzed with three articles each, the home team's newspaper discussed 17 local-positive events on average, compared to 12 local-negative events. The AP articles did not show any significant difference in their coverage of one team versus the other. A t-test shows a significant difference between the Globe and the Sun articles at $p < 0.01$.

3.3 Bias in Interpreting Plays

Third, the local articles describe the events of the game with more focus on the home team's players. This bias is difficult to quantify because it is often reflected in the subtle nuances of the language used. The following shows one of the more straightforward examples of how a play was described by the three articles. The first is from the AP article and shows the objective account of two Seattle home runs. “Richie Sexson and Raul Ibanez added solo home runs for Seattle in the third inning, the first time the Mariners hit consecutive homers this season.”

The Boston Globe, interprets the same home runs as the pitcher's mistakes. “Gonzalez gave up back-to-back home runs leading off the third to Sexson and Raul Ibanez.”

The Seattle Times attributes the home runs to the batters' performance, further elaborating with the appropriate statistics and highlighting the Mariners' lead. “In the bottom of the inning, Sexson led off with his 10th homer, putting him on pace for 46 for the season, and Ibanez hit his fifth for a 5-4 Seattle lead.”

Although there are many ways to express the different interpretations, the most frequent ways are putting the home team's player as the subject, providing statistics that highlight the home team player's strengths, and mentioning scores when home team has captured the lead or has caught up to within a small margin.

4. Future Work

The next step is to apply the analyses in creating a system for generating highlights. As an intermediate step, we have also collected audio highlights from local radio stations and plan to do similar analyses. We have built a prototype system, and a demo will be shown at the CSW.

References

Babaguchi, N., Kawai, Y., & Kitahashi, T. (2004). Personalized abstraction of braodcasted american football video by highlight selection. *IEEE Transactions on Multimedia*, *6*, 575–586.

Maybury, M., Greiff, W., Boykin, S., Ponte, J., McHenry, C., & Ferro, L. (2004). Personalcasting: Tailored broadcast news. *User Modeling and User-Adapted Interaction*, *14*, 119–144.

Xie, L., Xu, P., & Chang, S. (2004). Structure analysis of soccer video with domain knowledge and hidden markov models. *Pattern Recognition Letters*, *25*, 767–775.

Zhang, D., & Chang, S. (2002). Event detection in baseball video using superimposed caption recognition. *Proceedings of ACM Multimedia'02* (pp. 315–318).

Zhou, W., Vellaikal, A., & Kuo, C. (2000). Rule-based video classification system for basketball video indexing. *Proceedings of ACM Multimedia Workshop* (pp. 213–216).

Incremental Optimization of Large Robot-Acquired Maps

Edwin Olson EOLSON@MIT.EDU
MIT Computer Science and Artificial Intelligence Laboratory, 32 Vassar Street, Cambridge MA, 02139 USA

1. Introduction

Many robotics applications require the robot to build a map of its environment. We consider the problem of *Simultaneous Localization and Mapping* (SLAM), i.e., building a map of an environment while exploring it for the first time. SLAM algorithms approach this by identifying features in the environment (e.g., the corner of a desk) and determining the relative positions of features. A robot's sensors are imperfect, so the relative position of one feature to another is almost always considered probabilistically– typically with a Gaussian distribution.

We can think about the map as a graph: features are nodes in the graph, and measurements which relate two features are edges. Each edge represents a rigid-body transformation and its uncertainty. We make the realistic assumption that each edge represents an *independent* constraint.

The heart of the SLAM problem is to determine the "best" map, the physical locations of features such that the constraints have maximum probability. We consider the case where the features are locations visited by the robot; as shown by (Montemerlo, 2003), positions of other features can be efficiently computed once the robot trajectory is known.

The classical method for SLAM problems is the Extended Kalman Filter (EKF). However, the EKF has several undesirable aspects: for N features it is $O(N^2)$ in space and time. It also performs poorly in the presence of highly non-linear constraints. The latter is true because the EKF commits to a linearization point at the time each constraint is incorporated; if the point at which the linearization occurred was inaccurate, linearization errors are introduced that cannot be undone later. In the SLAM problem, the orientation of the robot appears in most of the constraint equations in sine and cosine operations, which result in substantial linearization errors when the heading is not well-known. Sparse Extended Information Filters (SEIFs) also suffer from linearization errors, and incur $O(N^3)$ costs when computing the state estimate.

In this paper, we present an algorithm for optimizing pose graphs that is dramatically faster than the published state of the art. The improved performance arises from two sepa-

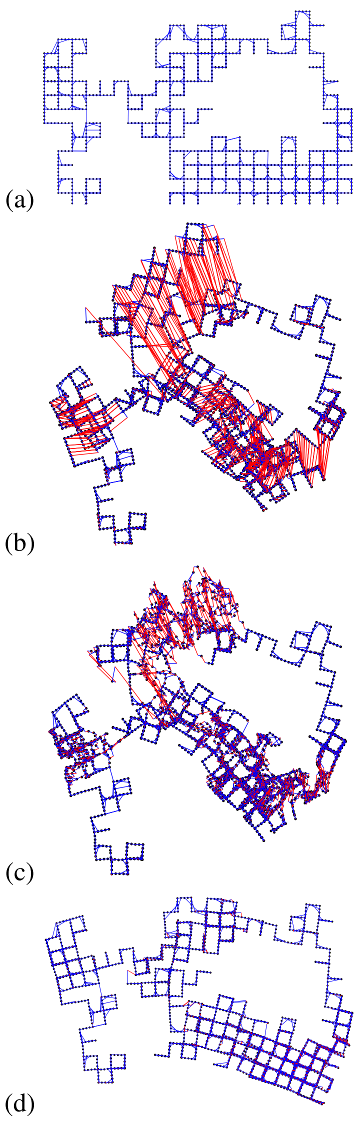


Figure 1. Sample synthetic problem. 3500 nodes with 2100 additional loop closures (A is 10500×10500, J is 16797×10500). Poses are shown as dots, lines represent constraints. Subplot (a) shows the ground truth map, (b) shows the simulated corrupted raw data, (c) shows the result of Duckett's Gauss-Siedel after 30 seconds of CPU time, and (d) shows the results of our method after convergence (about 10 seconds). The convergence rates for this experiment are shown in Fig. 3

rate ideas:

- The use of a different state representation which leads to a Jacobian that is better-suited to local iterative methods
- A variant of Stochastic Gradient Descent (SGD), a local iterative optimization method which escapes local minima more readily than Gradient Descent, Conjugate Gradient Descent, or Gauss-Seidel. Our variant exploits additional information available in the SLAM problem, allowing Newton steps rather than simple gradient steps.

2. Derivation

The maximum likelihood map can be incrementally computed using an iterative numerical approach. This approach has a number of distinct advantages: memory grows only as $O(N + E)$ (for N features and E edges), it can relinearize observations as the state estimate changes, and the incremental nature of the optimization means that an approximate map is always available for online path planning and exploration. The full state covariance never needs to be explicitly computed, however it can be reconstructed if necessary. A family of such approaches has been studied before (Duckett et al., 2000) and improved (Frese et al., 2005).

Before proceeding, we show how the graph can be optimized by solving a linear problem ($Ax = b$).

If x is the state vector representing robot poses, and $f()$ represents the constraint equations with expected values u and variances Σ, we can write:

$$-log\ P(x) \propto (f(x) - u)^T \Sigma^{-1} (f(x) - u) \tag{1}$$

We proceed by linearizing $f(x) = F|_x + J|_x \Delta x$, using matrices $F|_x$ and $J|_x$. At any particular iteration, we will simply write F and J, and will use $d = \Delta x$. We also set $r = u - F$, the residual. Eqn 1 then becomes:

$$\begin{aligned} -log\ P(x) &\propto (Jd - r)^T \Sigma^{-1} (Jd - r) \\ &= d^T J^T \Sigma^{-1} J d - 2 d^T J^T \Sigma^{-1} r + r^T \Sigma^{-1} r \end{aligned}$$

We wish to improve our map by finding a d that maximizes the probability. Differentiating with respect to d and setting to zero, we find that:

$$(J^T \Sigma^{-1} J) d = J^T \Sigma^{-1} r \tag{2}$$

This is the elementary $Ax = b$ linear algebra problem. If we solved for d directly (via inversion of A, or better by LU decomposition), we would have the method of nonlinear least squares. However, the size of A makes a direct solution impractical. Instead, we will estimate d.

When the state estimate is corrupted by significant noise, the local gradient will typically not point in the direction of the global minimum. Consequently, gradient methods typically fail to achieve a satisfactory solution.

In the SLAM problem, individual constraints all result in quadratic surfaces, ideal for optimization. It is only the sum of a number of constraints that leads to difficulties, so we propose using iterative methods that operate on only one constraint at a time. The optimal d can be written in terms of the sums of individual constraints by rewriting Eqn. 2 as:

$$d = (J^T \Sigma^{-1} J)^{-1} \sum J_i^T \Sigma_i^{-1} r_i \tag{3}$$

Naturally, we still cannot invert the information matrix ($J^T \Sigma^{-1} J$), but we can approximate the inverse using its diagonal elements; this approximation preserves the local gradient of the cost function. This is roughly equivalent to Jacobi Preconditioning, which uses the same approximation.

The canonical Stochastic Gradient Descent algorithm iteratively evaluates the gradient for each constraint (one constraint per iteration) and moves x in the opposite direction at a rate proportional to the *learning rate*. In the SLAM context, we can do better; we know what step size corresponds to a *Newton step*– a step that obliterates the residual of a given constraint. We still employ a learning rate parameter in order to ensure convergence, but the Newton step serves as an upper bound. While extensive research has been done in the area of learning rate schedules, we have found that a simple harmonic series ($1/t$) as originally suggested by (Robbins & Monro, 1951) works well.

3. State Space Representation

Previous authors used the *absolute global position* for their state space; i.e., the state vector was composed of (x, y, θ) values. The Jacobian of a rigid-body constraint between two poses is consequently *sparse*, acting like a "spring" connecting just those two poses. However, in addition to a loop-closure constraint, there is a segment of the robot's path that connects any two poses. For example, in Fig. 2, a loop constraint exists between poses A and D, but there is an additional path between A and D that goes through poses B and C.

If we alter the relative alignment of poses A and D in order

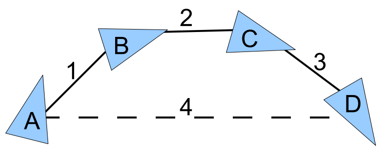


Figure 2. A simple pose graph. Optimizing constraint 4 typically has an effect on nodes B and C, in addition to A and D. Our proposed state space representation causes this dependency to appear in the Jacobian of constraint 4. This leads to more rapid convergence when using local iterative methods.

to reduce the error of constraint 4, poses B and C will also adjust position so that the total error will be reduced (due to the effects of constraints 1, 2, and 3.) Iterative methods, which use only a subset of the constraint information on each step, are unlikely to properly adjust B and C when they adjust constraint 4, since the effects of constraint 4 on nodes B and C appear in different rows of the Jacobian. This means that iterative updates to the state vector will be of poorer quality.

The Jacobian is a function of not just the constraint, but also the state space representation. If we change the state space representation, we can achieve a Jacobian that *does* capture the impact of moving two distant nodes on the nodes between them.

We use the *incremental global position* state space, in which the position of nodes is given relative to the previous node in the robot's trajectory, i.e.:

$$x = \begin{bmatrix} x_0 \\ y_0 \\ \theta_0 \\ x_1 - x_0 \\ y_1 - y_0 \\ \theta_1 - \theta_0 \\ \ldots \end{bmatrix} \tag{4}$$

The relative position of two nodes is now a function of all the incremental positions between them, so each row of the Jacobian now incorporates "springs" for all of the intermediate nodes. When J is premultiplied by the approximate inverse of the information matrix (from Eqn. 3), the "stiffnesses" of the intermediate linkages is set according to the strength of all the constraints which involve each node.

4. Results

With the incremental state space representation, the Newtonized Stochastic Gradient Descent algorithm estimates search directions d much more effectively, leading to rapid convergence, as illustrated in Fig. 1. Gauss-Seidel converges much more slowly.

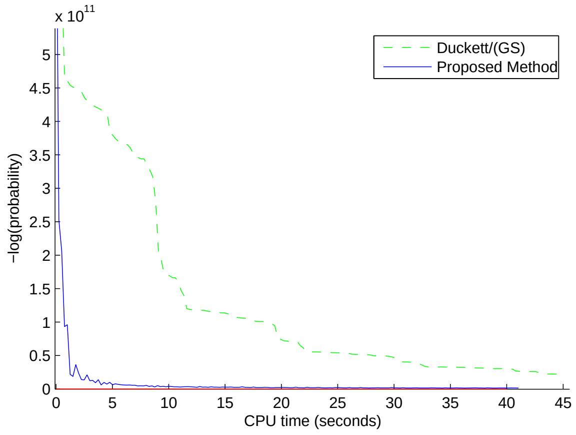


Figure 3. Convergence rates for Duckett's Gauss-Seidel approach and our method. Our method rapidly escapes local minima, converging quickly to a low-error solution.

Results of our algorithm are shown in Fig. 1. When the input graph is noisy, our method converges much more quickly than Gauss-Seidel relaxation, as shown in Fig. 3.

5. Conclusion

We have presented an iterative method for rapidly optimizing pose graphs, even in the presence of substantial initialization noise. This method shows promise in solving one of the open problems in SLAM: optimizing pose graphs after accumulating substantial error.

References

Duckett, T., Marsland, S., & Shapiro, J. (2000). Learning globally consistent maps by relaxation. *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA'2000)*. San Francisco, CA.

Frese, U., Larsson, P., & Duckett, T. (2005). A multi-level relaxation algorithm for simultaneous localisation and mapping. *IEEE Transactions on Robotics*.

Montemerlo, M. (2003). *FastSLAM: A factored solution to the simultaneous localization and mapping problem with unknown data association*. Doctoral dissertation, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA.

Robbins, H., & Monro, S. (1951). A stochastic approximation method. *Annals of Mathematical Statistics*, *22*, 400–407.

Reducing Configuration Overhead with Goal-oriented Programming

Justin Mazzola Paluska JMP@MIT.EDU
MIT CSAIL, 32 Vassar Street 32-G788, Cambridge, MA 02139

1. Introduction

Recent years have seen an explosion in the number and diversity of consumer-level electronic devices. Many of these devices work extremely well in one environment in a few scripted ways, but have the possibility of working in many other, unanticipated ways if users are willing to expend considerable effort configuring the devices and connecting them together. Unfortunately, this effort even confounds determined users with time and money to hire professional installers and technical support (Marcus, 2005).

As a motivating example, suppose a user would like to play a video from her laptop computer. The computer normally plays video on its own screen and internal speakers, but the user enjoys the experience more on her television and home theater system. As such, every time the user wants to watch a video from the laptop, she user must:

1. Connect the laptop's television output—if it has one—to one of television's inputs,
2. Connect the laptop's audio outputs to an adapter bought from RadioShack then that to the speakers' inputs,
3. Activate the inputs of the speakers and television,
4. Start the media player on the computer,
5. Instruct the laptop to render the media to its external outputs, and
6. Press "play" and hope that the television and computer agree on aspect ratios and other miscellany.

If the user would rather move the laptop to another room and play on the laptop's built-in components, she must reconfigure the laptop once again.

We believe that these configuration hassles can be reduced most of the time to "just play".

2. The "Just Play" Proposal

Configuration is difficult because devices usually cannot be controlled in a standardized way and, as such, leave it up to users to explicitly mediate between each device. Ideally, users should only need to tell their devices what to do at a very high level and the system should figure out how accomplish the low-level details.

We propose the following four part architecture to enable users to better use their devices by enabling automatic configuration:

A Common Wireless Interface and Control Mechanism A commoditized wireless medium reduces hardware connection mismatches to software problems that can be fixed with filters and adaptor functions.

Low-level Descriptions of Device Capabilities A television is not just a television, but a display device, NTSC tuner, audio output device, and a remote control receiver. These capabilities may be used in a variety of ways beyond simply watching television.

Specification of High-level User Intents Current systems don't know what the user wants to do and as a result can only do what the user explicitly tells it to do.

Goal-oriented Configuration System that can match high-level user intents with recipes for achieving those intents.

In such a system, the user tells the system to "Play Video" and the system responds by searching out for a way to "Play Video" given the resources at hand.

3. Initial Architectural Experiments

We built a prototype "Just Play" system for playing music using a Mac Mini, a laptop, and a speaker modified with an 802.11b wireless interface and MP3-to-analog decoding hardware.

Our prototype uses the O_2S Resources (Pham, 2005) over 802.11b wireless networks to provide the first two parts of the "Just Play" architecture. In particular, the O_2S resources framework provides us with a simple discovery system and a common interface through which to access devices as objects.

```
to PlayMusic(criteria):
  via MP3s:
  subgoals:
    source = MP3MusicStream(criteria)
    sink = AudioSink()
  eval:
    satisfaction = (source.satisfaction +
                    sink.satisfaction) / 2
  exec:
    fader.connect(source, sink)
```

Figure 1. Sample `PlayMusic` Technique

3.1 Goal-oriented Programming

For the last two architectural requirements, the prototype uses an extended version of the O_2S Goal-oriented programming system. Goal-oriented programming (Mazzola Paluska, 2004) provides a way of writing applications so that the algorithms, devices, and resources used can be evaluated and exchanged for ones with similar functionality as needed. There are two primary abstractions. *Goals* act as a specification layer. We use Goals to capture user and programmer intents. Goals are implemented by *Techniques*, a mixture of declarative statements and arbitrary code.

Goals are not bound to any particular Technique until runtime and this binding may change as better devices come up or the context in which a particular binding choice was made is no longer valid. The binding of Goals to Techniques is handled by a Planner that cooperates with Techniques to find the best way to satisfy the user's top level goals. Techniques may be Goal-oriented by declaring subgoals that the Planner recursively tries to satisfy.

3.2 "Just Play" Techniques

Our prototype Techniques use subgoals whenever possible to allow the Planner more choices in implementation. At the highest level is the `PlayMusic` Goal that the user invokes when she wants to listen to music. Figure 1 shows a sample Technique to `PlayMusic via MP3s`. A multitude of lower-level Techniques satisfy the subgoals of the `PlayMusic via MP3s` Technique down to Techniques for the capabilities of each device in our system. As new ways of playing music or new devices come along, we just add new Techniques to the Planner so it has more choices in satisfying the user's `PlayMusic` Goal.

A Technique may define `eval` code that refines the its `satisfaction`, or suitability to its Goal. For example, living room speakers satisfy the `AudioSink` Goal much better than do the speakers in a typical laptop, so the living room speaker Technique rates itself higher than that for a laptop speaker. From this evaluation, the Planner can choose the right `AudioSink` for the job at hand. The Planner also monitors `satisfaction` at runtime so if a device is no longer usable due to power or other problems, the Planner will notice and switch to new Techniques.

4. Related and Future Work

Configuration is a major burden in Grid Computing and provides a wealth of related work. For example, HP's SmartFrog (Goldsack et al., 2003) also offers a script-based auto-configuration, but concentrates mainly on starting and stopping services across the Grid, rather than connecting them together. Closer to our work is ISI's Pegasus (Deelman et al., 2004), which applies AI planning algorithms to generating scientific workflows on the Grid. However, Pegasus is based on moving files and programs around for execution, not connecting devices.

There are two areas in which we need to extend our prototype. First, we need to find a way to secure devices on the device network without configuration hassles. Whereas a laptop and speaker wired together are authenticated to each other by the wire between them, wireless networks offer no such authentication. This problem is compounded if users want to be able to share their device networks or use devices—such as projectors—that are a part of a shared infrastructure owned by no one user.

Second, we need to determine how to best give non-programmers a choice among equally good ways of satisfying a high-level goal. Programmers can write their own Techniques to do so, but we cannot expect most users to do this. A simple interface would be to choose a single plan and switch among them if the user rejects it. Alternatively, another interface lets users explicitly tell the system to use a device in a particular way, perhaps using speech recognition or gesture recognition.

References

Deelman, E., Blythe, J., Gil, Y., Kesselman, C., Mehta, G., Patil, S., Su, M.-H., Vahi, K., & Livny, M. (2004). Pegasus: Mapping scientific workflows onto the grid. *Lecture Notes in Computer Science*, *3165*, 11–20.

Goldsack, P., Guijarro, J., Lain, A., Mechaneau, G., Murray, P., & Toft, P. (2003). Smartfrog: Configuration and automatic ignition of distributed applications. http://www.hpl.hp.com/research/smartfrog/papers.htm.

Marcus, A. (2005). The out-of-box home experience: remote from reality. *interactions*, *12*, 54–56.

Mazzola Paluska, J. (2004). Automatic implementation generation for pervasive applications. M.eng, Massachusetts Institute of Technology.

Pham, H. (2005). A distributed object framework for pervasive computing applications. Master's thesis, Massachusetts Institute of Technology.

A Distributed Object Framework for Pervasive Computing Applications

Hubert Pham HUBERT@CSAIL.MIT.EDU
MIT CSAIL, 32 Vassar Street, Cambridge MA, 02139 USA

1. Motivation

Robust pervasive computing applications today span heterogeneous systems and often need to be dynamic and adaptive. However, traditional, asynchronous distributed systems are generally too complex and require developers to be deeply aware of the intricacies of the underlying system and platform. Furthermore, building adaptive applications also proves difficult because traditional systems tend to impose a static API between distributed components. The interfaces are determined at compile-time and provide no mechanism for changing the relationships between components during runtime, such as to adapt to hardware upgrade or failure.

This paper outlines several desirable traits of an adaptive software component architecture and presents a new set of abstractions that streamline development of distributed, pervasive systems. The current implementation forms the basis of the component system in O_2S (Saif et al., 2003).

1.1 Requirements

The first requirement is a clean and simple component interface, such that the application logic may modify or construct new implementations when adapting to changing environmental requirements. By separating the programming interface and the implementation technology, the component implementation can still be highly parallel and asynchronous, while a simple interface enables developers to construct, monitor, and debug these implementations.

Implementation technologies employed in a pervasive environment may span many different platforms and languages. Since the interface presented to the application logic is abstracted from the underlying implementation, the interface must be platform and language independent to fully capitalize on the wide variety of implementations available.

Both the interface and the implementation must be efficient. The interface should promote code reuse, enabling applications to adapt by re-configuring the overall implementation using basic, reusable modules. Once the application constructs an implementation, it must be efficient performance-wise to process high-bandwidth data streams.

2. Architecture

While there are many ways to use existing distributed object packages to fulfill the architectural requirements, this paper explores one abstraction that promotes a separation between policy and mechanism. We believe that an abstraction focused on this separation effectively simplifies the process of developing adaptive, distributed systems.

Three important features characterize the abstraction. First, the abstraction presents the developer with a simple API and environment, thereby simplifying the process of codifying the policy and application-specific logic. Second, developers use the simple interface to construct the desired application by connecting together a set of distributed software modules from a universe of generic components. Finally, while separating mechanism from policy may sacrifice performance for flexibility, the performance cost does not debilitate the component layer implementation.

In essence, the basic interface provides a mechanism for instantiating a collection of components on various hosts and interconnecting them into a network. The result implements a specific application or functionality; this mechanism promotes a circuit-diagram approach to application construction. The application logic also monitors the operation of the resulting circuit via a stream of high-level messages generated by the components. These message streams are used to report component failures, user inputs, or various resource-specific notifications. The health of devices hosting these components is also transparently monitored; component state updates and debugging output are collected, filtered, and serialized for presentation to the application logic. It thus becomes very natural to express the necessary logic behind adaptive applications, as the interface frees the developer from the implementation details that often complicate the model.

2.1 System Architecture

The system architecture comprises four components:

Synchronous Control A standard network object model (e.g., Remote Procedure Call) provides a synchronous control layer, which forms the basis of the simple environment for instantiating and connecting distributed modules. The network object model provides the veneer of a simple, sequential, and localized interface for controlling and monitoring the parallel, distributed component networks.

Data Streaming The data streaming mechanism connects components together into a highly parallel, distributed system of interconnected components. These data streams bypass the RPC system and hence do not incur the associated overhead. The operation of streams are somewhat autonomous, in that once the network is established, data simply flows between modules. Data streams are designed for applications that depend on routing real-time or rich media data between distributed modules. Streams also encourage component re-use by providing the mechanism for connecting together generic components to form new applications.

Serial Event Stream To monitor errors or other events generated by either the data streams or network objects, an event notification system provides a mechanism for sending serial messages to any network object's event queue.

Discovery & Health Monitoring Servers hosting network objects can often fail from network, power or hardware failure. The system is designed to detect such failures and inform the appropriate dependencies of the failed network object. The architecture also provides resource discovery, enabling applications to potentially recover from failures by discovering and substituting the failed object for an alternative resource during run-time.

2.2 The *Resources* Abstraction

The *Resource*s abstraction is a versatile RPC framework that facilitates the system design. With many traditional RPC packages, developers must generate client and server stubs for code modules, with some predetermined notion of where (which physical hosts) these modules will run. The amount of manual developer effort renders traditional RPC system unwieldy for dynamic, pervasive environment. The *Resource*s abstraction addresses these issues by providing an extensible and portable network object model that features dynamic stub generation, object interning and reference tracking. One resulting feature is that developers need not know *a priori* where code modules are executed, nor whether procedures are implemented locally or remotely: the invocation API is standardized, and the optimal invocation mechanism is always executed automatically.

3. Implementation Summary

As this architecture is platform and language independent, interoperable implementations of the architecture have been developed for a variety of languages (Java, Python, C) and platforms (Windows, Linux, Mac OS X). The Synchronous Control layer is an instantiation of the *Resource*s abstraction, which makes use of XML-RPC as the underlying transport mechanism – although the architecture is compatible with any modern RPC implementation. Data Streams are implemented as uni-directional TCP connections but represented as a *Resource* object, allowing developers to control and wire together these streams. Finally, special entities, called `Registry`s, monitor health and provide notification and lookup services. `Registry`s monitor liveness of objects via periodic UDP tokens and notify other objects whenever their dependencies fail.

4. Evaluation

This framework has been used to develop several O_2S applications; the abstraction of separating mechanism and policy has simplified distributed application development.

Performance-wise, benchmarks suggest a five-fold cost in using the *Resource*s abstraction for network object calls, when compared to Java RMI (Sun Microsystems, 1994). However, when streaming a 1MB file between hosts, the Data Streams implementation introduce some overhead compared to standard C sockets (two fold) but outperform RMI by a factor of five. The performance analysis is consistent with the architectural goals of the system; while RPC is flexible and convenient for constructing, connecting, and controlling distributed modules, we optimize performance for the high-bandwidth data streams.

5. Related & Future Work

Many standard RPC systems, such as Java RMI, CORBA (Object Management Group, 2001), and JINI (Waldo, 1999) subscribe to the conventional distributed application model, where applications are composed of statically-partitioned client-server modules. With traditional RPC derivatives, reusing components to construct different applications is often unwieldy, while adapting distributed applications by replacing constituent components is difficult.

Future work for this project include a security and authentication framework, as well as component hot-swapping for streamlined service upgrade and recovery.

References

Object Management Group (2001). *The common object request broker: Architecture and specification*. Object Management Group. 2.5 edition.

Saif, U., et al. (2003). A case for goal-oriented programming semantics. *Ubicomp 2003*.

Sun Microsystems (1994). Java Remote Method Invocation. `http://java.sun.com/rmi/`.

Waldo, J. (1999). The Jini architecture for network-centric computing. *Communications of the ACM*.

Modeling Online Sketching as a Dynamic Process

Tevfik Metin Sezgin MTSEZGIN@CSAIL.MIT.EDU
MIT Computer Science and Artificial Intelligence Laboratory, 32-235 Vassar st., Cambridge MA, 02139 USA

Abstract

Online sketching is an incremental and dynamic process; sketches are drawn over time, one stroke at a time, and can be captured with devices such as Tablet PCs and pen based PDAs. We have shown that the dynamic properties of the sketching process contain valuable information that can aid recognition. We describe a framework that can handle complex user input. Specifically, we show how we can take advantage of the regularities in sketching even when users draw objects in an interspersed fashion.

1. Introduction

Online sketching is an incremental and dynamic process: sketches are drawn one stroke at a time and be captured in devices such as Tablet PCs and pen based PDAs. This is unlike scanned documents or pictures which only capture the finished product. The dynamic properties of the sketching process contain valuable information that can aid recognition (Sezgin & Davis, 2005). In particular, in a number of domains the order in which users lay out strokes during sketching contains patterns and is predictable. We have presented ways of taking advantage of these regularities to formulate sketch recognition strategies (Sezgin & Davis, 2005). Here, we describe a framework that can handle more complex user input. Specifically, we show how we can take advantage of the regularities in sketching even when users draw objects in an interspersed fashion (e.g., start drawing object A, draw B before fully completing A, come back and complete drawing A).

2. Sketching as a stochastic process

Previous work has shown that in certain domains, stroke ordering follows predictable patterns and can be modeled as a Markovian stochastic process. Work in (Sezgin & Davis, 2005) shows how sketches of mechanical engineering drawings, course of action diagrams, emoticons and scenes with stick-figure can be modelled and recognized using Hidden Markov Models. In these domains, HMM-based modeling and recognition is possible because objects are usually drawn one at a time using consistent drawing orders. The HMM-based approach exploits these regularities to perform very efficient segmentation and recognition.

The HMM-based recognition algorithm scales linearly with the scene size, but requires each object to be completed before the next one is drawn. In certain domains, although there is a preferred stroke ordering, objects can be drawn in an interspersed fashion. For example, in the domain of circuit diagrams, people occasionally stop to draw wires connected to the pins of a transistor before they complete the transistor. One way of thinking about such a drawing scenario is that, instead of a single Markov process, we have multiple processes that generate observations, and the task is to separate observations from these processes. We model such drawing behavior as a multimodal stochastic process that can switch between different individual Markov processes, each of which captures drawing orders for individual objects. Although the new approach can also be described as a HMM, it is more easily described and understood using its dual representation as a dynamic Bayesian net (DBN).

Our approach to modeling interspersed drawing behavior is general enough to allow an arbitrary number of objects in a domain to be drawn in an interspersed fashion, but in practice people usually intersperse at most two objects. For example, in the circuit diagrams, unlike other circuit components, transistors have three connection points (emmiter, collector, base), and sometimes people draw the wires connecting to these points when the transistor is only partially drawn, causing interspersing of transistor and wire strokes. We have created a model specialized to handle interspersing of wires with other components in circuit diagram sketches. [1]

3. The network structure

Next we introduce our DBN model for circuit diagrams which handles interspersed drawing orders while still allowing polynomial time inference in the number of strokes.

We model the circuit diagram sketching process using a

[1]Although it is also possible to have a model general enough to allow interspersing between any two objects, we use this specialized model due to the nature of interspersing in our domain.

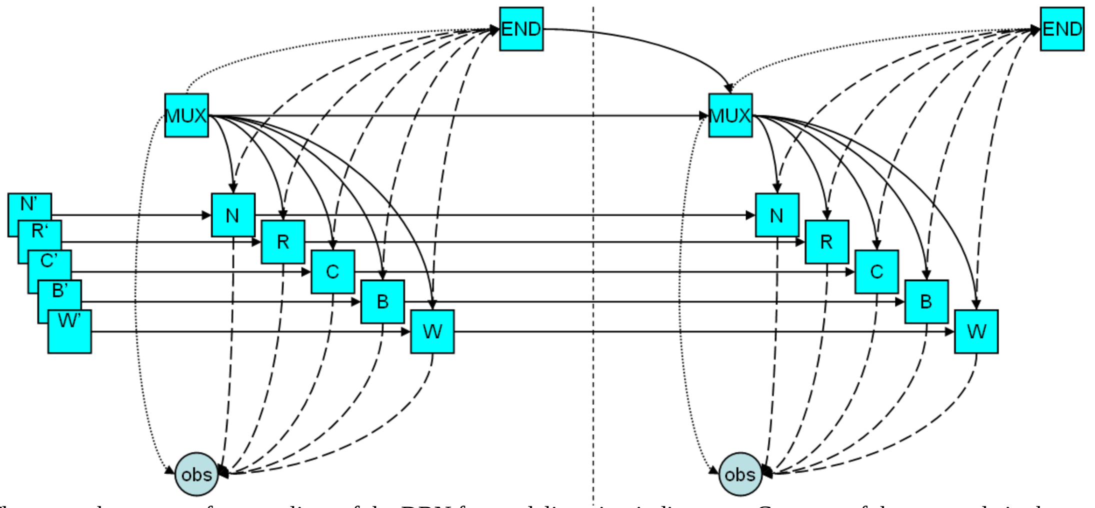


Figure 1. The network structure for two slices of the DBN for modeling circuit diagrams. Contents of the **OBS** node is shown in Fig. 2.

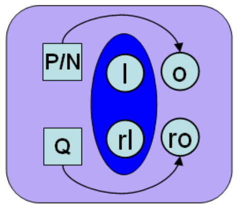


Figure 2. Details of the observation node. **L** and **RL** are Gaussian nodes that capture length and relative length. **O** and **RO** capture orientation and relative orientation. **P/N** and **Q** are mixture parameters for the relative features.

DBN (Fig. 1). The square nodes are discrete and the circular nodes are continuous. All nodes except the **OBS** node are hidden. The observation node **OBS** captures a number of features computed using the properties of primitives derived from strokes and the details of this node is shown in Fig. 2.

The hidden nodes in Fig. 1 and their connections specify the generative process that models the way in which objects in the domain are drawn. In our domain, we have five objects: NPN transistors, resistors, capacitors, batteries, and wires. Nodes **N**, **R**, **C**, **B** and **W** model the way in which these objects are drawn. Based on the value of the **MUX** node, only one of these processes is activated. The **END** node is simply a binary variable that species whether the latest observation completes drawing of the currently active object.

4. Node descriptions

We now describe each node in detail. We will adopt the generative process view of the model and describe the dynamics of the model from that perspective, but the reader should keep in mind that the model is used for assigning probabilities to series of observations obtained by encoding sketches (inference) and not for generating observation sequences.

4.1 The MUX variable

MUX keeps track of the main object the user is drawing (which can be interspersed with wires if it is a transistor). The actual observables are generated based on the value of this node and the individual object process nodes (i.e., **N**, **R** etc.). As a result, if there are N different objects that the user can draw, then the **MUX** node has N states. In addition, this node enters a special state when a pair of objects are being interspersed. There is a unique state for each pair of objects that can be interspersed. In our case, because only wires can be interspersed with transistors, there is only one such state. This is the state that we enter when the user starts drawing wires in the middle of a transistor and entering this state serves as a reminder that after the wires are drawn, we should complete that transistor that we initially started. So the **MUX** state has $N+1$ states (N for individual objects and one special state for interspersing wires with transistors).

$\mathbf{MUX_{t+1}}$ is conditioned on $\mathbf{MUX_t}$ and $\mathbf{END_t}$. The reasoning behind this conditioning is twofold: if there is no wire/transistor interspersing, the user may start drawing a new object only if the previous object is completed, and the probability of drawing a particular class of object may depend on the type of the last object.

4.2 The object variables (N, R, C, B and W)

These variables capture how individual objects are generated. In isolation, each node captures the state transition dynamics for an object, and when paired with the observation node, each node can be seen as an HMM that can generate features for that object. These nodes can change

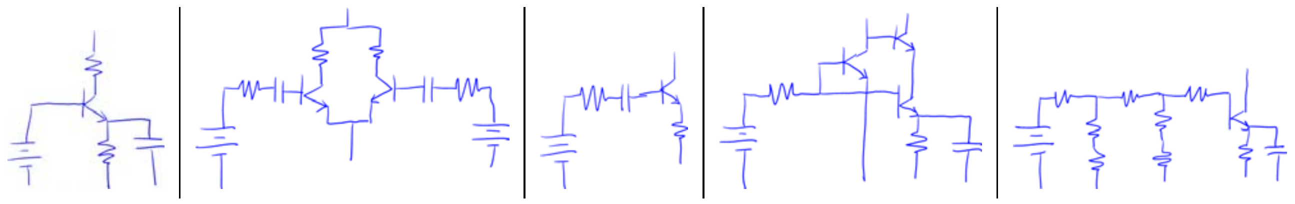


Figure 3. Examples of sketches

state only if they are active (as indicated by the MUX node). For example, the R node can change state only if the MUX node is in its *resistor* state; and in order to handle interspersing the W node can change state only if the MUX node is in its *wire* or *the user is interspersing wires with transistors* states. State transitions are Markovian, with each node conditioned on its value from the previous time slice. Finally the initial frame contains uniform object priors for the object nodes (N', R', C', B' and W')

4.3 The END variable

This discrete binary node is conditioned on one of the object variables, (i.e., at any time, only one of the arcs coming from the object variables is active). The choice of which parent is active is governed by the value of MUX. This is an example of the *switching parent* mechanism of DBNs that we use. The dotted arrow from MUX to END indicates that MUX is the parent that controls the switching behavior for parents of END, and only one of the the dashed arrows from the object variables to END is active based on the value of MUX.

4.4 The OBS variable

This is another example where we have switching parent behavior in our model. As in the END variable, the OBS variable is conditioned on only one of the object variables as determined by the value of the MUX variable. The details of the OBS node is shown in Fig. 2 (P/N, Q, L, RL, O and RO). L and RL are Gaussian nodes that capture length and relative length. O and RO capture orientation of the current primitive and the relative orientation with respect to the previous primitive. O and RO are continuous variables modelled as mixtures of Gaussians, while P/N and Q are the mixture variables that respectively model positive/negative slope for O and the quadrant of the current primitive with respect to the previous primitive for RO.

5. Implementation and results

In order to test our model, we collected circuit diagrams from electrical engineers. We asked users to draw multiple instances of circuits shown in Fig. 3. We collected ten examples of each circuit for a total of fifty circuits. Our current results are limited to data from a single user, but we believe they serve as a proof of concept.

5.1 Training

In order to train and test our model, we labeled the data by assigning object labels to groups of strokes. During training, in addition to the original observable node OBS, the values of MUX and END nodes were supplied, thus the only hidden nodes were the individual object nodes.

5.2 Classification

Once the model is trained, classification is performed by computing the most likely assignment to the MUX variable in each frame for the observation sequence derived from a given sketch.

5.3 Recognition performance

We used instances of the first four circuits in Fig. 3 as training examples and tested our system on all instances of the last circuit in Fig. 3. With only one example of circuits 1-4, we obtained a 73% correct classification rate. Using two and three examples of circuits 1-4 resulted in 89% and 93% correct classification rate consecutively. In cases where there were interspersing, we were able to detect interspersing 67% of the cases.

These results suggest that even as few as four examples can yield good recognition rates, and increasing the number of training examples results in better recognition rates even if the examples come from the same circuit. In addition the results suggest that the conceptual mechanism required to detect interspersed drawing works. We expect the recognition performance to get better with more training data and more data collection is currently underway.

Research Support

This research is funded by the MIT iCampus project and Project Oxygen.

References

Sezgin, T. M., & Davis, R. (2005). HMM-based efficient sketch recognition. *International Conference on Intelligent User Interfaces, San Diego CA January 2005.*

Engineering Transcription-Based Logic

Reshma P. Shetty RSHETTY@MIT.EDU
Thomas F. Knight, Jr. TK@CSAIL.MIT.EDU
MIT Computer Science and Artificial Intelligence Laboratory, 32 Vassar Street, Cambridge MA, 02139 USA

1. Introduction

Synthetic biology (http://www.syntheticbiology.org) is an emerging engineering discipline concerned with the design, fabrication and analysis of systems built from biological parts. Similar to the way electrical engineering takes advantage of the science of physics and chemical engineering takes advantage of chemistry to develop useful engineered systems, synthetic biology seeks to make use of biology. The potential application space for synthetic biology is enormous spanning areas such as chemical energy, materials and information. However, for the construction of synthetic biological systems to be routine, biology must be developed as a technology. Current systems are severely limited by the lack of many, well-characterized biological parts and devices.

The focus of this work is on engineering devices capable of implementing digital logic in cells. In these devices, information is encoded as transcription rates (or the rate at which DNA is transcribed to RNA). Thus, these transcription-based logic devices are composed of proteins that bind to DNA (termed a transcription factor or repressor) thereby regulating the transcription rate of that DNA (see figure 1). In this work, I describe a model that defines device behavior in terms of biochemical parameters like binding affinities and synthesis/degradation rates. Analysis of the model permits identification of which biochemical parameters have the greatest influence on device behavior. I also find target values for the key biochemical parameters. Determination of optimal parameter values informs the design of novel transcription-based logic devices.

2. Previous work

Current transcription-based logic devices are usually composed of bacterial repressor proteins (Elowitz & Leibler, 2000; Gardner et al., 2000). Since there are a limited number of these naturally occurring bacterial repressors, the scale and complexity of the systems that can be assembled from these devices is limited. To address this limitation, I will engineer synthetic transcription factors from zinc finger DNA binding domains and leucine zipper dimerization domains. Such an implementation change has several advantages. First, it should enable the eventual construction of a library of transcription-logic devices since there are large numbers of both kinds of domains available. Second, it will add modularity to the design of these devices since the two functions of the transcription factor, DNA binding and dimerization, are separated. This separation should enable independent tuning of the two domains. Third, since leucine zippers domains are capable of both homo- and heterodimerizing, a wider range of functions can be implemented in transcription-based logic.

The construction of logic devices from zinc fingers and leucine zippers has been proposed previously (Batten et al., 2004). This work differs from that presented here in several key ways. First, the devices proposed by Batten *et al.* encode signals as protein concentrations rather than transcription rates. The advantage of using transcription rates as the signal carrier is that devices are composable: any device may be connected to any other device. Devices whose output is protein concentration can only be connected to devices which take the same protein as input. Second, Batten *et al.* ask the question, given typical biological parameter values, what kind of device performance is expected? In this work, I instead ask, given that I as the device engineer have some measure of control over device design, how should I design the device in order to achieve the best possible device performance? By approaching the model from a purely design perspective, I obtain somewhat different results. Based on the results of my model analysis and previous work on dimeric zinc finger proteins, I have designed a novel implementation of transcription-based logic devices from zinc fingers and leucine zippers (Pomerantz et al., 1998).

3. Models inform device design

I develop a model that describes the device output in terms of the device input and relevant biochemical parameters. To enable the comparison of different device designs and to evaluate the affect of varying parameter values on device performance, I quantify device performance using existing metrics developed for digital logic devices like noise

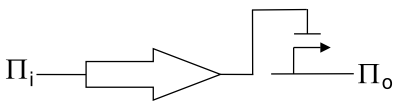


Figure 1. Schematic diagram of a transcription-based logic device: an inverter. The input signal Π_i causes the transcription of a gene encoding a repressor protein. The repressor protein dimerizes (two proteins bind to each other) and binds to its cognate promoter regulating the output transcription signal Π_o.

margins: the amount of noise a device can tolerate on its input signal without giving an erroneous output signal (Hill, 1968).

Several key observations which inform device design arise from the model analysis.

1. The parameter α_i, defined as the product of the ratio of mRNA and protein synthesis to their decay rates and copy number, determines the device input protein swing (the range of input protein concentration over which the device operates) as well as the device fan out (the maximum number of outputs the device can drive).

2. The protein-protein and protein-DNA binding affinities are the primary determinants of the shape of the device transfer curve. I obtain approximations for the values of these binding affinities that lead to good device behavior. Interestingly, it is not the absolute value of these parameters that determines device behavior (as most previous work suggests) but rather their value relative to α_i.

3. Inclusion of nonspecific DNA binding in the model leads to larger noise margins in the transfer curve.

4. An alternate device design in which several nonfunctional, high affinity protein binding sites are present yields a substantially improved transfer characteristic as measured by the noise margin.

4. Design of transcription-based logic devices

Using the model results as a guide, I selected previously characterized zinc fingers and leucine zippers to construct an initial set of synthetic transcription factors (Wolfe et al., 1999; Newman & Keating, 2003). I specify the DNA sequences encoding the DNA binding and dimerization domains so that each domain is a separate BioBricks part (see http://parts.mit.edu for more information). Designing the synthetic transcription factors in this way allows easy mixing and matching of DNA binding and dimerization domains via BioBricks standard assembly techniques. Previously, transcription factors were specified as a single BioBricks part and thus not very modular. Another contribution of this work is that many of the designed devices use heterodimerizing leucine zipper domains rather than the typical homodimerizing domains. Such devices will be capable of carrying out the logical NAND operation demonstrating that transcription-based logic is capable of implementing arbitrary logic operations.

5. Future Work

Having completed the DNA sequence specification of my transcription-based logic devices, I am in the process of fabricating the synthetic transcription factors (and cognate promoters) using standard molecular biology techniques. Analysis of device behavior should either yield working transcription-based logic devices or shed light on how to better engineer these devices in the future. Additionally, I am also developing methods for quantitatively characterizing device behavior. These methods will use both quantitative RNA measurements to characterize the device transfer curves and flow cytometry to assess variations in device performance between different cells. The measured transfer curves can be directly compared to the model results in order to validate the model. Moreover, insights from the model should aid in debugging issues in device function.

References

Batten, C., Krashinksky, R., & Knight Jr., T. (2004). A scalable cellular logic technology using zinc-finger proteins. *3rd Workshop on Non-Silicon Computing*.

Elowitz, M. B., & Leibler, S. (2000). A synthetic oscillatory network of transcriptional regulators. *Nature*, *403*, 335–8.

Gardner, T. S., Cantor, C. R., & Collins, J. J. (2000). Construction of a genetic toggle switch in *Escherichia coli*. *Nature*, *403*, 339–42.

Hill, C. F. (1968). Noise margin and noise immunity in logic circuits. *Microelectronics*, *1*, 16–22.

Newman, J. R. S., & Keating, A. E. (2003). Comprehensive identification of human bZIP interactions with coiled-coil arrays. *Science*, *300*, 2097–101.

Pomerantz, J. L., Wolfe, S. A., & Pabo, C. O. (1998). Structure-based design of a dimeric zinc finger protein. *Biochemistry*, *37*, 965–70.

Wolfe, S. A., Greisman, H. A., Ramm, E. I., & Pabo, C. O. (1999). Analysis of zinc fingers optimized *via* phage display: evaluating the utility of a recognition code. *J Mol Biol*, *285*, 1917–34.

Short Talk Abstracts

Personifying Public Key Infrastructure

Jacob Beal and Justin Mazzola Paluska

We believe cryptography can be visible and comprehensible to the average user by showing a face for each public key involved in a transaction. Currently, checking to see if a key is correct requires looking at long strings of alphanumeric characters, and most browsers display nothing but a lock icon during routine transactions. The result is that users are largely unaware of the certificates which they interact with, or the trust issues involved in accepting or rejecting certificates. We propose to fix this by mapping the signature of a certificate to parameters specifying a realistic computer-generated face to be displayed during the transaction, salting the mapping with a password to prevent an attacker from guessing which faces will look similar. When the key changes, the face changes and the user will likely notice. They then know to be suspicious until they've developed a good reason to trust the new face.

LabelMe: A Database and Web-Based Tool for Image Annotation

Bryan Russell, Antonio Torralba, Kevin P. Murphy, William T. Freeman
presentation by: Biswajit Bose

Research in object detection and recognition in cluttered scenes requires large image collections with ground truth labels. The labels should provide information about the object classes present in each image, as well as their shape and locations, possibly other attributes such as pose. Such data is useful for testing, as well as for supervised learning. This project provides a web-based annotation tool that makes it easy to annotate images, and to instantly share such annotations with the community. This tool, plus an initial set of 10,000 images (3000 of which have been labeled), can be found at
http://www.csail.mit.edu/~brussell/research/LabelMe/intro.html
or Google search: LabelMe.

Object Manipulation and Control for Simulated Characters

Yeuhi Abe

In animation it is important that a character can manipulate objects in its environment. Motion capture is one way to create high quality animations, but is restricted to static, prerecord instances of each motion. A good way to ensure physically correct animations for a wide range of manipulation task is by simulating both the character and the object. By modeling the character as a set of linked rigid structures, the problem can be formulated analogously to that of manipulation control in robotics. However, most manipulation task are only concerned with the motion of one part of the body (e.g. the hands), leaving the motion of the rest of the body under determined. In other words, the inherent redundancy in the kinematics of anthropomorphic characters makes the control problem ill posed. Thus the goal of this project is to develop control algorithms for the simulation of anthropomorphic characters performing manipulation tasks that incorporate movement models of human figures to help deal with the redundancy in a realistic fashion.

Weapons of Mass Construction

Justin Werfel

The goal of my work is to develop systems to automate construction with swarms of autonomous agents. That is, I want to be able to take an indeterminate number of simple, identical robots; give them a picture of something I want built; and have them proceed to reliably build it without further intervention. The approach is to use a partially built structure as a reference, such that robots capable of only local observations will act to complete the desired structure, avoiding intermediate configurations where further progress is blocked. Robots do not explicitly communicate nor coordinate their actions. A few fixed behaviors are sufficient to produce arbitrary solid structures, without centralized control or preplanned construction sequences. Adding capabilities to the building materials can increase the availability of global structural knowledge, thereby increasing robustness and greatly speeding construction. I'll show simulation results in two and three dimensions, and a one-robot hardware prototype in two dimensions.

Integrating on a Spatial Network Without Coordinates

Jacob Beal

I show how to calculate surface and volume integrals on a space-approximating network without coordinate or range information. A sensor network might use such integrals to estimate the number of people in a crowd or the severity of a pollutant spill. The integral is approximated using Thiessen weights estimated from the neighborhood size and expected communication range of each node, and has a predictable error due to edge effects. In simulation, the estimated integral value is within one standard deviation of the predicted value.

Automatic Human "Diary" Generation

Vivek Kale

Especially with today's complex and fast-paced lifestyles, we need some way of tracking what one does throughout the day. The Human Dynamics Group at the Media Lab has recently begun to make this seemingly impossible task into reality. We have implemented real-time feedback systems to recognize patterns in one's actions, speech, biological conditions, and even social interactions. This is made possible by cutting edge hardware and complex algorithmic techniques which can be used to collect and analyze human events over some period of time. All of this can be achieved with only a small Linux-based computer the size of one's palm equipped with audio input, camera, motion sensors, and GPS technology. What makes this groundbreaking is its reliability and computational speed--making it practical for applications ranging from casual office meetings to military operations and communications.

Modular Static Analysis with Sets and Relations

Viktor Kuncak

Complexity of data structures presents a challenge for current static analyses, forcing them to report false alarms or miss errors. A static analysis can track properties of a Java object while it is referenced by a local variable, but when the object is stored in a data structure, this information is lost, forcing the analysis to make a worst-case assumption.

I will describe a new system for verifying programs with complex data structures. The system is based on a methodology for specifying interfaces of data structures by writing procedure preconditions and postconditions in terms of abstract sets and relations. The system then separately verifies that 1) each data structure conforms to its interface, 2) data structure interfaces are used correctly, 3) program satisfies desired high-level application-specific invariants. The system verifies these conditions by combining decision procedures, theorem provers and static analyses.

Hierarchical Recursive Feature Elimination: A Proposed Method for Reducing the Set of Features Used in an EEG-based Epileptic Seizure Detector

Elena Glassman

This research is concerned with reducing the number of channels (therefore reducing electrodes) and features computed from each channel of an EEG-based, patient-specific epileptic seizure detector (Shoeb, 2003), while still maintaining performance. Such a reduction will decrease overall computational complexity and power consumption and hopefully increase patient comfort.

A Recursive Feature Elimination (RFE) implementation (Lal et al., 2004), an SVM-based greedy feature eliminator which eliminates entire channels (groups of features) at a time, reduced, on average across 20 patients in our dataset, the number of channels from 21 to 8.35.

To circumvent computationally intractable problems resulting from large numbers of features, the Hierarchical RFE (H-RFE) method is proposed. In H-RFE, RFE is first applied to remove as many pre-defined groups of spatially similar channels as possible, then remove as many of the remaining channels as possible, and finally remove as many features from the remaining channels as possible. Work on its implementation is ongoing.

Molecular Simulations

Greg Pintilie

Molecular simulations are used to study and predict the structure and behavior of molecules. The atoms in a molecule are approximated using spheres. The 3D structure which minimizes an energy function is obtained using optimization techniques. The energy function includes terms for forces between atoms, forces due to bonds between atoms, and other empirical terms. Molecular simulations can be used to study protein structure, for example to predict how proteins fold or how they bind other molecules. Much work still needs to be done to improve the accuracy and efficiency of such methods. Such tools would allow us to better understand how proteins function, and how to design better drugs for treating various diseases.

Protein Structure Prediction with Multi-Tape Grammars and Support Vector Machines

Blaise Gassend, Charles O'Donnel, Bill Thies, Marten van Dijk, Srinivas Devadas

Predicting the 3D structure of a protein from the sequence of amino acids that make it up is a major unsolved problem in biology. Context free grammar methods have had much success in RNA structure prediction, but are insufficient to capture the two-way long-range interactions that are present in protein beta sheets. Moreover, the free energy data of elementary interactions, which is used when optimizing RNA structures, is unavailable for proteins. To get around these two issues, we want to use multi-tape grammars to model protein secondary structure, and to use machine learning methods to find pseudo free energies that are consistent with experimentally observed protein structures. So far, we have experimented with Support Vector Machine methods on structured output spaces to learn pseudo free energies in the simplified case of proteins with no beta sheets.

A Synthetic Lattice for Structure Determination of Uncharacterized Proteins

Julie Norville

Proteins are constructed from amino acid chains which are folded into complex three-dimensional structures. The structure of many proteins remains unknown. In order to determine a protein's structure it is not sufficient to examine only one molecule because it will be destroyed during the imaging process. Instead, one must examine an ensemble of molecules. One type of easily interpreted ensemble of molecules for structural determination is a crystalline lattice. Currently, a solution with the right pH, ionic strength, or temperature must be found in order to grow large and well-ordered crystals. This process is painful, unreliable, and time consuming. In nature, several classes of proteins have the property of forming two-dimensional crystalline arrays. I propose using such a natural crystalline array as a template; proteins can be bound to this lattice, using the natural structure of the array to form two-dimensional crystals. While common structural determination techniques require three-dimensional crystals, there are standard methods for which two-dimensional crystals are sufficient.