Learning by Learning To Communicate

by

Jacob Stuart Michael Beal

S.B., Massachusetts Institute of Technology (2000)
M.Eng., Massachusetts Institute of Technology (2002)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy in Computer Science and Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2007

Author . . . .
Department of Electrical Engineering and Computer Science
August 24th, 2007

Certified by . . . .
Gerald Jay Sussman
Panasonic Professor of Electrical Engineering
Thesis Supervisor

Accepted by . . . .
Arthur C. Smith
Chairman, Department Committee on Graduate Students

Learning by Learning To Communicate

by

Jacob Stuart Michael Beal

Submitted to the Department of Electrical Engineering and Computer Science
on August 24th, 2007, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy in Computer Science and Engineering

Abstract

Human intelligence is a product of cooperation among many different specialists. Much of this cooperation must be learned, but we do not yet have a mechanism that explains how this might happen for the "high-level" agile cooperation that permeates our daily lives.

I propose that the various specialists learn to cooperate by learning to communicate, basing this proposal on the phenomenon of *communication bootstrapping*, in which shared experiences form a basis for agreement on a system of signals. In this dissertation, I lay out a roadmap for investigating this hypothesis, identifying problems that must be overcome in order to understand the capabilities of communication bootstrapping and in order to test whether it is exploited by human intelligence.

I then demonstrate progress along the course of investigation laid out in my roadmap:

- I establish a measure of *developmental cost* that allows me to eliminate many possible designs
- I develop a method of engineering devices for use in models of intelligence, including characterizing their behavior under a wide variety of conditions and compensating for their misbehavior using *failure simplification.*
- I develop mechanisms that reliably produce communication bootstrapping such that it can be used to connect specialists in an engineered system.
- I construct a demonstration system including a simulated world and pair of observers that learn world dynamics via communication bootstrapping.

Thesis Supervisor: Gerald Jay Sussman
Title: Panasonic Professor of Electrical Engineering

Acknowledgments

No thesis is an island. No idea is birthed outside of a network of family, friends, colleagues, and critics that help to give it form and mold its awkward first fumblings into maturity. So let me begin by thanking a few of the many who have had a hand in shaping me as a researcher and in making this work a reality:

Gerry Sussman, my advisor, for the pointer to Kirby's work that started it all, for constructive criticism, encouragement, and argument, for explaining to me that everybody has dry spells, and for his insights into the relationship between language, computation, and robust engineering. Patrick Winston, for exciting me about intelligence and being susceptible to my enthusiasm in return, and for his invaluable aid in my ongoing battle to learn to speak both precisely and clearly at the same time.

Abi Harper, my wife, for teaching me ways around writer's block, coder's block, and perfectionism, for being my equal, and for always having a sharper understanding of my work than I expect. Jean and Jon Beal, my parents, for giving me the foundation on which I build and for being always a safe harbor and source of support in time of need.

David Harper and Julie Sussman, whose careful editing has greatly improved the quality of this document. Shaun Bennett and Alice Harper for lending me places of retreat where I could work monastically when I needed. Tim Shepard and Justin Mazzola-Paluska, constant colleagues and sharp critics who keep me grounded and connected to the larger world. Jessi Kleiss, for deep friendship and understanding from 3,000 miles away.

Tom Knight, for introducing me to paths not taken, and who, along with as others in the synthetic biology community like Drew Endy and Randy Rettberg, has encouraged and tried to answer my naive questions. Joel Moses, for shaking my mind with philosophical connections I am still digesting. Rod Brooks, for encouraging me to spread dangerous ideas.

Joe Foley, Jade Wang, Richard Tibbetts, Peter Litwack, Ken Clary, and other Assassin's Guild members who have debated game design with me and who have contributed to the development of failure simplification. David Houston, for pointing me at some important insights on the nature of research, and Sara Bennett, for giving me an appreciation of complexity, the slow struggle toward enlightenment, and a reminder about the profound simplicity of the scientific method.

All those who have pushed me and challenged me on my ideas, asking the right questions and criticising me when I needed it, especially Hal Abelson, Vikash Mansinghka, Whitman Richards, and Brian Williams.

Many other friends and colleagues deserve credit as well: Jonathan Bachrach, Liz Baraff, Bob Berwick, Keith Bonawitz, Won Chun, Dave Cliff, Michael Coen, Diana Dabby, Ian Eslick, Seth Gilbert, Bob Hearn, Leslie Kaelbling, Nancy Lynch, Marvin Minsky, Radhika Nagpal, Sajit Rao, Edwina Rissland, Deb Roy, Dana Scott, Howie Shrobe, Push Singh, Peter Szolovits, Josh Tenenbaum, Christof Teuscher, John Wilbanks. They and many others share in the credit for this document; any blame is mine alone.

Contents

List of Figures

Chapter 1

Introduction

Human intelligence is one of the basic mysteries of our world, and one that has proven remarkably difficult to untangle.

As an engineer studying the subject, my approach is to try to build an intelligence with human-like capabilities, on the assumption that the obstacles I encounter along the way will give me insight into the nature of intelligence.

I feel safe in assuming that this is an extremely complicated problem, and that as an engineer, I should only expect to be able to work on one part at a time. So far so good: engineers are great at breaking systems down into modular parts so that each is specialized to handle one part of the problem and we can work on each specialist separately.

In building an intelligence, though, there is a real problem: it is hard to isolate a specialist part. If we want a system that exhibits the broad competence, flexibility, and common sense of human intelligence, then even the simplest of matters are "AI-complete," a joking term that means that an artificial intelligence task spirals outward and ends up involving the whole complexity of intelligence.

Consider, for example, navigating the humble traffic light—my example of choice for this dissertation (Figure 1-1). What could be simpler? Go when the light green, stop when it is red—a simple interaction between motor and vision. But when the light is yellow, go if you can make it before it turns red—sequencing and timing. Right turn on red—an exception—except where there is a "No Turn on Red" sign—and language gets involved. When turning left, will the oncoming cars yield? Here in Massachusetts[1] you'd best make eye contact, judge their intentions, and maybe figure out some primate dominance issues. It only gets worse from here: near my house, there is a light that shows red and a green right arrow simultaneously, then changes to yellow, then to red and a green left arrow. Or a slow pedestrian is still in front

[1] "Massachusetts: where people want to put their car where your car is." –Abi Harper

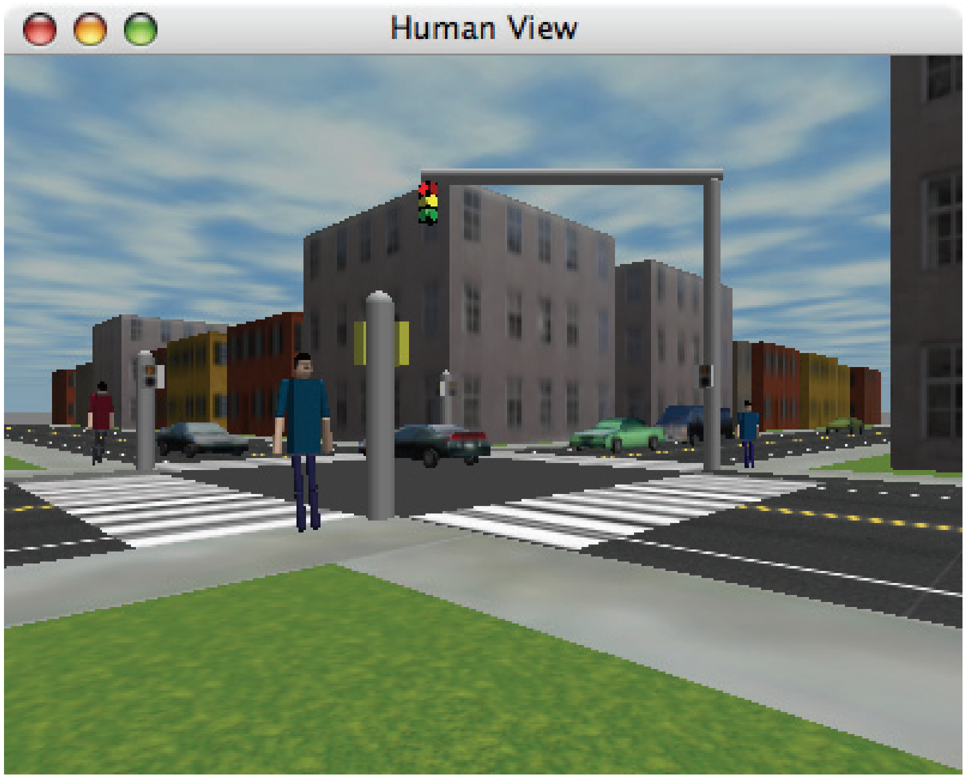


Figure 1-1: Screenshot of a simulated four-way intersection showing a car running a yellow light at around noon.

of you when the light turns green. Somebody might run their red light. It is raining heavily and the lights are out. There is an accident, and you need to go around, interpreting the gestures of the officer directing traffic. There is a parade, a street festival, a kid chasing a ball, heavy fog, black ice, you smell smoke, your cell-phone rings, somebody honks their horn.

Pretty much every matter is like this, potentially requiring complicated cooperation from any arbitrary combination of specialists. Humans are very good at this, so much so that when an appropriate team of specialists assemble themselves on the fly to address a novel situation, we think it merely natural and look down on those who are less nimble in their synthesis.

This presents a real problem for us as students of intelligence. We do not understand how humans carry out this feat and have not yet been able to duplicate it, despite heroic cognitive architecture projects such as SOAR[41], ACT-R[27], Cyc[15] and OpenMind[35] (to name only a few). How can we attempt to build a system that is smart like a human if we do not know how to produce agile cooperation among a group of specialists?

The mystery is deepened by the fact that most of this teamwork involves things that must be learned, rather than things that could be built into our DNA. While instinctive capabilities like hand-eye coordination are surely involved, they are insufficient to explain "high-level" behavior like deciding to flex your ankle to press the brake and keep the car from moving forward because the officer directing traffic is gesturing for another car to go but for you to wait and you are willing to submit to the officer's wishes.

Thus, if we are to understand human intelligence, we must understand how the various specialists that contribute human intelligence learn to work together. My hypothesis is *learning by learning to communicate*—that the specialists learn to cooperate by learning to communicate, exploiting the phenomenon of *communication bootstrapping*, in which shared experiences form a basis for agreement on a system of signals.

In this dissertation, I lay out a roadmap for an investigation that will prove or disprove this hypothesis (containing much more than one dissertation worth of work) and carry out the first few steps. Some of these early steps are problems that pertain not just to my hypothesis but to general problems in software engineering and the study of intelligence. As a result, the work presented in this dissertation is a significant contribution both to computer science and to the study of intelligence, regardless of whether the "learning by learning to communicate" hypothesis ultimately proves correct.

1.1 Communication Bootstrapping

A network of devices exhibits *communication bootstrapping* when they use shared experiences to reach agreement on a system of signals for communicating with one another.[2] Originally a possible explanation for how different specialists that contribute to human intelligence might learn to understand one another, it also forms a basis for my more radical hypothesis that human intelligence may arise largely from the struggle of the various specialists to understand one another.

1.1.1 Learning to Communicate

It is not immediately clear how the various specialists that contribute to human intelligence can understand one another at all. The regions of brain apparently specialized for things like vision, language, and motor control are centimeters apart, and a distance of a few centimeters is a massive distance on a cellular scale. It appears to

[2]Communication bootstrapping is originally defined in [4] and [3].

be costly to make precise connections over such large distances (more on this in Section 4.3). Combined with the significant variations that inevitably take place during development, it is clear that most of the connections between distant parts of the brain cannot be laid out with a deterministic wiring diagram like the ones we use when we fabricate microprocessors.

Rather, the brain must be able to self-organize the connections that form between specialists into an effective communication link. Communication bootstrapping began as a model of how this might be carried out, and was inspired by Kirby's work on language evolution[21, 22].[3]

Kirby has proposed an alternate explanation of universal grammar, the common structure found in all languages with native speakers.[4] The more popular explanation of universal grammar is that it is a precise mechanism built in the human brain, then configured for particular languages. Kirby's alternate proposal is that humans have only a shared set of learning biases. In Kirby's model, when an adult and child share an experience, these biases mean that the child is likely to jump to the right conclusions when trying to break the adult's speech into meaningful fragments. Universal grammar would then be the result of selection for languages that fit well with these learning biases.

Whatever the truth may be about human language, Kirby's theory hints that specialists could learn to communicate with one another using only coincidences between the senses and a common mechanism for connecting together signals and their meanings.

For example, the walk light and the audible signal that accompanies it usually occur together. Communication bootstrapping between the vision specialist and the hearing specialist results in agreement on a signal that means the walk light to the vision specialist and the accompanying sound to the hearing specialist. Thus, when the hearing specialist signals the presence of the sound, the vision specialist interprets it as the walk light, which it knows how to look for and understand.

I have demonstrated this for a limited domain in previous work[4, 3], using a simple architecture that I review in Section 6.1. The extension of communication bootstrapping to a broader range more useful for building an intelligence is one of the

[3]A note on related work: Kirby's work is closely related to that of Steels on grounded language acquisition[38], that of Yanco on self-configuring communication for mobile robots[45], and Batali on learning grammar in recurrent neural networks[2]. Indeed, a small sub-field on synthetic language has blossomed—a good summary can be found in [23]. In general, however, the systems they have studied have small vocabularies, converge slowly, and are restricted to linear utterances such as character strings.

[4]Even invented languages, such as American Sign Language, might not conform to universal grammar initially, but they are remolded by the children who are raised speaking them.

results of this dissertation.

1.1.2 Learning by Learning to Communicate

Communication bootstrapping also allows us to investigate another, more radical hypothesis: that human intelligence may arise largely from the struggle of various specialists to understand one another.

Recent work in cognitive science suggests that there is something very special about how the various specialists contributing to human intelligence learn to work together. Infant studies show that humans are born with essentially the same cognitive faculties as other mammals,[5] and it is hypothesized that our unique intelligence is not the result of any particular faculty, but of the cooperation that develops among them and appears to be related to language[37]. It is not known, however, whether language enables cooperation to develop or vice versa.

For example, human adults can reorient themselves to find a location specified as a combination of two types of feature, color and geometry, while children less than five years old and rats only use geometry, a single feature, to reorient[17]. The color and geometry faculties somehow work together in adults, but not in children or rats. Moreover, the transition between infant and adult capabilities correlates with production of the words "left" and "right"[18] and adult reorientation is impaired by simultaneous performance of a language task but not a rhythm task[19]. Cooperation between faculties is thus apparently related to language.

In another example, small children combine three separate faculties to develop the concept of number[8]. Humans, like many animals, have built-in faculties for perceiving numeric features, such as analog magnitude (an imprecise measure of the "bigness" of a set) and parallel individuation (tracking of a few particular items). In humans, however, there is a standard developmental sequence by which analog magnitude and parallel individuation combine with a third faculty, sequence memorization (in particular, the count list "one, two, three, four, ... ,") to form the positive integers. Around age two, a child starts to recognize the difference between "one" and larger numbers, some months later adds "two," then "three," then makes a sudden leap to a general understanding of positive integers. Somehow, between the ages of two and four, these three faculties reliably combine to produce a breakthrough in mathematical ability.

I hypothesize that these sorts of conceptual leap are enabled by communication bootstrapping. Communication bootstrapping is used in two different ways in my

[5]Our unique language abilities emerge later.

hypothesis. First, when two specialists agree upon a signal, they may each interpret it differently, so that the signal captures a relationship between two concepts. For example, if the motor specialist and the vision specialist agree on a signal that means "release gripped object" to the motor specialist and "object goes downward" to the vision specialist, then that signal captures a causal relationship that describes how dropping works. Second, since different specialists may have very different views of the world, the process of agreement may serve as a powerful filter of ideas: if an idea can survive translation, then it is likely to represent something real about the world, rather than a fluke of one specialist's processing.

Investigation of this hypothesis is the main goal of this dissertation. There are severe obstacles (outlined in the roadmap) that must be overcome in order to allow a conclusive investigation of this hypothesis. The results of this dissertation with the broadest immediate applicability overcome two of these obstacles: first, a metric for evaluating biological plausibility and second, principles aiding in the design and integration of complex systems.

1.2 Roadmap

I am taking an exploratory engineering approach to the study of intelligence. I conjecture that if we attack an intelligence-related problem with designs constrained to within the envelope of biological plausibility, then the principles that allow our designs to work may illuminate the mechanisms that support human intelligence. The details, of course, are expected to be quite different simply because of the number of engineering decisions that must be made: it would be surprising if there were precisely one way to make something smart like a human.

This approach is ideal for exploring and taming poorly understood phenomena like communication bootstrapping. The challenge is to choose good constraints and a tractable and illuminating problem.

My roadmap for investigation consists of four stages:

1. State the "learning by learning to communicate" hypothesis precisely.

2. Define a domain for exploratory engineering that will limit the amount of wasted effort.

3. Incrementally extend the range of communication bootstrapping to support a system that learns cooperative behavior, sanity-checking each step of extension against example scenarios.

4. Determine which features of the final system are key to success, so as to be able to construct experiments to test for analogous features in humans.

I will lay out each of these stages in turn.

1.2.1 Precise Statement of My Hypothesis

We can take care of the first stage now:

> Human intelligence depends on communication bootstrapping, which allows some of the specialists that contribute to intelligence to learn to cooperate by learning to communicate.

Let me expand on this slightly, pointing out what this hypothesis does and does not say.

- I speak only about "human intelligence," and am not concerned with whether communication bootstrapping is unique to humans. I would not be surprised if monkeys and rats use it too; I would be surprised if insects did.
- I am not asserting that communication bootstrapping is the only missing puzzle piece for intelligence, only that I think the puzzle cannot be solved without it. This means that I am *not* expecting that mastery of communication bootstrapping will allow us to immediately produce systems that are smart like humans.
- I say only that it is useful for "some of the specialists" and do not assume that all links between specialists involve communication bootstrapping. I expect, in fact, that communication bootstrapping is most effective when there is a base of "lower level" connections handling things that would be awkward to express with small numbers of discrete symbols (e.g. a map between visual coordinates and arm configurations for reaching to those coordinates).
- I say "learn to cooperate by learning to communicate" but do not yet define exactly what is being communicated or how that helps learn to cooperate. These will be defined as exploratory engineering gives us a better understanding of the potential and limitations of communication bootstrapping.

1.2.2 Construct an Exploratory Engineering Domain

Above all else, defining the engineering domain is about minimizing the amount of effort that gets wasted on blind alleys and maximizing the likelihood that our engineering investments will be cumulative.

We cannot eliminate these problems entirely: exploratory engineering is always a messy business filled with bad decisions, mistakes, and unpleasant new discoveries.[6] If we knew enough to avoid these, we would not need to be doing exploratory engineering.

There are some particular pitfalls, however, that have long bedeviled the study of intelligence. I want to take particular care to limit the impact of:

- work being made obsolete by our rapidly changing understanding of biology
- work that is too disconnected from biology to have predictive power
- systems that work only in tightly controlled circumstances
- systems that have many different parameters controlling their behavior
- devices that have unexpected interactions when they are connected together
- degradation of our understanding of a system as its complexity grows
- problems tracing credit and blame to particular design decisions (including design decisions we are not aware of—i.e. bugs)

I address the first two problems, relating to biology, with a new metric for evaluating biological plausibility presented in Chapter 4. The rest are software engineering problems. I introduce two new principles aiding in the design and integration of complex systems in Chapter 5 and show how they can be applied to the remaining five problems. These contributions do not eliminate the pitfalls, but reduce the peril to a manageable level.

1.2.3 Extend the Range of Communication Bootstrapping

Communication bootstrapping has previously been demonstrated only for simple relationships under carefully controlled conditions. In this stage, we extend its demonstrated range incrementally until it supports a system that learns cooperative behavior.

We begin with an architecture taken from previous work on communication bootstrapping[4, 3]. I review this architecture in Chapter 6 and generalize it to a core architecture for this endeavor.

Step by step, we will extend the architecture to support more of what is needed for a set of parts to learn to cooperate:

[6]As is usual in scientific practice, I am not presenting my mistakes unless they illustrate important lessons.

1. use of signals to predict sensory data and vice versa
2. agreement on signals capturing asymmetric relations (e.g. causality)
3. agreement on signals capturing relations at different time scales
4. learning from streams of observations rather than a pre-digested sequence of examples
5. signals that can be used distinguish between individual objects
6. use of signals to agree on a working set of objects
7. agreement on signals involving more than one object
8. use of multi-object signals for abstraction
9. use of signals to drive actuation
10. agreement involving more than two specialists
11. prediction failure as a driver for exploratory behavior
12. agreement on signals capturing arbitration between competing models
13. use of signals for prediction beyond the immediate future
14. agreement involving specialists not directly connected to sensory observations

Chapter 3 presents a four-way intersection scenario that I use to test my system as I develop it. Chapter 7 handles the first four problems with the introduction of a self-organizing symbolic link and learning based on temporal interval relationships. Chapter 8 handles the fifth and begins work on the sixth with introduction of a mechanism for shared focus. I have sketched designs to address the remaining problems, but their implementation and integration is future work and I will not present the incomplete sketches in this dissertation.

1.2.4 Test for Key Features in Humans

This last stage is necessarily the most nebulous, since we will not know what the key features are until we have a working design. We will see, however, that even a partial design can produce testable hypotheses.

By the time we finish the extension stage, we will have a clear understanding of how communication bootstrapping might be used to implement learning by learning to communicate. We will then be able to make quantitative predictions about human behavior, brain structure, or other such biological phenomena, and use these predictions to design experiments that test for my hypothesis and rule out alternate hypotheses such as:

- Learning cooperation is unnecessary because there is always just one part that is in charge.
- Learning cooperation is unnecessary because cooperation is instinctive and does not need to be learned.
- Cooperation is learned, but at a sub-symbolic level where communication bootstrapping is not a useful model.

Although this stage is largely beyond the scope of this dissertation, Section 7.1.4 presents a testable hypothesis regarding self-organizing symbolic links.

1.3 Organization of Dissertation

The remainder of this dissertation is organized as follows:

- Chapter 2 gives an overview of the system I develop throughout the rest of the document.
- Chapter 3 gives a detailed explanation of the simulated four-way intersection scenario that I use to test my designs.
- Chapter 4 introduces a cost model that addresses the problem of connecting the investigation back to biology. This chapter is of general interest to students of intelligence.
- Chapter 5 explains the software engineering problems faced in exploratory engineering research into intelligence and presents the design process I use to tame these problems. A simple decision-making device is used as an example. This chapter is of general interest to software engineers.
- Chapter 6 reviews the previous work on communication bootstrapping, and generalizes it to the architecture that I use in this dissertation. Also discussed are standards for judging success.
- Chapter 7 explains the problems introduced by learning about relations other than equality. These are resolved with a self-organizing symbolic link and a learning mechanism based on temporal interval relationships.

- Chapter 8 explains how relations can be communicated between parts using a focus of attention and discusses the implications of shared focus for learning.
- Chapter 9 summarizes the contributions of the dissertation and discusses avenues for further investigation and potential impact.

Additional relevant material is contained in the appendices:

- Appendix A contains a summary of all the technical terms and variables I have defined. The reader is encouraged to look there if ever they feel confused about what something means.
- Appendix B contains experimental data referenced in the dissertation.
- Appendix C contains the data sheets for five key devices.

Chapter 2

System Overview

I develop my ideas in the context of a system that connects two *specialists*, vision and hearing, as they observe a simulated four-way intersection. Each specialist is building a model of its environment on the basis of these observations, in order to predict its future experiences.

Each specialist's *observations* are provided by some lower-level system that reports a list of things with properties and relations to one another. For example, a car may be reported to the vision specialist as a thing that looks like a car, occupies a small part of the visual field, is approximately red, is moving to the right, is above something that looks like a road, and so on. Chapter 3 details how observations are produced from the simulated four-way intersection.

The two specialists are connected by a communication *channel*, but have no built-in signals with which to communicate their observations. Over time, however, the specialists agree on a system of signals.

They learn to interpret the messages they send to each other with the aid of temporal relationships and a shared focus of attention. The resulting system of signals captures dynamics of the environment, allowing each specialist to interpret messages as predictions of its future experiences, even when those messages regard situations that neither specialist has ever before experienced.

2.1 From Observation to Communication to Prediction

The specialists build *models* that describe their perceived environment in terms of *objects*. Each object is a persistent bundle of sensory features. For example, the visual perception of a car is an object. It has *features* such as color, size, and shape.

The model also includes binary *relations* between objects, such as the car being above the road.

A few of the objects in a specialist's model are selected at any moment; we call this selected set its *focus of attention*. Each specialist frequently sends its partner a message describing each object in its focus and the relations the object participates in. For example, when the "cuckoo" sound turns on during a walk cycle, the change may attract the attention of the hearing specialist, which then starts telling the vision specialist that it hears "cuckoo." It also may tell it that the cuckoo sound object is to the right of loud engine sounds, to the left of a conversation, and other relations the cuckoo sound object may participate in.

When a message arrives at a specialist from its partner, the message is fed through a *relation map* that translates each message element into predictions about how the receiving specialist's model will change. For example, consider the signal that the hearing specialist uses for the feature of sounding like a "cuckoo." When this signal arrives at the vision specialist's relation map, it stimulates a causal relation that predicts that a "DON'T WALK" light will vanish and another causal relation that predicts that a "WALK" light should appear. Likewise, the signal that the hearing specialist sends to represent loud engine sounds results in a prediction in the vision specialist that a car will appear, and the "right-of" relation may be translated directly. The message that a "cuckoo" sound is to the right of loud engine sounds is thus translated into a prediction that a car will soon appear to the left of the place where a "DON'T WALK" light is being replaced by a "WALK" light.

2.2 Signal Agreement and Interpretation

The bulk of this dissertation is devoted to two problems in implementing a system that behaves in this way:

- How can two specialists agree on a system of signals for describing objects and relations?
- How can a specialist learn the relations that interpret its partner's messages as predictions about its own model?

In my system, these two problems are solved separately.

2.2.1 Agreeing on Signals

The messages that a specialist sends describe objects and relations using *symbols* and *inflections*. Symbols encode features; inflections are markers that are placed on symbols in order to tie features together into objects and describe the relations linking those objects. Messages of this sort are explained in Chapter 6 and discussed in more detail in Section 8.1.

The specialists send messages to one another over a channel composed of a vast number of arbitrarily connected simple communication paths. A symbol is encoded as a sparse subset of the paths in the channel and detected when there is coherent activity on that subset; an inflection is encoded as a sparse pattern of pulses in the activity for a symbol.

Initially, the two specialists have no agreement on encodings for symbols or inflections. They establish these agreements by babbling randomly on the channel when it is idle, rapidly coming to an agreement on the encodings. Quick convergence is a consequence of the sparseness of the encodings. As agreement forms on individual symbol or inflection encodings, they are added to a pool of pre-agreed signals that await allocation and use. The agreement process is described in Section 7.1.

For example, when the hearing specialist needs a signal for "cuckoo," it allocates an unused symbol from the pool that have been agreed on in advance. Because this symbol has been agreed on in advance, the vision specialist can instantly recognize it. If it already knows the inflections on the symbol, it can interpret its relationships to other symbols. The vision specialist does not, however, have an interpretation for the symbol: when the symbol is fed through the relation map, no predictions result.

2.2.2 From Signals to Predictions

A specialist learns to make predictions by observing temporal relations between the contents of its focus of attention and the sequence of messages it receives from its partner. The temporal relations provide evidence for causal relations that connect an incoming symbol to a feature or an incoming inflection to a relation in the specialist's model. These causal relations are accumulated in the relation map that translates incoming messages into predictions.

A *shared focus* mechanism ensures that the sequence of incoming messages is usually related to the contents of the receiving specialist's focus of attention. For example, when a "DON'T WALK" light appears and attracts the attention of the vision specialist, the hearing specialist will start listening for a sound in that direction, even before the "cuckoo" sound begins. The shared focus mechanism is described in

Chapter 8.

The sequence of messages is reinterpreted as a collection of intervals when each symbol is present in the message. Relations between these intervals and intervals when a feature is present in the focus of attention are incrementally classified using Allen's time relations. These time relations are then heuristically interpreted as positive, negative, or neutral examples regarding each possible causal relation. The examples nudge a simple strength estimate up or down, so that when positive examples dominate, the relation is accepted and used for interpretation. Learning from message sequences is described in Section 7.2.

For example, when a walk cycle begins, the hearing specialist starts sending messages including the signal for "cuckoo." The vision specialist tracks the time relations between the interval when messages containing "cuckoo" are present and the intervals when each of the features in its own model is present in the focus of attention. One of these features, that an object looks like a "DON'T WALK" light, disappears as the messages containing the signal for "cuckoo" begin arriving. This time relationship is evidence in favor of the signal for "cuckoo" predicting that a "DON'T WALK" light will vanish. After a few experiences of this sort, enough evidence has accumulated, and the vision specialist begins predicting that "DON'T WALK" will disappear when it hears the signal for "cuckoo."

2.3 Summary

We thus have a system of two specialists that create agreement on a pool of signals during their idle time. They learn to interpret the messages they send to each other with the aid of temporal relationships and a shared focus of attention. Together, these result in a system of signals that captures dynamics of the environment, allowing each specialist to interpret messages as predictions of its future experiences, even when those messages regard situations that neither specialist has ever before experienced.

Chapter 3

Example Scenario

As I go about my investigations, I need a realistic scenario to check my work against and to provide concrete goals to aspire towards. The point of the scenario is not to measure success, but to use as a sanity check that my designs are at least behaving as I expect them to. The work presented in this dissertation will hit only the lowest goals in the scenario, but a fully developed architecture completing the roadmap presented in Chapter 1 should satisfy all of the goals.

Our example scenario for the communication bootstrapping system will be two specialists (vision and hearing) jointly observing a simulated four-way intersection. This complex scenario will allow the same scenes to be understood at many different levels, giving us a series of increasingly difficult benchmarks to test the integration of the two specialists.

I will not, however, control the scenario to aid learning—no gradually increasing complexity of situations or other such pedagogical devices. A baby is not granted such easements, and neither will the systems I build. Instead, the system should be able to ignore any complexity that it cannot yet hope to comprehend.

3.1 Simulated Scenario

First, a note regarding my choice to work in simulation, rather than with real-world data. While real-world data has the advantage of making arguments for validity easier, it introduces serious problems: one must collect and manage a large library of data, signal processing of audio and video data is difficult and computationally expensive, and testing whether a system is behaving correctly requires careful human annotation of the data.

Working in simulation will allow me to avoid these problems by taking a generative approach to the scenario. There is, of course, the danger that bugs in the simulation

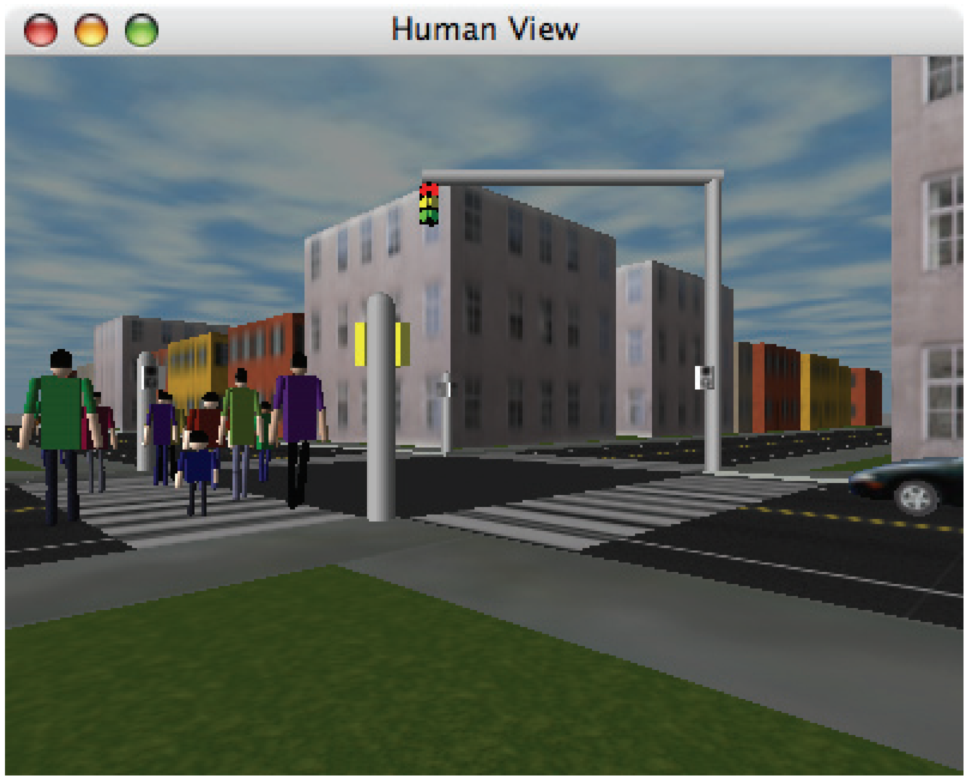


Figure 3-1: Screenshot of the four-way intersection simulation: child pedestrians surge as school lets out in the afternoon.

will cause the simulation to be missing important facts about the real world. I work to minimize this danger by building my scenario using a pre-existing simulator, the Open Dynamics Engine[36]. The Open Dynamics Engine is a free open source physics engine with an active development community, which has previously been used for dozens of projects including published research in robotics (e.g. [25], [16], and [44]). Likewise, observations are produced from the simulation using OpenGL, a standard 3D graphics library, and OpenAL, its companion 3D sound library.

The scenario I have chosen is a four-way intersection with a single traffic light in the middle (Figure 1-1 and Figure 3-1 to 3-4). There are four basic types of object in the scenario:

- **Background:** large fixed objects: streets, sidewalks, grass and buildings. There is also a day/night cycle that affects the lighting and the color of the sky.

- **Traffic Light:** A single three-color light with four faces, suspended on a horizontal pole crossing the intersection. The vertical poles on the corners (one of which supports the light pole) contain non-visible walk-buttons that pedestrians

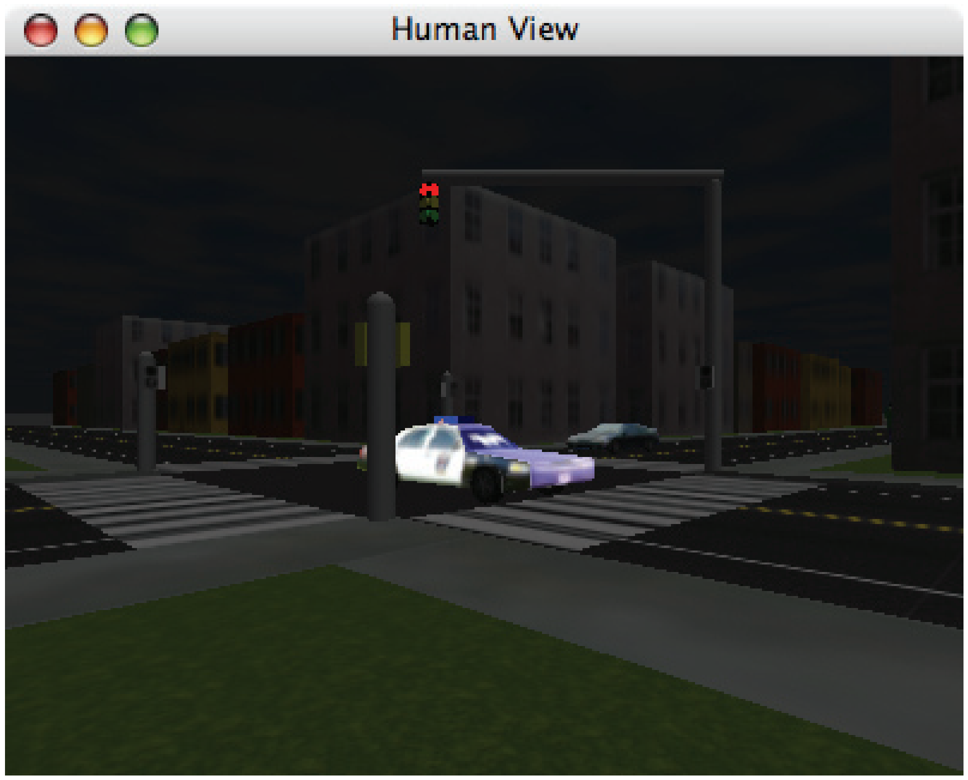


Figure 3-2: Screenshot of the four-way intersection simulation: a police car with flashing lights exercising right-of-way at night.

push. The vertical poles also hold the boxes that show “WALK” and “DON’T WALK” signals.

- **Cars:** Cars of various different colors and types (e.g. sedan, pickup, SUV, van). Three types of emergency vehicles are included: a police car, an ambulance and a tow truck. All have lights that can flash: blue for the police car, red for the ambulance and yellow for the tow truck. The tow truck is a wrecker that can carry cars on its flat bed. Car windows are opaque so that no driver or passengers can be seen.

- **People:** Adults and children, travelling alone or in groups.

The stoplight follows the ordinary green-yellow-red light cycle, except when a pedestrian pushes the walk button. Pushing the walk button results in a walk cycle after the next East/West yellow light. When the walk light is on, the boxes make a “cuckoo” sound (one of the standard audible walk signals here in Massachusetts).

Cars show up at the intersection with random likelihood dictated by time of day—low when people are sleeping, medium at mid-day and evening, and high during rush hours. Most cars go straight and a few turn left or right. For the most part the cars

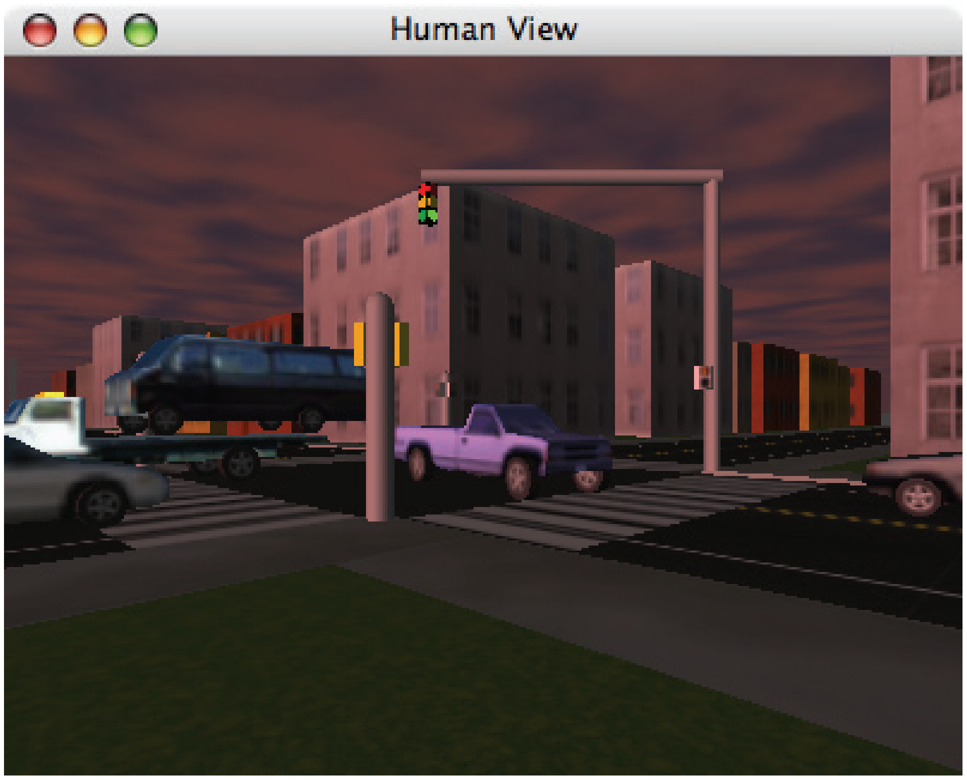


Figure 3-3: Screenshot of the four-way intersection simulation: tow-truck at dusk, picking up the first of two cars in a T-bone accident.

obey traffic laws: drive on the right, stop for red lights, yield on a left turn, right turn on red, and yield to pedestrians. Cars will usually run yellow lights and will occasionally break other traffic laws. Cars also have the goal of avoiding accidents and will usually brake to avoid them. Cars make different noises when idling and driving, and squeal their brakes when they slow too quickly. Cars honk when something forces them to avoid an accident or when another car hesitates. Cars yield for police and ambulances with flashing lights, who do not respect traffic laws. Cars that get in an accident make a loud crashing sound and lose the ability to drive.

Pedestrians arrive randomly, much the same as cars, except they may appear singly or in groups. The pedestrian daily schedule is the same as that of cars, except that their rush hour is a "school rush" in the morning and early afternoon where there are many child pedestrians, and there are also many pedestrians in the evening. Like cars, every pedestrian has a goal destination, and most know to push the walk buttons to achieve it. Pedestrians will also sometimes change goals when they meet a friend. Pedestrians usually obey traffic rules, though they will usually jaywalk when no car is coming. Pedestrians also try to avoid collisions, usually going around each other and running away from oncoming cars to the nearest sidewalk. Walking makes

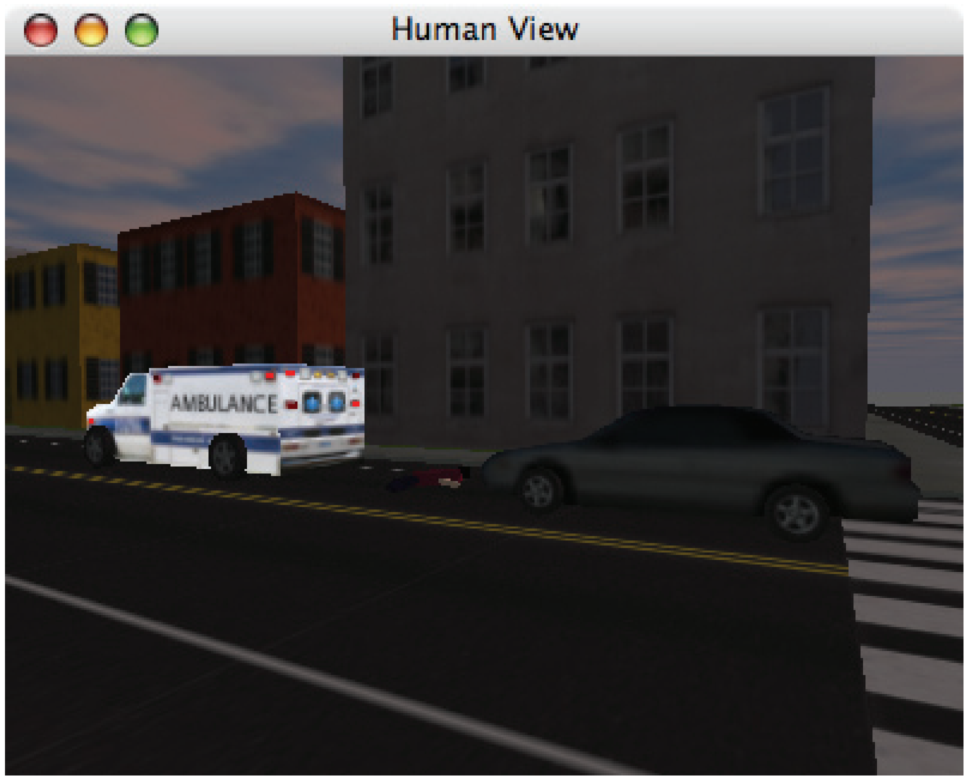


Figure 3-4: Screenshot of the four-way intersection simulation: close-up of an ambulance picking up an injured pedestrian at the scene of an accident.

a faint sound. Pedestrians also usually talk when they are in groups and will yell at cars they run away from. Pedestrians that meet with an accident (usually being hit by a car) scream and fall to the ground unconscious.

When an accident occurs, the participants usually stop moving (though sometimes there are hit-and-runs), and an emergency vehicle (ambulance for pedestrians, tow truck for cars) is summoned to clear up the accident. This is essentially a garbage-collection function, and since failure of the garbage collector is likely to cascade badly, all other pedestrians and cars stop moving and the emergency vehicle cheats in its physics interactions to avoid getting wedged.

3.2 Two Senses

We may think of our system as an infant sitting on the porch of a building at the corner, staring in fascination at the world going by.[1] It sits a little way off the sidewalk, facing across the intersection, observing with two senses: vision and hearing. We will be looking at what is learned by communication bootstrapping between the specialists

[1]Of course, we are going to leave it out all night watching cars, which is a bit inhumane.

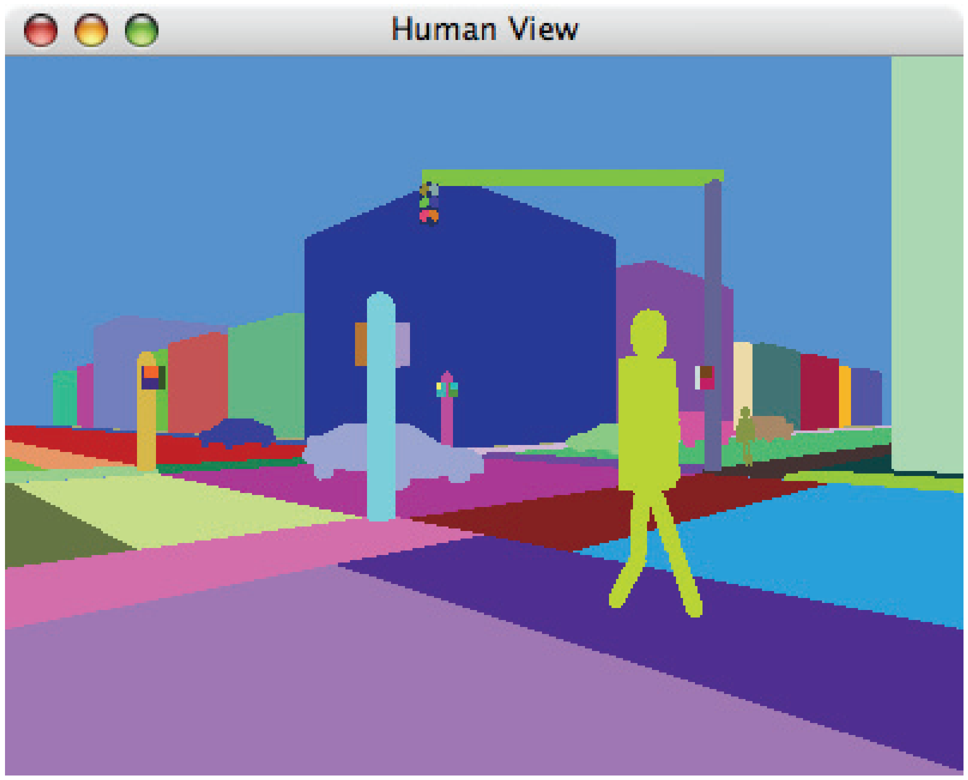


Figure 3-5: Visual observations from the simulator are obtained using a false-color image that fakes the process of segmentation and recognition.

for these two senses. To avoid the interesting complexities of actuation, the system is not able to move.

Each sense delivers observations scraped from the simulator at a designated sampling rate (the rate can be different for the two senses).

For the hearing specialist, each sound source is an object carrying three types of feature:

- **Type:** one or more of 13 values: engine, idling, driving, horn, cuckoo, screeching brakes, talking, yelling, walking, etc.
- **Direction:** relative to the direction of gaze: 8 overlapping 60-degree sweeps at approximately 45-degree intervals.
- **Loudness:** seven overlapping 15 db ranges from 40 to 100 db at approximately 10 db intervals. Below 32.5 db is unheard, above 100 db registers as 100 db.

For the vision specialist, each visible object is assumed to have been segmented from the image by a low-level vision system and recognized as belonging to a particular category. Segmentation and recognition is faked using a false-color rendering of the scene (Figure 3-5). Each object has four types of feature:

- **Type:** one or more of 28 values: car, sedan, person, adult, child, light, pole, etc.

- **Color:** 8 overlapping color regions: red, yellow, green, blue, magenta, cyan, dark, and bright.

- **Size:** ten overlapping 10 degree ranges, from 0 to 80 degrees, at approximately 8 degree intervals. Size is the number of degrees of vision covered by a circle containing as many pixels as the object does.

- **Motion:** relative to the direction of gaze: 8 overlapping 60-degree sweeps at approximately 45-degree intervals (e.g. up, down-left).

Objects also have relations to other nearby objects. Stationary objects relate only to those they contact visually, mobile objects relate to other nearby mobile objects as well. There are seven relations: contact, above, below, left, right, in front, and behind. These relations are computed using a rough directional nearest-neighbor calculation.

3.3 Measure of Success

As we extend the capability of communication bootstrapping systems, they should be able to extract increasingly subtle structure from the four-way intersection scenario. The goals for learning are arranged in a series of increasing complexity. As we extend the range of communication bootstrapping, they should be achieved approximately in sequence.

1. **Common Cross-Sense Binary Relations:** The simplest to learn, these show a basic ability to find correlations between senses. Examples: engine sound is a super-set of car, walk signal equals cuckoo sound.

2. **Rare Cross-Sense Binary Relations:** These demonstrate the ability to direct learning to more subtle correlations. Example: screeching brakes stops object motion.

3. **Simple Patterns:** These demonstrate that the ability to find agreement capturing multi-object relations. Examples: yellow light changes to red light, a car approaching a pedestrian causes a horn sound.

4. **Complex Patterns:** These demonstrate the ability of multi-object relations to work with patterns involving more than one relation between objects. Example:

contact between a car and person makes the person move fast and scream, a pedestrian at the corner causes a cuckoo sound.

5. **Abstract Patterns:** These demonstrate that patterns can be incorporated into other patterns. Example: a pedestrian at the corner causes a walk cycle.

6. **Patterns Incorporating Separated Events:** These demonstrate that a pattern can be learned despite distractors (possibly incorporating abstractions). Example: an accident causes an ambulance to appear.

7. **Multilevel Abstraction:** These demonstrate that learning can take place when not closely tied to observations. Examples: accidents are likely during school rush hour, running a red or jaywalking can cause an accident.

8. **Exceptions:** These demonstrate that it is possible to learn patterns that manipulate other patterns. Examples: police cars do not stop at red lights when their own lights are flashing, jaywalking is safe when there are no cars coming.

In this dissertation, I attain only the first two goals.

Chapter 4

Relationship to Biology

We can learn about human intelligence by using a model of hardware and development costs, ignoring almost all the details of biology. The basic argument is that neither the gross anatomy of the brain nor the behavior of individual cells nor the behavior of the whole poses sufficient constraint on the algorithms that might implement intelligence, but that the process of engineering an intelligence under this cost model poses similar challenges to those faced by a human growing from a single cell to an adult. This will allow us to explore organizational ideas freely, yet retain confidence that when a system works, the principles allowing it to work are likely to be similar to those that allow human intelligence to work.

4.1 Neuroscience Is Not Enough

Although our current knowledge of biology tells us a great deal about how human intelligence *cannot* be structured, we know little about the algorithms that actually make our intelligence work.

Let us start big, with our knowledge about the anatomy of the brain. By studying tissue structure, injuries, and intensity of blood flow, we know quite well the gross location of broad functions, like "vision" and "motor control." Other things, like "memory" or "language" are more slippery, with many different places involved in different ways. Knowing how much activity is going on in what places, however, tells us nothing about what computations are being executed or their relative importance.

To see why, consider a modern computer like the laptop I am using to write this. Modern computers spend almost all of their cycles either doing nothing or putting pictures on the screen. No matter its importance to my life, editing this document occupies effectively zero processing power[1], and storing the text takes only a few millionths of the available storage.

If you scanned my computer's components for activity, correlating carefully with when I hit various keys, you could probably discover the hardware controlling the keyboard and hard drive. When you try to learn how the spell-checker works, however, you learn only that it depends on interactions between the processor, memory, the hard drive, the keyboard, and the graphics coprocessor. You may investigate further, determining that when interaction with the hard-drive is interrupted, no spell-checking occurs at all, while disrupting interaction with the graphics coprocessor allows spell-checking to run, but ineffectively. Still, the information about how it actually works is obscured by all the other things going on in the background.

We are in much the same position concerning the anatomy of the brain: we know quite a bit about what places are involved with various sorts of activity. What we do not have is a clear understanding of what computations are being executed in these places. The few exceptions are instances like early vision, where the behavior is very closely coupled to sensory input, or like pituitary gland activity, where the behavior is highly specialized. Our knowledge of anatomy tells us virtually nothing about how the parts work together to produce human intelligence.

What about the other end of the scale, the behavior of individual cells? We know quite a bit about the various types of cells in our brains, and especially about the neurons where it appears that most of the computation takes place. We can even stick probes into brains and measure the behavior of some individual neurons. Knowing what an individual cell does, however tells us nothing about the larger computation in which it is participating.

Again, consider trying to figure out how my spell-checker works. Sticking delicate probes into the guts of my poor laptop, you discover that there are lots of tiny parts. Although they all share common features, different parts have specialized to perform different functions. Some parts switch on when their inputs are active, while others switch on when their inputs are not active. Some parts store values and repeat them again later, while others relay values across long distances. When you apply this knowledge to the spell-checker, however, you again come up blank. Each time you run the spell-checker, different parts of the memory are used, and although the same parts of the processor are used each time, the pattern of use is always different—the rest of the work the computer is doing keeps interfering in different ways. With some hard work, you can find the parts that test for equality, and show that when there are errors they behave differently from when there are none. Still, the information

[1]I use `emacs`.

about how it actually works is obscured, both by the rest of the activity going on in the computer and by the fact that the parts are so generic that they could be doing anything.

We are in much the same position concerning the behavior of neurons. We know quite a bit about what sort of computation an individual neuron can do, what a collection of interconnected neurons can compute, and how long a computation would take. The problem is that a large collection of neurons can potentially compute just about anything, and we know very little about what they actually are computing. Because of this, knowing about individual neurons tells us virtually nothing about how large numbers of neurons might work together to produce human intelligence.

Finally, what about comparison to human behavior? By measuring reaction times, relative preferences, and other objective features of cognition, cognitive science has produced a great wealth of information about the range of behavior produced by actual human intelligences. Knowing what a complete human intelligence does, however, tells us nothing about what its components are or the types of functions they compute.

Once more, consider my spell-checker. Sitting down at the keyboard, you discover that hitting "Escape-$" causes it to tell whether a word is correct—a fine stimulus response! Comparing across computers and applications, you discover that the spelling reflex is universal (except for a few diseased individuals) and document a range of stimuli that activate the spelling reflex in different contexts, including "Command-:" and "F7." Some further analysis reveals universals and range of variations: for example, the first suggestion to correct "teh" is always "the," but the number and nature of other suggestions varies widely. Combining this with previous work on component activity, you can make some pretty good theories about how the keyboard and graphics coprocessor contribute to spell-checking, but the contribution of the processor and memory is still a mystery. For that, you start setting up much more precise experiments, timing how long spell-check takes to start and how long between words, varying the size of the document, the distribution of errors, whether music is playing at the same time, and so on. You learn many facts, including that longer documents take longer to check, and that music can slow the process down, presumably by competing for a shared resource. But you end up stymied still, because both the memory and the processor are always interacting with one another, and every manipulation you apply still involves both equally.[2] In the end, all you know is that processor and memory work together to spell-check, but nothing about how they do so.

Measurements of human behavior have the same problem: we can measure the

[2]This continuous interaction is likely if the dictionary does not fit in the processor cache.

behavior of the whole system, and compare the results of different experiments in order to understand which parts are involved and how the behavior changes if a part is damaged. What we do not know is how the parts interact with one another to produce the observed behavior. Even apparently modular behavior might involve small but important interactions between many components. The exceptions are phenomena like optical illusions, where the behavior appears to involve very specific resources and not depend much on interaction. Moreover, we do not know how sensitive the observed behavior might be to small flaws in our models: an almost correct model missing one critical link may produce a much worse fit than a model that is completely incorrect, but well-tuned for a particular domain. Because of this, measurements of human behavior tell us virtually nothing about how the parts work together to create the behavior we observe.

4.2 Development Is Like Engineering

I have argued that if I were to build a system that conforms to what we currently know about brain anatomy, individual neurons, and human behavior, then I have no reason to believe that it will be smart like a human. Moreover, if I build something that is not smart like a human, I do not know enough about how the specialists in my system should interact to figure out what I should change in order to make it smarter.

Again, here is the challenge I face as an engineer: I need to build many different specialists and connect them together to form something that is smart like a human. This is hard because the specialists are complicated, so I need to work on them at different times, and how to specify the goal is unclear, so I will make many mistakes in building them. Can I even dare hope that such a disjointed and flawed process can produce specialists that learn to work together and be smart like a human? It seems to need a miracle.

Such a miracle has happened not once, but billions of times, every time a human grows from a single cell to an adult.[3] First, because the various specialists contributing to human intelligence are distributed across long distances (centimeters are huge in cellular terms!), the details of each distant specialists are likely to develop largely independently. Our problems in engineering an intelligence also lead to specialists being developed largely independently, though for the engineer the separation is due

[3]This is why I am discussing development rather than evolution: humanity evolved approximately once, so chance and special circumstances might be much more in play there. Evolution and development are intimately related, however, so evolution is by no means excluded from discussion.

to the complexity of the problem rather than physical distance. Second, there is a large amount of variation during the development of the brain, meaning that a specialist cannot make many assumptions about the structure of the other specialists it must work with. In engineering, independent development of specialists also leads to variation, because we do not understand the problem well enough to prevent our conception of the system from changing over time or between engineers.

Development overcomes these problems, regularly integrating the various specialists into a functional intelligence. Even most mentally ill or developmentally disabled people are basically functional examples of integrated human intelligence. It is just that our standards are very high and we notice even small differences in behavior. Yet we engineers have not yet been able to do the same as we struggle to build an intelligence.

I propose that there are organizational principles exploited by biology that we have not yet recognized and tamed to our purposes. What I want, then, is a sandbox where experimentation can help us identify principles of this type. The way that biology does it may not be the only way—indeed, I would be surprised if it were—but I will hunt for a solution that is biologically plausible because I know that at least one such solution exists. As an additional benefit, if the systems we build are biologically plausible, then it is reasonable to guess that the organizational principles that allow us to integrate the specialists in our systems may be related to the principles that allow human development to work.

4.3 Hardware and Development Costs

What constraints shall I set for myself? On the one hand, I want simple constraints so that I can explore organizational ideas freely. On the other hand, my systems must not be biologically implausible.

I have chosen to judge plausibility with a biologically-inspired model of hardware and development cost. This model will allow me to judge an individual device without knowing how it will be employed, by measuring the asymptotic cost of the device as its capacity increases.

This measurement, in turn, tells us what constraints there are on our ability to use the device as part of a broader explanation of intelligence. If a device is costly, we can only use a few small-capacity copies. If a device is cheap, then we may be able to use vast numbers of small instances or a few instances with vast capacities. Thus, for example, a device involved in handling words should have a low cost per word, because we need to handle thousands of different words, but a device managing our response to hunger could be costly because hunger is one of a very few direct survival urges.

I will measure two types of cost familiar to computer scientists: time complexity and space complexity. Because I am making an analogy to biological systems, however, I will measure these costs both for a mature system and for its development. The relative importance of these costs differs from what we have become accustomed to on digital computers.

- **Time Complexity:** Biological systems are much slower than silicon systems, operating at frequencies ranging from kilohertz to millihertz depending on the mechanism, so time is a more precious resource. On the other hand, there is the opportunity for massive parallelism. The available time is generally long for development tasks, ranging from months in the womb to years of childhood, but may be short for responses during mature execution, on the order of seconds or less.

- **Space Complexity:** Space complexity refers to the amount of hardware comprising the mature system and the size of the program encoding its development. For the mature system, this generally means neurons, of which there are on the order of 100 billion in the brain (approximately 20 billion in the neocortex), each connecting to an approximate range of 100 to 10,000 other neurons.[4] For development, human DNA contains approximately 3 billion base pairs—a little under 1 gigabyte of data.

Finally, I will assume that that devices are not perfect. Current fabrication techniques for silicon computers are so accurate that perfection is a reasonable assumption in many cases: errors during manufacturing can usually be detected, and the device discarded, while errors during execution are corrected using parity checks and the like. This sort of perfection is not found in biological systems: even the simplest parts vary during development and misbehave occasionally during execution.

Using this cost model also helps insulate my sandbox from our rapidly changing understanding of biology. These changes are much more likely to adjust the relative cost of devices than to entirely rule out a devices.

4.4 Cost Assumptions

Based on my own highly incomplete knowledge of the field, and particularly on recent work in synthetic biology (see, for example [43], [24], [42], and [31]), I make a few

[4]Hardware also implies metabolic cost.

assumptions about the cost of devices. While my knowledge leads me to believe these are reasonable upper bounds, it is possible that I am mistaken. If that turns out to be the case, then the devices I present will not be invalidated, but need only to have their costs adjusted.

Simple Programs While the computing power of a single neuron is still unclear, a precisely constructed network of neurons can compute just about anything. For a simple program with no looping or recursion, I assume that execution, hardware, and development costs are all proportional to the number of operations in the code plus the number of bits in the data types it manipulates.

When there is looping or recursion, I measure the complexity by unrolling the loops to their maximum length (unless they can be trivially parallelized) and expanding the recursions to their maximum depth.

I assume that variation during development will cause a small percentage of program instances to simply fail. Errors in execution will result in the program occasionally failing to produce a result.

Communication Paths To create a *communication path* between two devices, some sort of signal must be sent to guide the growing path from its source to its intended destination. Since these signals are likely to be diffusing chemicals, two signals can only be distinguished if they use different chemicals or are separated in space or time. On the other hand, a signal can guide many paths at once to the same destination.

I abstract this process with the following path creation operation: given a set of source devices and a set of destination devices, create a path from each source to somewhere in the destination set. Note that I place no upper limit on number of paths connected to a device, despite the fact that neurons appear to have a fan-out limited to around 10,000 connections. This is because a device is not one, but potentially many neurons, and the fan-out can also be boosted exponentially by adding intermediate stages (e.g. three stages is 10^{12} connections).

The encoding cost is proportional to the number of simultaneous nearby path creation operations times the number of bits to be transmitted simultaneously on a path, and the development time is proportional to the maximum distance between source and destination devices.

Once created, a path takes hardware proportional to the number of bits that can be transmitted simultaneously. It takes a constant amount of time to send each set of bits, and they arrive at the other end of the path after a delay proportional to its length.

I assume that variation during development will result in a small percentage of missing or extra paths. Errors in execution add a small amount of noise to the bits being transmitted.

Sets of Devices Filling an area with copies of a device is fairly cheap: since they develop and execute independently, both development and execution can be done completely in parallel.

During morphogenesis, coordinate systems are established with chemical gradients and used to select regions where specific development programs execute[9]. This adds at most a constant encoding cost to the encoding cost for a single part. Since the relevant coordinate system may be established when the region is still very small, I will assume that creating a set of devices adds only a constant time cost to the time cost for developing a single device.

Once created, the hardware cost is equal to the number of devices in the set times the complexity of a single device, plus a small constant. The execution time for a set of devices is equal to the execution time for a single device plus a small constant.

I will assume that variation during development will result in a small percentage change in the size of the set. Inclusion in a set adds no new errors to execution: the only errors are the errors of the individual devices.

I will also assume that, for only a constant additional cost to encoding and development time, each device in the set can have connections to all others within a small fixed radius, connecting the set into a mesh-like network.

Chapter 5

Design Process

In this chapter, I will explain my process for engineering devices for use in building an intelligence. I introduce two new ideas, *dossiers* and *failure simplification*, which complement one another in the design process. Together, they allow us to understand and manage the behavior of a device over a large range of conditions and configurations. Devices engineered with this process can be connected together to form larger systems that behave well across a wide range of conditions, even when some of the devices within them are misbehaving. We can thus ensure that broad competence and flexibility are preserved as a system gets more complex.

5.1 Two Powerful Ideas

A *dossier* is a visualization of a device's behavior across a broad range of conditions and configurations. The dossier helps us engineer a device by reducing the potentially vast complexities of behavior down to a series of pictures that let us see at a glance where there is desirable behavior, where there is misbehavior, and where the device transitions between the two.[1] When we create a dossier, it also shows us how robust or fragile a device is: the narrower the regions of desired behavior, the more fragile the device. By making it easy to see fragility, the dossier becomes a vital part of the debugging process and helps us to avoid creating devices that later surprise us by being inadequate for our needs.

We then use the information in the dossier for *failure simplification*, a way of limiting the impact of a device's misbehavior on the other devices it interacts with. First, we must recognize that it is often not possible to prevent misbehavior, and that the transitions between desirable behavior and misbehavior are likely to contain many

[1]This is much the same role as Poincaré sections serve in the study of dynamical systems.

interesting, hard to understand, and hard to evaluate complex behaviors. Failure simplification is the tactic of changing the type of possible misbehaviors in order make the device easier to understand and cope with, as well as to eliminate the hidden interesting behaviors in the transition.[2] This often involves some sort of pre-emptive failure, and is generally "paid for" by shrinking the range of desirable behavior.

These two ideas, dossiers and failure simplification, are a complementary pair. Dossiers help us understand and improve a device's range of behavior. That understanding then allows us to use failure simplification to improve its interactions with other devices. Together, they let us incorporate the devices into larger systems that have desirable behavior over a broad and predictable range of conditions.

In the rest of the chapter, I flesh out these ideas and explain how I apply them to the creation and use of devices. First, I take a moment to explain what makes this process different from other engineering processes. In the next section, I explain how to design a device and produce a data sheet capturing the important information about how to use it. Finally, I explain how to use well-specified devices to create a larger system and predict its behavior.

As I explain my process, I use my simplest device, a *codetector*,[3] as a running example. The purpose of a codetector is to make a decision on whether to accept or reject a proposal, given a stream of evidence for and against the proposal. Both incredibly simple and frequently used in my designs, this device will be a good example to illustrate the design process.

5.2 What Makes This Process Different?

Why are current engineering techniques insufficient to handle the problems of designing an intelligence? Why do we need to introduce these new techniques? The basic problem is that an intelligence needs to both be autonomous and to behave reasonably across such a broad range of conditions.

Systems that behave reasonably across a broad range of conditions are needed in many places in our increasingly uncertain and interconnected world. Some, like earthquake-resistant buildings or peer-to-peer networking are susceptible to ordinary engineering techniques because they address a well-defined and narrow goal. Others, like the Internet or business enterprises[34], depend on human intervention in their response to unusual conditions.

[2]Failure simplification often effectively appears in the design of the best systems, but has generally been a matter of art rather than routine.

[3]short for "coincidence detector"

A more direct comparison can be made between conventional software design and the problems we will face in designing an intelligence. Here, the key differences are:

- In conventional software design, the design of a device tends to strongly control the conditions under which it is employed. In designing for an intelligence, the conditions are largely uncontrolled, and a device must be designed to behave well over a large range of conditions.

- In conventional software design, the specification tends to draw a sharp distinction between correct and incorrect behavior. In designing for an intelligence, the specification tends to be a set of sometimes contradictory qualitative behaviors, which can be satisfied to varying degrees.

- In conventional software design, a device is expected to comply with its specified interface, and it is a bug if it does not. In designing for an intelligence, I will ask instead the range of conditions under which the device behaves reasonably, and consider a narrow range a critical design flaw.

- In conventional software design, the result of violating a device's assumptions or interface specification[4] is undefined and may result in arbitrary behavior. In designing for an intelligence, I will map out the dysfunctional behaviors of a device and provide instructions for limiting the impact of such behavior.

Of course, as conventional software projects get very large, their sheer complexity starts to make them more like building an intelligence, simply due to the number of different devices in the system, the inevitability of bugs, and the version changes both within a piece of software and in the rest of the system it interacts with.

Programmers have been wrestling with the problems of complexity for a long time. Fred Brooks' famous "No Silver Bullet" paper[6], published twenty years ago, does an excellent job of laying out the ways people hoped to overcome programming complexity, and argues that none are likely to succeed. Twenty years later, a great deal of progress has been made in all the many fields he touches on—program verification, high-level languages, and expert systems, to name a few—and yet his thesis still essentially holds. Despite all the work, current software engineering techniques are essentially unable to stand up to the challenges of building devices for an intelligence.

Two recent research thrusts should be noted, however, as at least entering the same problem space as my work. First is the autonomic computing field pioneered by IBM[20], and its relatives in Self-* computing (e.g. [13]). Like my work, these are

[4]counting defined exceptional cases as part of the interface specification

also interested in resilient and adaptive behavior, but differ in the specifics of their approach.

Second is failure-oblivious computing[32], which takes a more radical stance and declares that errors should in fact be generally ignored: instead of crashing or raising an exception, problems just result in a “safe” default being returned. While this may help keep a program from crashing, it is insufficient for our needs because it lacks a way of directing development from merely not crashing to actual desirable behavior.

5.3 Creating Devices and Data Sheets

A device and the data sheet that characterizes the device are created in an intertwined process of iterative design. Starting with an *interface specification* that says what we want the device to do, we describe the *mechanism* by which the device attempts to fulfill those desires. We then generate a *dossier* characterizing the device's behavior across a broad range of *conditions* and *configurations*. If we are satisfied with what the dossier tells us, we use it to create a *usage specification* for the device; otherwise, we change the mechanism to address problems revealed by the dossier and try again.

The data sheet for a device contains all of this information, capturing both the completed device and intuitions stemming from its engineering. Here is the data sheet template for a device created with my design process:

- **Interface Specification:** Three aspects of the relationship between a device and its environment:
  - **Conditions:** the external environment affecting the device
  - **Range of Behavior:** the set of observable actions that a device may exhibit
  - **Desirable Behavior:** criteria for determining whether a device's actions are appropriate
- **Mechanism:** how the device implements its interface
- **Configuration Parameters:** adjustable values controlling the behavior of the device
- **Dossier:** Analysis and experimental surveys of the device's behavior.
- **Usage Specification:** Instructions on how to connect this device to other devices:

  - **Configuration Policy:** Given a range of conditions, what configurations are expected to produce desirable behavior over a large portion of the range?
  - **Limiting Conditions:** Given a configuration, what conditions will produce mostly misbehavior and should therefore be avoided?
  - **Failure Simplification:** Assuming that misbehavior will occur, how can its impact on other devices be limited?
  - **Cost:** Time and space costs for development and mature operation, plus expected types of variation and error.

Let us now examine each stage of the process in turn.

5.3.1 Interface Specification

We begin with an interface specification, describing how a device should interact with its environment. There are three parts to this specification:

- **Conditions:** what aspects of its environment are outside a device's control? Examples are input from other devices or the structure of a network connecting devices together.
- **Range of Behavior:** what does a device do that other devices can observe? This can be messages sent, data placed where it can be read, or any other observable activity. The passive phrasing reflects the fact that a device should not assume its behavior will produce any particular response from other devices.
- **Desirable Behavior:** what do we want a device to do? These are generally qualitative requirements, and often contradictory. Later in the design process, we will determine how to measure satisfaction quantitatively, and how to manage the tension between contradictory requirements.

At this stage, the device is a black box, with nothing yet said about either its internal workings, nor the conditions under which its behavior is desirable.

Example: Codetector Interface Specification

- **Conditions:** A codetector receives a stream of evidence as input (Figure 5-1). The stream of evidence must have been preprocessed into two values: **positive** supporting acceptance and **negative** supporting rejection. When the raw evidence is equivocal, preprocessing should simply discard it, reducing the number of examples in the stream.

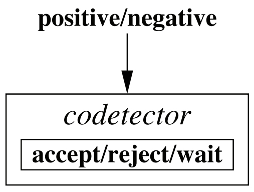


Figure 5-1: A codetector uses a stream of evidence carrying values **positive** and **negative** to set its decision to **accept**, **reject** or **wait**.

- **Range of Behavior:** A codetector exposes one value, its **decision**, As evidence arrives, the codetector examines it and sets its decision to one of three values: **accept**, **reject**, or **wait** (meaning that more evidence is needed for a decision). The decision always has one of the three values, initially **wait**.

- **Desirable Behavior:** A codetector is behaving as desired when it does all of the following:

  - decides **accept** or **reject** quickly
  - makes its decision in accordance with the evidence presented
  - does not waver in its decision[5]
  - can change its decision in the face of changing evidence

  Notice that the requirement for non-wavering decisions is opposed to the requirement that decisions be able to change. The major benefit of applying my design process to the mechanism will be a prescription for how to manage the tension between these two requirements.

5.3.2 Mechanism

Creating a mechanism that fulfills the interface specification is an ordinary exercise in programming ingenuity.

Example: Codetector Mechanism

The user of a codetector assumes that evidence relevant to the proposal is relatively sparse, and that therefore successive pieces of evidence should be treated as independent. Given this, repeated positive or negative pieces of evidence are unlikely to be the result of chance, and a short streak should result in a decision to accept or reject.

[5]This is important even if the evidence is not decisive, because presumably some other device will be acting on the decision.

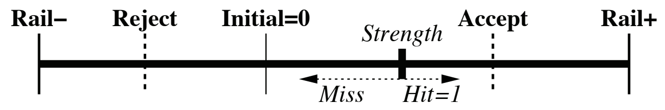


Figure 5-2: A codetector uses a simple estimator to track the strength of a hypothesis. Positive evidence increments the strength, negative evidence decrements it, and when it is past the *accept* or *reject* threshold the evidence is considered decisive. Since relationships may change over time, the strength is bounded by rails to prevent it from moving irretrievably far from the threshold.

Although the independence assumption is often violated, we will see in Section 5.3.4 that behavior can remain desirable even with significant violation.

The actual mechanism is extremely simple. The heart of a codetector is an incremental estimator of decision strength (Figure 5-2). The estimator's state is a single number, its current strength estimate, which starts at 0 and is adjusted each time evidence arrives: on positive evidence, it rises by 1; on negative evidence, it falls by the constant *miss*.[6] The larger that *miss* is, the more skepticism in the decision making process: it takes a smaller fraction of negative evidence to sink a proposal.

The decision is set by the current strength estimate. When the strength is at or above a constant threshold *accept*, the decision is **accept**; when it goes below *reject*, the decision is **reject**; in between the decision is **wait**. The closer *accept* and *reject* are to zero, the more haste: decisions will be faster, but also are more likely to need revision.

Because a decision may need to change in response to changing conditions, there are constants (*rail+* and *rail−*) further out beyond *accept* and *reject* that limit how far the strength can move past the decision thresholds. The larger the rails, the greater the potential commitment: it takes more contradictory evidence to change a long-standing decision.

5.3.3 Configuration Parameters

Once we have a proposed mechanism in hand, we identify the set of parameters in its configuration: these are often numerical variables, but may include other choices like policies and network structures. The defining characteristic of a configuration parameter is that there is more than one plausible value, and no compelling *a priori* reason to assume that one value must dominate. When possible, the number of parameters should be limited, because we will need to explore them thoroughly when

[6]The codetector's behavior is invariant to shifting and scaling of parameters, so the choice of zero and one is just a convenient normalization.

generating a dossier.

Example: Codetector Configuration Parameters

The codetector has five configuration parameters:

- $miss$ is a negative number.
- *accept* is a positive number.
- $reject$ is zero or a negative number.
- $rail+$ is greater than *accept*.
- $rail-$ is less than $reject$.

5.3.4 Dossier

Now that the mechanism has been described—hopefully with some guesses about how its configuration affects its behavior—we can start asking how it behaves under different conditions and configurations. If the device gives desirable behavior across a wide enough range of conditions to satisfy us, we can go on and finish creating the data sheet; if not, we bring the insight gained in generating the dossier back to mechanism design and attempt to correct the defects.

We will examine a mechanism by generating a *dossier* for it.[7] Our dossiers will contain experimental surveys of system behavior, where each survey measures the behavior of the system across a representative set of conditions and configurations, plus any other available supporting information. Dossiers will often represent a brute-force approach to understanding system behavior—a typical survey may contain thousands to millions of data points.

The goal of a dossier is not perfect understanding or optimal configuration. The goal is to make it easy to reliably produce mostly desirable behavior. Limiting the choice of parameters is one of the most important functions of the dossier, since a mechanism may begin with a vast and incomprehensible set of parameters. The key is to balance the flexibility of the configuration against the difficulty of comprehending the relationship between configuration, conditions, and behavior.

As such, generating a dossier is often an iterative process, starting with a rough survey, interpreting the results to identify major phases of behavior, then refining to gather more information as the relationships between conditions and configurations

[7] According to the Random House dictionary, a dossier is "a collection or file of documents on the same subject, especially a complete file containing detailed information about a person or topic."

begin to resolve. As surveys are added to the dossier, the most important parameters can be identified and others eliminated. It may be reasonable to eliminate a possible setting for a parameter even when tuning it provides the best behavior under some conditions: the benefit from tuning may be outweighed by the benefit derived from better understanding of a simpler device.

Why Not Theoretical Analysis?

Why do we need to resort to brute force, rather than theoretical analysis? Having had some mathematics in my training, my first instinct is to sit down with a pencil and paper and start analyzing the mechanism, searching for some nice mathematical models. Unfortunately, I have had little success in theoretical analysis of successful devices, such as the codetector, across a broad range of conditions and configurations.

For example, I first tried to analyze the codetector in terms of probability, as its action away from the rails can be viewed as incremental calculation of the log-likelihood ratio of two hypotheses, with *miss* defining the relationship between the prior probabilities of the **accept** and **reject** hypotheses. But this model breaks down at the rails and did not shed light on the trade-off between non-wavering and changeable decisions.

I then considered the codetector in terms of a random walk. This gives some insight on how much wavering to expect immediately after the decision changes. There is also exists such a thing as a "reflecting random walk," which can model behavior involving the rails. Random walks, however, do not provide answers about what happens when the independence assumption is violated.

While someone wielding mathematical tools I do not know might well be able to crack these problems and produce a clean analysis of the codetector or other devices, I would not generally count on this as a routine path to insight. Some reasons why:

- Devices are generally non-linear in their actions, often with sharp transitions, making behavior harder to compute. For example, a codetector's strength is shifted linearly by the evidence it receives, except when it is near a rail.

- Broad ranges tend to include regions where simplifying approximations break down. For example, a codetector's rail non-linearity usually has minimal short-term interaction with its decision, except when $miss$ is on the same order as to the difference between $rail-$ and $reject$.

- The transition between disable behavior and misbehavior is often gradual and unclear. For example, the definitional tension between non-wavering decisions and ability to change means that there is no easy-to-define point where appropriate change transitions into wavering.

All of this suggests that a good theoretical analysis of a device would often be a significant research project by itself. For a device as simple as the codetector, these problems could probably be solved with a little work. More complicated devices, or devices made by composing other devices, are likely to be much worse.

What I want, however, is to make the engineering of devices routine, since there are many devices to develop and debugging a device during development will require many analyses. For that, I must abandon theory and turn to experimentation on a massive scale.

Description of a Survey

A description of a survey has four parts:

- **Conditions:** the combinations of conditions to be surveyed.
- **Configurations:** the combinations of configuration parameters to be surveyed.
- **Experiment:** the test to be conducted for each combination of conditions and configurations, including what data is to be gathered.
- **Results:** interpretation of the results of the survey, with particular emphasis on identifying major phases of behavior.

Visualization is an important part of interpreting the dossier, given the sheer volume of data. One tool I will often employ is a tableau of charts (such as in Figure 5-8 in the example below), which compacts four to six variables into two dimensions, varying two on the vertical and horizontal axes of each chart and the rest across the vertical and horizontal axes of the tableau of charts and groupings within the tableau. Such a tableau is then read by looking at the details of a single chart or at visual trends across the aggregate.

Example: Codetector Dossier

We start the dossier-building process for a codetector by examining some limiting cases to help us guess what behaviors will be seen. With this information, we do a rough initial survey, then follow up with a detailed survey using simplified representative conditions and configurations.

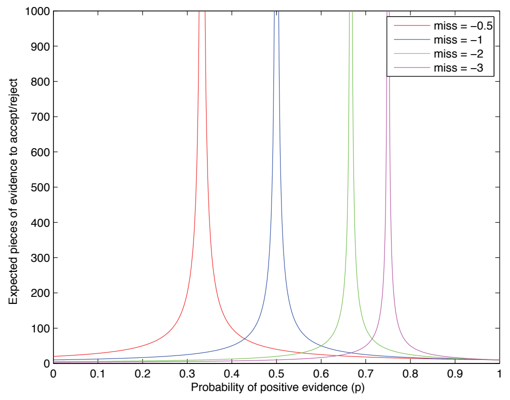


Figure 5-3: A codetector fed a random stream of evidence converges to a decision quickly except near the decision threshold. This graph shows, for several values of $miss$, the relationship between probability of **positive** evidence and expected time to decision for a codetector configured with $accept = 10$, $reject = -10$, $rail+ = \infty$ and $rail- = -\infty$.

Limiting Cases First off, if the evidence is all positive, then the codetector will quickly choose **accept** and never change. If a small random fraction of the evidence is negative, then it will not change this as long as $rail+$ is well above $accept$. Likewise, if the evidence is all negative, then the codetector will quickly choose **reject** and never change, and this will not be affected by a small fraction of positive evidence.

If subsequent pieces of evidence are independent and the behavior of the evidence stream is approximately constant over the whole stream, then we may model it as being generated by random coin-flips with probability p of a piece of evidence being **positive**. We can then consider the behavior of the codetector as a random walk. Ignoring the rails, this means that after n pieces of evidence, the expected strength is $n(p + (1-p)miss)$.

By plotting the probability of **positive** against expected time to equidistant decision thresholds for various values of $miss$ (Figure 5-3), we see three expected phases of behavior: *fast accept* at high probability, *fast reject* at low probability, and *dither*—a long period of waiting—around the transition point where $p + (1-p)miss = 0$ (when we need to refer to this point later, we will call it the accept/reject boundary). The fast accept and fast reject phases are both desirable behavior; dither is misbehavior because the decision is not fast.

What about when the character of evidence is different in different parts of the stream? The most extreme case would be evidence that switches back and forth between long streaks of **positive** and long streaks of **negative**. In this case, the codetector would change its decision repeatedly, rocketing back and forth from rail to rail. The speed of a round trip from rail to rail is regulated by the size of $miss$ and the distance between rails, $(rail+ -(rail-)) \cdot (1+1/m)$. This is a fourth phase, *oscillate*, which is expected when the evidence is not independent and the balance between positive and negative evidence fluctuates greatly over long sequences in the stream.

The oscillate phase is caught in the definitional conflict of desirable behavior. If the length of the fluctuations is too short, then oscillate is misbehavior, because the decision wavers, but if the length of the fluctuations is long enough, then oscillate is desirable because the evidence is changing. This does tell us, however, that a codetector can always change its decision in the face of changing evidence.

This covers the limit cases well when the thresholds and rails are far apart and $miss$ is a small fraction of $reject$, and gives us a good guess as to what phases of behavior we should expect to see in our surveys. In order to understand the boundaries between limit cases, and how much we can push the system before it begins to misbehave, we need to start conducting experimental surveys.

Experimental Conditions To survey behavior, we will need to generate a synthetic stream of evidence. The stream of evidence will be regulated by three parameters, based on our analysis of limit cases:

- *presence* is the base probability of **positive** evidence.
- $nonuniformity$ is the amplitude of dependence effects.
- *volatility* is an integer indicating the largest granularity of dependence.

Dependence, here, means correlations between elements close together in the evidence stream, and is applied fractally. For *volatility* $= k$, there are k layers of dependence, operating on exponential segments from 1 to 2^{k-1} in length. Each layer independently flips a fair coin to determine whether each segment will have a positive or negative dependence effect: if it is positive, then the probability of **positive** is raised by $nonuniformity/2$ for that time segment; if it is negative, then it is lowered by $nonuniformity/2$. Thus, the probability of **positive** is distributed randomly (with a binomial distribution) in the range $presence \pm volatility \cdot nonuniformity/2$ (clipped at 0 and 1). See Figure 5-4 for probability levels generated with varying levels of volatility.

In terms of interpreting behavior, we will consider *presence* to be the signal that should be driving the decision, and $nonuniformity$ and *volatility* are measures of the strength of masking effects from dependence in the evidence stream. Thus, when considering desirability, a change in evidence would be modelled as a change in *presence*,

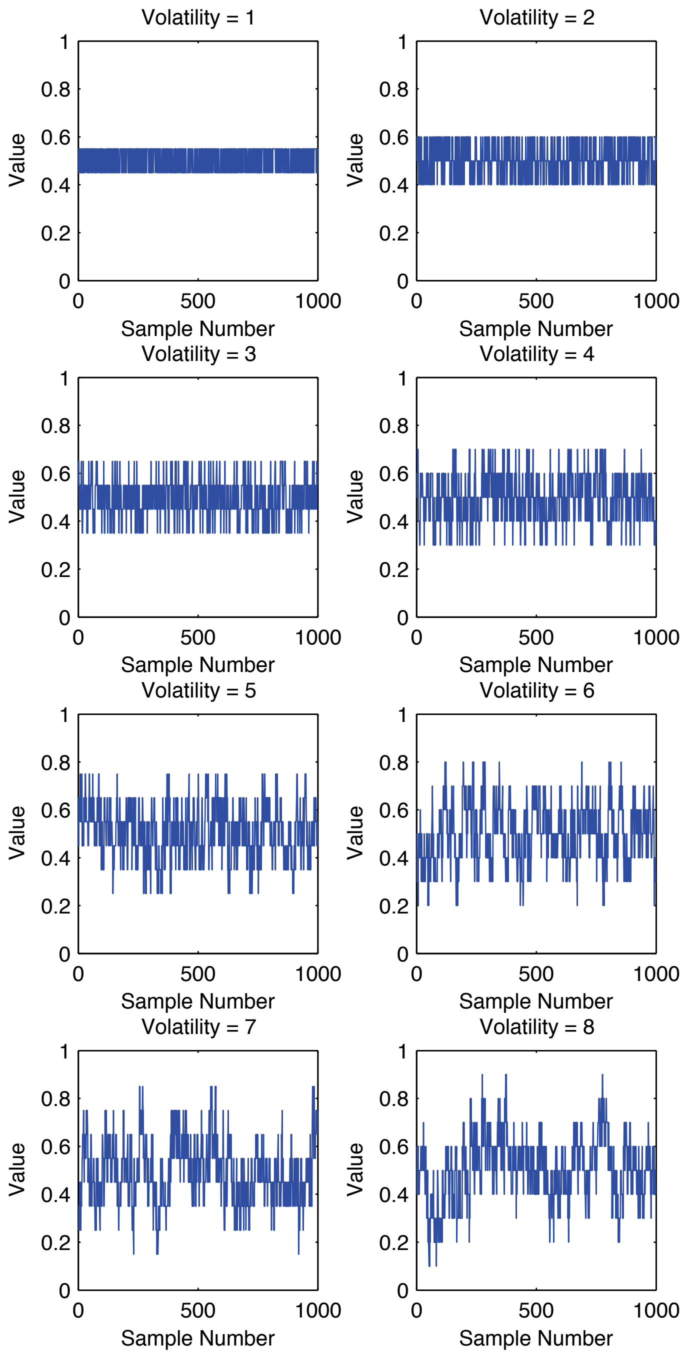


Figure 5-4: The *volatility* parameter controls the time-scale of correlation in the input stream. The higher the *volatility*, the longer the period of variation. These graphs show the probability of **positive** evidence over 1000 pieces of evidence, with $presence = 0.5$, $nonuniformity = 0.1$ and *volatility* ranging from 1 to 8.

even though such a change may be indistinguishable over the short term from the effects of $nonuniformity$ and *volatility*. Oscillation from dependence will be considered "bad."

Rough Survey We begin with a rough survey, in which all the condition variables and all five configuration parameters (*miss*, *accept*, *reject*, *rail*+, and *rail*−) vary independently. Letting everything vary independently is a safe start, though expensive. Giving all of the variables except *presence* exponential ranges from tiny to rather large makes it unlikely that any major behaviors will be missed. The goal is to use this information to enable a more precise survey with fewer variables.

- **Conditions:** *presence* ranges from 0 to 1 in steps of 0.1, *volatility* ranges from 0 to 7 in steps of 1, and $nonuniformity$ takes the values 0, 0.01, 0.05, 0.1 and 0.2, for a total of 440 combinations.
- **Configurations:** All variables range in multiples of two: *miss* ranges from -1/2 to -8, *accept* ranges from 2 to 32, *reject* takes the value 0 and also ranges from -2 to -32, *rail*+ ranges from 4 to 128, and *rail*− ranges from -2 to -128. In total, there are 6300 combinations, although combinations in which a rail is not farther from zero than its accompanying threshold are discarded.
- **Experiment:** For each of the 2.77 million combinations of conditions and configuration, I run a single trial in which a codetector is given 10,000 pieces of evidence. I collect data after each piece of evidence, recording the amount of time when the decision is **accept**, the amount of time when the decision is **reject**, and the number of times the codetector changes "zone", where the "zone" is detected by the strength assuming a value of 0, *rail*− or *rail*+. Fast accept and fast reject are characterized by low zone changes and one high decision time, oscillate is characterized by high zone changes, and dither is characterized by low zone changes and low decision times.
- **Results:** A quick skim of the results shows two opportunities for simplification: symmetric parameters and fixed volatility. Further analysis demonstrates that these apparent opportunities actually exist.

  I analyzed the rough survey to find support for doing a detailed survey with *volatility* = 7 and only symmetric parameters. To this end, I created bar graphs showing the distribution of comparisons between *volatility* = 7, symmetric parameters, and the rest of the trials. Complicating the analysis is the fact that the system behaves very differently in different phases: when a data point is produced by misbehavior, the distribution of data produced by the experiment should be much more random than that produced from a combination of conditions and configuration that produces desirable behavior.

Remember, the overall question we want to answer is, “What is the relationship between conditions, configuration, and behavior?” To get this from a restricted survey, we need a map from general conditions and configuration to the restricted survey, and confidence that the general behavior is not significantly less desirable than the behavior of the point it maps to in the restricted survey.

I test this with an analysis of results from the rough survey, comparing the results of each trial to the corresponding trial for simplified conditions and configuration. I make two comparisons for each point: the first is number of zone changes: when comparing two data points, the one with less zone changes is likely to be more desirable, since that means less oscillation. I thus use the distribution of general minus simplified.

The second comparison tests the speed and solidity of decision, as measured by the absolute difference between accept and reject decision times. The higher the difference, the less dither or oscillation in the decision, so when comparing two data points, the one with the higher difference is likely to be more desirable. I thus use the distribution of simplified minus general.

When *presence* is near the accept/reject boundary, our previous analysis tells us that we should see misbehavior. If the simplification is good, we expect to see a wide distribution with a mean of less than zero. When *presence* is far from the boundary, on the other hand, the distribution may be wide or narrow, but the distribution should have almost nothing significantly above zero.

To let us see these cases and the transition between them, each distribution graph is a bar graph showing nine distributions at various proximities to the boundary: the leftmost is $|presence - boundary| < 0.1$, the next is $0.1 \leq |presence - boundary| < 0.2$, and so on in steps of 0.1 up to $0.8 \leq |presence - boundary|$.

Now we can finally look at the results, starting with parameter symmetry. Figure 5-5(b) shows that the difference of decision times in an asymmetric trial is predicted by the symmetric trial in the preferred direction of motion: if *presence* is above the accept/reject boundary, we select the symmetric trial with the same *accept* and *rail*+; otherwise, we select the symmetric trial with the same *reject* and *rail*−. Likewise, Figure 5-5(a) shows that the zone changes of an asymmetric trial are predicted by the symmetric trial corresponding to side with the nearest rail.

Moving on to *volatility*, Figure 5-6 shows that both difference of decision times and zone changes are predicted by *volatility* = 7. The prediction is not as clean as for symmetric versus asymmetric, but it does clearly hold.

Finally, Figure 5-7 shows the result of combining both simplifications into a single map, showing that it is, in fact, reasonable to do a detailed survey with only symmetric parameters and *volatility* = 7, since it gives good conservative predictions for the general collection of conditions and configurations.

Detailed Survey Based on the results of the rough survey, we can now make a more detailed survey of a smaller characteristic region of space. Given what we learned from the rough survey, we can make conservative estimates of behavior if we know the behavior with high volatility and symmetric thresholds and rails. This survey explores the remaining range thoroughly, with an eye towards visualizing the results.

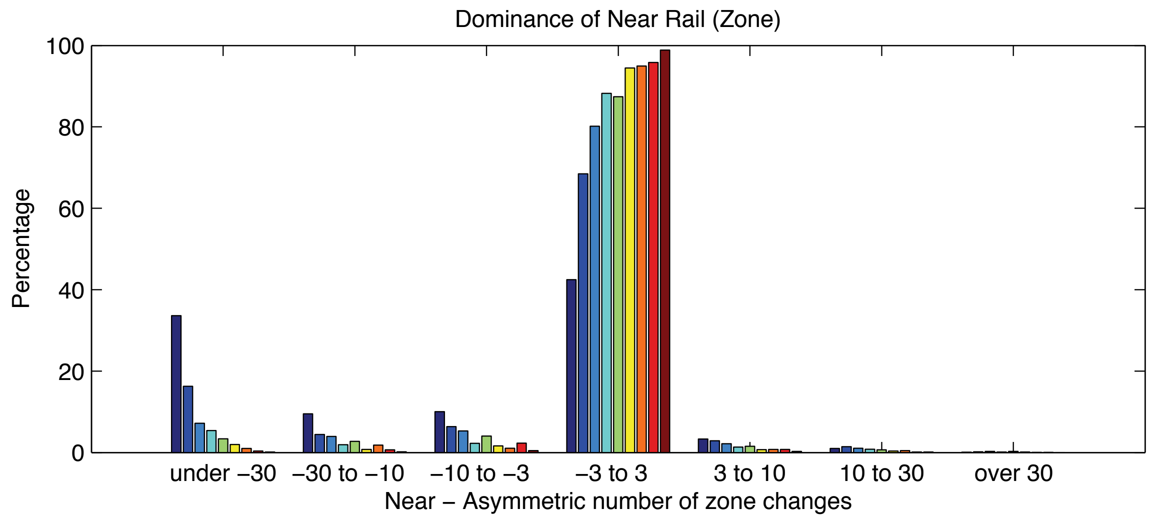
- **Conditions:** *volatility* is fixed at 7 (meaning the granularity of the longest dependencies is 64 pieces of evidence), *presence* ranges from 0 to 1 in steps of 0.01, $nonuniformity$ ranges from 0 to 0.2 in steps of 0.01, for a total of 2121 combinations.

- **Configurations:** *miss* ranges from $-\frac{1}{2}$ to -8 in powers of two, *accept* ranges from 2 to 32 in powers of two, and *reject* = −*accept*. Finally, the rails are a multiple of *accept* and *reject*, with the multiple ranging from $\frac{3}{2}$ to 4 in steps of $\frac{1}{2}$. In total, there are 150 combinations (5 *miss* values, 5 thresholds, 6 rail multipliers).

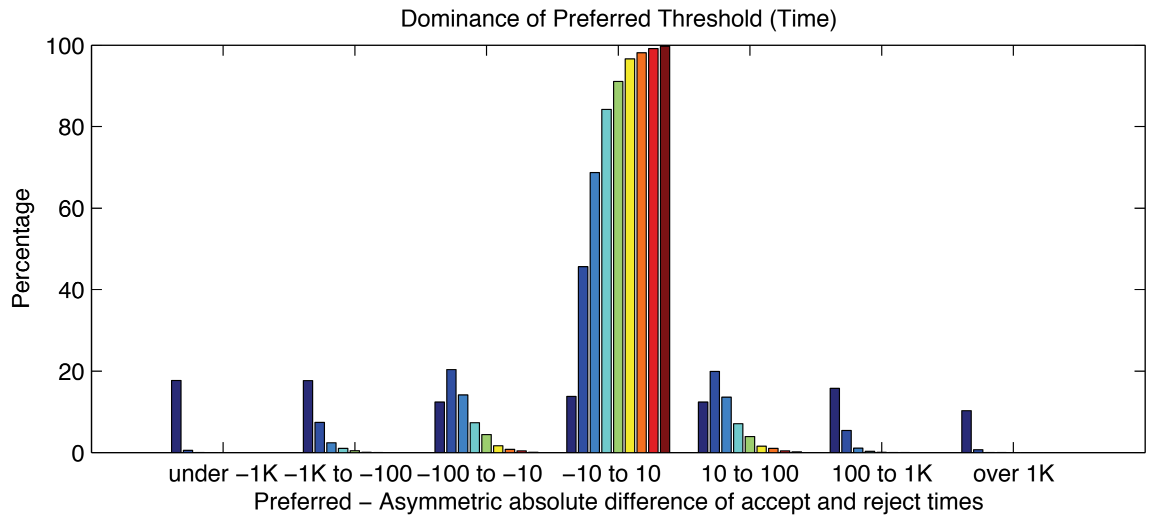
- **Experiment:** For each of the more than three hundred thousand combinations of conditions and configuration, I run ten trials in which a codetector is given 1000 pieces of evidence. Data is collected as in the rough survey.

- **Results:** The results are summarized by the tableau shown in Figure 5-8. Each of the five square blocks shows the results for one *miss* value; within a block, the threshold rises from top to bottom and rail multiplier rises from left to right. Finally, in each chart, *presence* rises from top to bottom and $nonuniformity$ rises from left to right.

  The color of each pixel indicates the behavioral phase of the conditions and configuration for that location in the tableau, using the three components of its color:

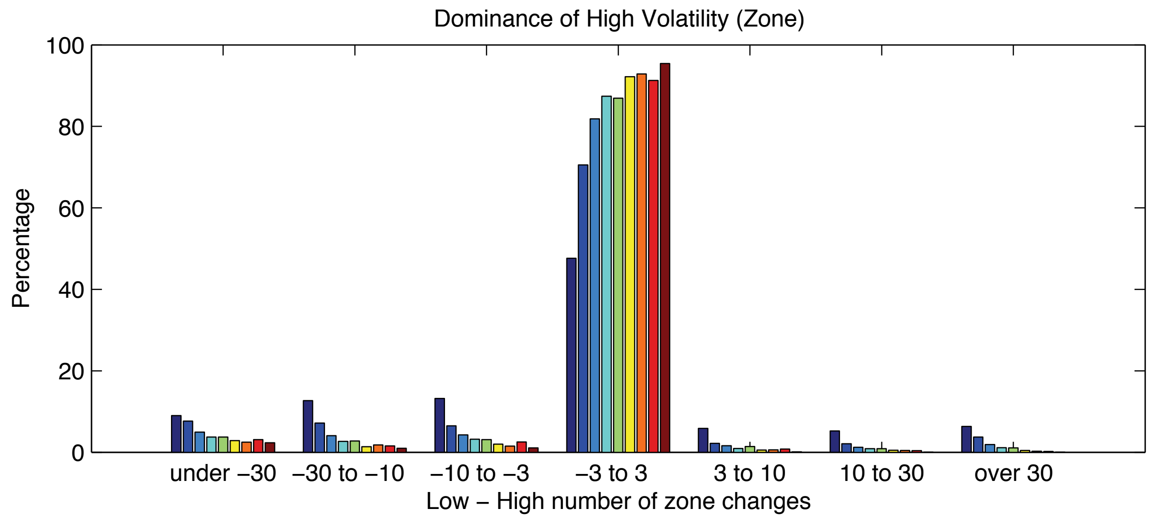


(a) Zone Dominance

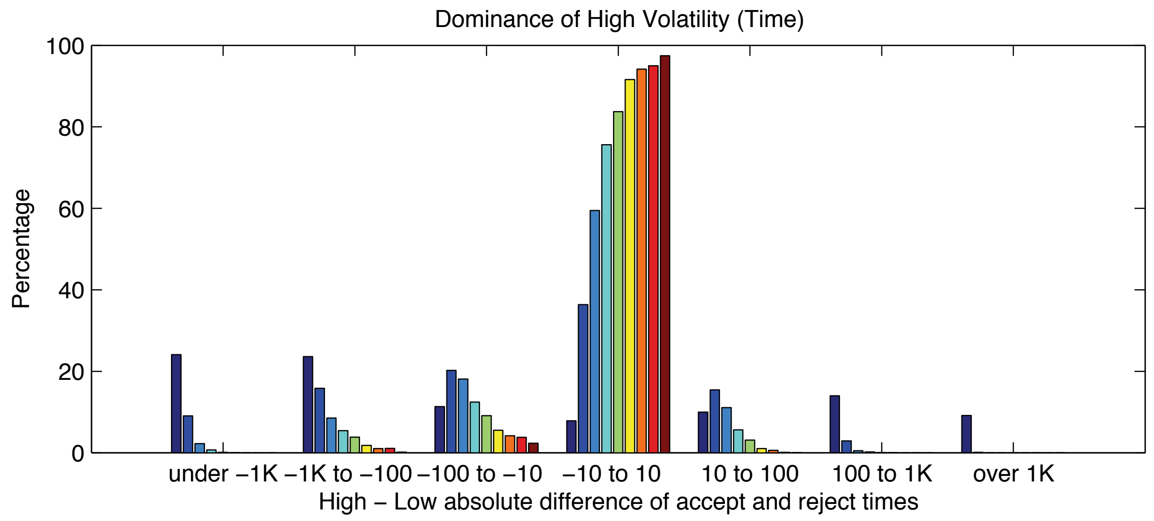


(b) Time Dominance

Figure 5-5: Behavior of a codetector with symmetric parameters gives a conservative approximation of the behavior of a codetector with asymmetric parameters. Asymmetric zone changes are lower than symmetric for the closer rail (a) except near the accept/reject boundary (cool colors), and asymmetric difference of decision times is predicted by symmetric for the preferred threshold (b) except near the accept/reject boundary (cool colors).

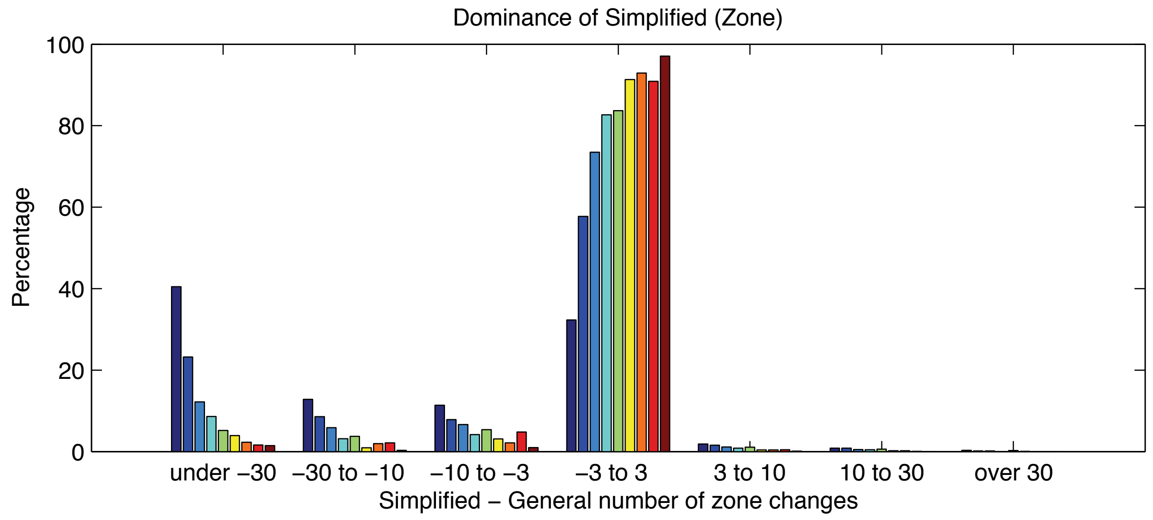


(a) Zone Dominance

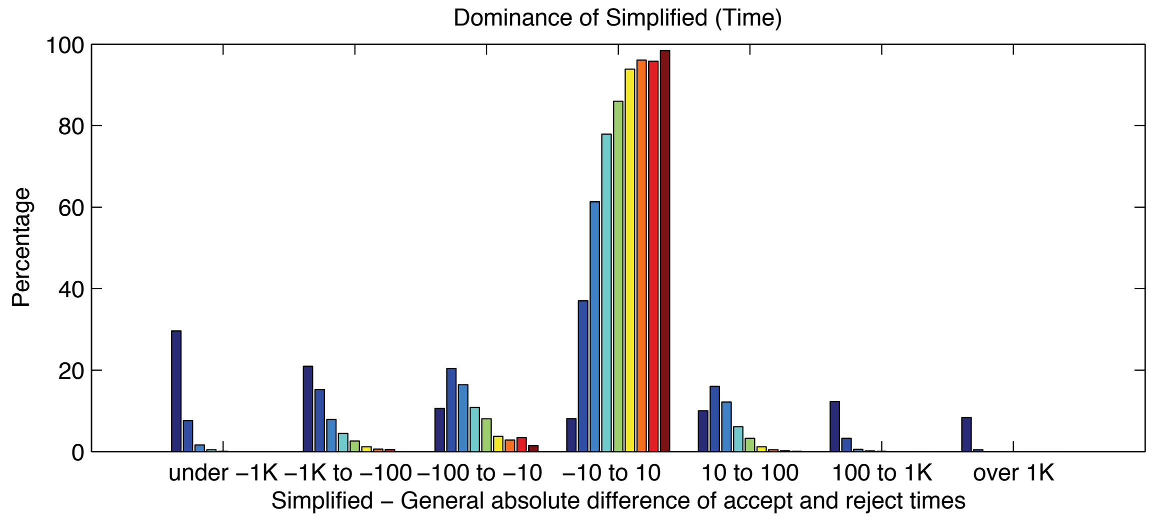


(b) Time Dominance

Figure 5-6: Behavior of a codetector in conditions of high *volatility* gives a conservative approximation of low *volatility* behavior. Low *volatility* gives lower zone changes (a) except near the accept/reject boundary (cool colors) and lower difference of decision times (b) except near the accept/reject boundary (cool colors).



(a) Zone Dominance



(b) Time Dominance

Figure 5-7: Behavior of a general codetector with can be predicted conservatively from the behavior of a codetector with symmetric parameters in conditions of high volatility. Both zone changes (a) and difference of decision times (b) are almost never worse in the simplified case than the general case except close to the accept/reject boundary (cool colors).

  - Blue indicates fast acceptance. Intensity is calculated from the amount of time when the decision is **accept**, calibrated so that *accept* time outside acceptance is full intensity and ($accept * 101$) time outside acceptance is zero intensity.
  - Red indicates fast rejection. Intensity is calculated from the amount of time when the decision is **reject**, calibrated so that $accept/-miss$ time outside rejection is full intensity and ($accept * 100 + accept/-miss$) time outside rejection is zero intensity.
  - Green indicates oscillation misbehavior. Intensity is calculated from the number of zone changes, subtracting one to ignore the first decision. The intensity is calibrated based on the minimum length for a round trip from rail to rail, with full intensity at 0.2 transitions per round trip time and zero intensity at zero transitions per round trip time.

Thus, solid primary colors and black indicate clear regions of the four predicted behaviors. Black indicates dither, since it means that the codetector is neither settling quickly nor oscillating. Patchy regions, secondary colors, and dim colors all indicate boundaries that may harbor other forms of behavior.

This survey shows that the relationship between conditions and configuration is mostly well differentiated into the four predicted behavioral phases. There are some boundary regions, but they are relatively small and are most unclear when they involve a transition between the two undesirable behaviors.

The survey also confirms our intuitions about the effects of a codetector's parameters on its behavior. The larger the *miss* (skepticism), the fewer things are accepted, and the easier it is to oscillate. Raising the thresholds does not change the region of dithering, but it lowers the amount of oscillation. Pushing the rails outwards has the same effect. Finally, the survey shows that parameter space is fairly smooth, so small changes in parameters will have little effect on the behavior of the codetector.

Overall, this is a good result. Because behavior changes gradually as the configuration varies, and because there are many settings that give good behavior across a majority of conditions, it is easy to choose an acceptable configuration, even when there are other constraints (for example, skepticism will often be constrained how the codetector is being used).

5.3.5 Usage Specification

Having built a dossier and gained insight on the behavior of a device, we can complete the data sheet with instructions on how to connect this device to other devices.

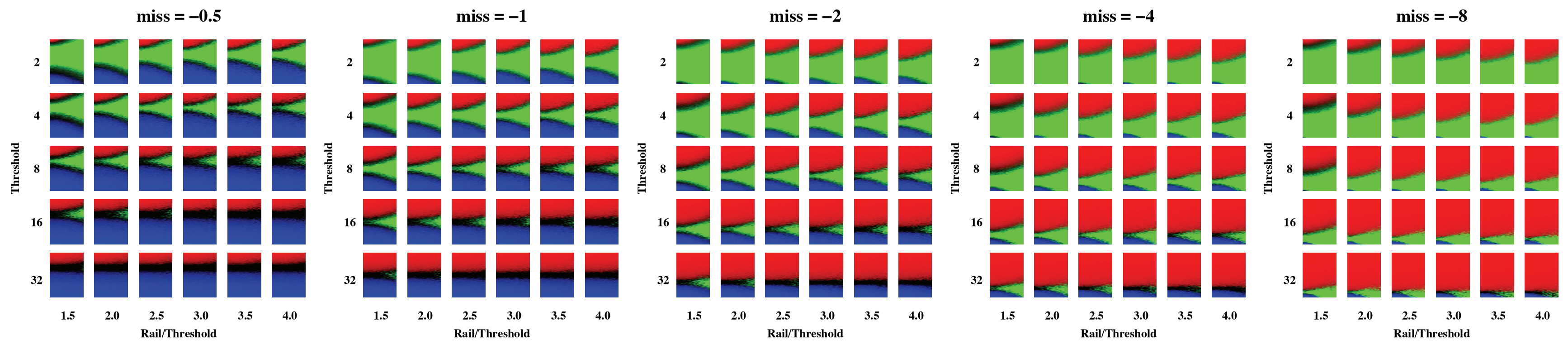


Figure 5-8: Tableau of codetector behavior survey. Within each chart, *nonuniformity* ranges from 0.0 to 0.2 left to right, and *presence* ranges from 0.0 to 1.0 top to bottom. Threshold is *accept* and $-reject$, and Rail is $rail+$ and $-(rail-)$. Red indicates fast rejection, blue is fast acceptance, and green is undesirable oscillation. Lack of all three (black) indicates dither. Mixed colors are transitional regions with unknown forms of behavior.

The usage specification consists of four parts:

- **Configuration Policy:** Given a range of conditions, what configurations are expected to produce desirable behavior over a large portion of the range?
- **Limiting Conditions:** Given a configuration, what conditions will produce mostly misbehavior and should therefore be avoided?
- **Failure Simplification:** Assuming that misbehavior will occur, how can its impact on other devices be limited?
- **Cost:** Asymptotic cost of a device measured using the six measures presented in Section 4.3: time and space costs for development and mature operation and the type of developmental variation and error in execution that may occur.

The first two are essentially just a summary of the insight gained from the dossier, and cost usually can be drawn from the mechanism in a fairly straightforward manner. Failure simplification, on the other hand, will require some additional work.

The first important thing to remember, when thinking about failure simplification, is that the distinction between desirable behavior and misbehavior may be fuzzy (as seen in the codetector's dossier) or even purely contextual (as in the tension between non-wavering and ability to change requirements for the codetector).

Furthermore, the dossier-building process makes no promises of complete knowledge about behavior, only that all of the major phases of behavior will be identified. The lessons of chaos and complexity science tell us that we should be suspicious of the boundaries between phases. Within the vast basin of a phase, there is not likely to be trouble; in the transition between phases, all manner of strange and delicate behaviors may lurk. If we were students of complexity, we would dive into the borders to see what they contain; as engineers, we must banish them as foes of predictable behavior.

Failure simplification thus starts with the recognition that not all misbehavior is created equal: some sorts are harder to detect than others, and some exhibit more complex behavior than others. By modulating behavior to produce simpler and harder to detect misbehavior, we can effectively sweep the complexity of the system under the rug, generally at the cost of getting simple misbehavior in some circumstances when we would have otherwise ended up with desirable behavior.

I originally learned about failure simplification from a study of the distributed computing aspects of live-action roleplaying games[5]. Ranged combat in games run by the MIT Assassins' Guild uses a distributed consensus algorithm whose requirements—correctness, partition tolerance, and wait-free resolution—are formally impossible to satisfy simultaneously[14]. Even a transient failure of wait-free resolution or partition tolerance is severe, since it means that the game must be halted, wreaking havoc on the players ability to suspend disbelief. Loss of correctness, on the other hand, makes the game unfair and prevents players from having fun.

The Guild combat algorithm resolves this dilemma by noticing that lack of correctness is harder to detect, and managing failures so that they primarily take the form of transient or unnoticeable violations of correctness. A broad range of potential failures is thereby simplified into a small number of low-impact failures, allowing the overall behavior to satisfactorily approximate that of the ideal, impossible algorithm.

Given a dossier, my preferred approach to failure simplification will be to create a test for desirable behavior far from the phase transitions, then select a simple default behavior for when the test is not satisfied. Since the test is for behavior far from the transitions, some desirable behavior will get swept up, and turned into misbehavior by the default. There will still generally be complex transitions with hard to define boundaries: the test/default approach, however, transforms these from areas of uncertain behavior to areas where it is uncertain which of two well-understood behaviors will be exhibited.

Notice that failure simplification does not promise anything about preventing misbehavior from spreading throughout the system—for the want of a nail, the kingdom may still be lost! What failure simplification generally does do, however, is extend the time it takes for misbehavior of one device to detectably impact the behavior of others devices that interact with it. Across multiple linkages of this type, the time scales can become very long indeed, and once they are comparable to the lifespan of the system, then the question becomes moot.[8]

Example: Codetector Usage Specification

- **Configuration Policy:** Assume that *miss* is set by outside requirements for skepticism, or chosen arbitrarily if that is not the case. Assuming we want fast response to change, we should choose the lowest values of *rail+* and *accept* where there is little oscillation misbehavior. In the tableau (Figure 5-8) this is the upper-left chart with (almost) no green.

  For example, with $miss = -2$, filling out the configuration with $accept = 16$, $reject = -16$, $rail{+} = 64$, and $rail{-} = -64$ is expected to produce good results.

- **Limiting Conditions:** The transition between the fast accept and fast reject

[8]Intriguingly, this might be a way to bypass Gödel's inconsistency theorems.

phases always contains a narrow band of dither. Inspecting the tableau, we can see that the location and breadth of this region is unaffected by everything but *miss*. As such, if anything is known about likely *presence* values, *miss* should be chosen to ensure that the bulk of the *presence* values are likely to fall outside this band.

- **Failure Simplification:** Assuming that the codetector is being used to judge the validity of a proposal, it is generally safe to discard the proposal on rejection, with the assumption that if it is a good proposal it will be submitted again and have a good chance of getting accepted the next time around.

  As such, we can manage the misbehavior of a codetector by testing for fast acceptance and treating it as rejection if the test fails. If all evidence is **positive**, then it takes *accept* rounds to decide on **accept**. Thus, if any period longer than 3 to 5 times *accept* goes by without the decision being **accept**, then the proposal can be discarded.

  Finally, if nothing is known about likely (or desired) *presence* values, then *miss* can be chosen randomly from a range of values.

- **Cost:** The codetector is a simple program, incorporating neither communication paths or other devices. As such, all four of its costs are constant ($O(1)$) with regards to its conditions and configuration. Variation has a small chance of producing a codetector that is simply dead on arrival, and error during execution may result in occasional perturbations of the strength estimate.

	Development	Mature
Time	$O(1)$	$O(1)$
Space	$O(1)$	$O(1)$
Imperfection	DOA	perturbation of strength

To gain a clearer picture of the effect of applying failure simplification, we can run a survey across the same range of conditions and configurations as the detailed survey in the dossier. For each combination, I run ten trials with a single codetector under failure simplification: if the codetector decides **reject** or goes for $4 \cdot accept$ steps without an **accept**, it is discarded and replaced. A trial runs for 10 attempts or 1000 steps, whichever comes first, and is counted as eventual acceptance if the final decision is **accept**.

Figure 5-9 shows a tableau of the results, organized as before, but with different colors: blue indicates the fraction of trials with eventual acceptance, and red indicates the fraction with consistent rejection. Looking across the tableau, we can see that the behavior is roughly consistent with transforming phases of misbehavior and transitional regions into rejection. Where threshold and rails are small, the region of acceptance is generally smaller than before; where they are large, it is often larger, as multiple attempts give more chances to succeed.

5.4 Using Data Sheets for System-Building

With data sheets in hand, building a system from devices is fairly straightforward and routine—exactly what we want.

We start with conventional software engineering, wiring devices together with a simple superstructure of mechanisms whose behavior can be completely analyzed without resorting to dossier-building, like sets or communication paths. The connections between devices should obey the specifications for failure simplification.

Once the devices have been connected together, they can be configured. The trick is that some of the conditions for each device are set by the behaviors of the other devices it interacts with. Starting with a scratch configuration for each device, then, a good configuration can be found by adjusting devices one-by-one to give them a good range for their current conditions, iterating over the whole set until none needs a large adjustment. Because a good device has a very loose dependence between conditions, configuration, and behavior, even such a brain-dead process will generally produce a good enough configuration quickly. If it does not, then it is a clear indication that something deeper is wrong with the system design.

Notice that the configuration is expected to be good enough, and not optimal—finding an optimal configuration may still be very hard indeed. Even knowing that something is optimal, however, generally requires much better knowledge of the ultimate conditions of use than we usually have when working on an intelligence. As such, unless there is reason to believe that optimality is at least an order of magnitude better than "good enough," I consider it a false goal for systems of this sort.

Finally, there is no reason to avoid using the same design process as we build the system, ending up with a full data sheet including a dossier on its behavior and instructions for how to connect it to other systems.

5.4.1 Example: Agreeing on Signals

For an example of building a system by composing devices, let us consider a simplified version of communication bootstrapping. This is not the mechanism we will use, but a related problem that provides a simple-to-explain example. The challenge here is

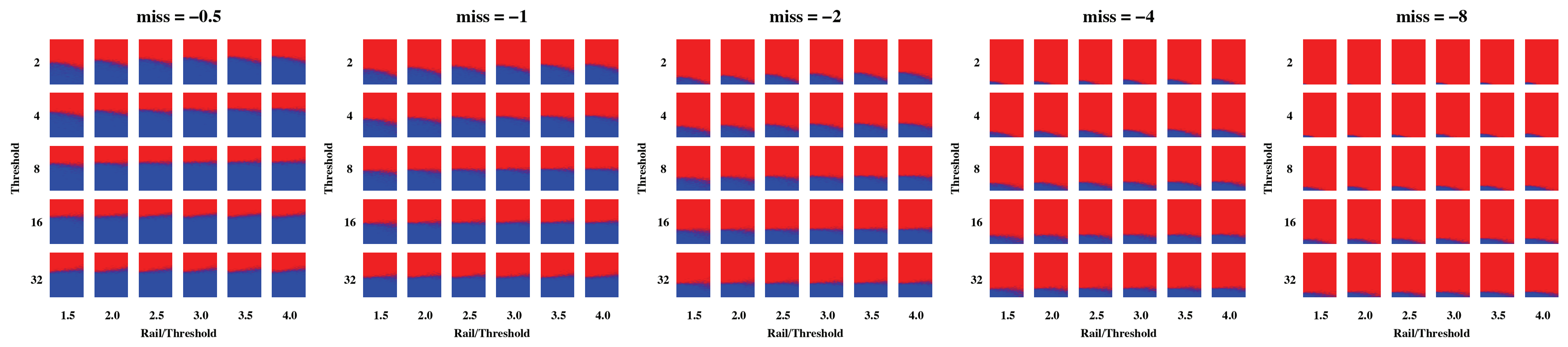


Figure 5-9: Tableau showing a survey of codetector behavior under failure simplification, discarding upon a decision of **reject** or $4 \cdot accept$ without an **accept**. Within each chart, $nonuniformity$ ranges from 0.0 to 0.2 left to right, and $presence$ ranges from 0.0 to 1.0 top to bottom. Each pixel shows the fraction of ultimate acceptance/rejection over 10 trials, where a trial runs for 10 attempts or 1000 pieces of evidence (whichever comes first). Red indicates consistent rejection, and blue indicates eventual acceptance.

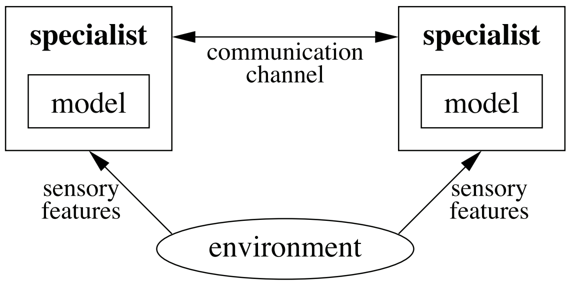


Figure 5-10: A signal agreement system consists of two specialists, connected by a two way communication channel, and an environment that sends each specialist its own set of sensory features.

for two specialists to agree on the meaning of signals—for example, consider the sort of things a vision specialist and a hearing specialist might want to communicate about traffic lights:

- A signal that means "green light" to the vision specialist and the sound of moving cars to the hearing specialist. When the vision specialist sees the green light and sends the signal, the hearing specialist expects moving cars, and can warn the motor system not to step into the road.

- A signal that means "flashing blue" to the vision specialist and the sound of sirens to the hearing specialist. When the hearing specialist hears a siren and sends the signal, the vision specialist knows to expect flashing lights and can start looking for the police car.

Using paired codetectors, we can solve this problem for strong equality associations between discrete examples. We will approach this just like we approached building the codetector, starting with the interface.

Interface Specification

- **Conditions:** The signal agreement system consists of two specialists, connected by a two way communication channel (Figure 5-10): messages on the channel are sets of signals and any number can be sent in both directions without interfering. Each example is presented to the specialists as a set of binary sensory features that arrive from their environment.

- **Range of Behavior:** Each specialist has a model of the environment that is a set of sensory features—whether currently seen or hallucinated. The observable behavior of the system is the contents of the two models after an example has been processed.

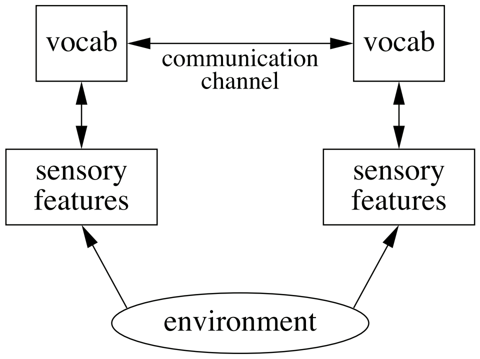


Figure 5-11: In the signal agreement system, each specialist contains a vocabulary that maps between signals and sensory features from its environment.

- **Desirable Behavior:** Signal agreement is behaving as desired when:
  - a specialist's model contains every feature that it received from the environment.
  - assuming that sensory features correspond to underlying features of the environment, the "same" features should appear in each specialist's model.

  Note that the second requirement can only be satisfied for features that are largely equivalent, like those in the examples above. This extremely simple definition of agreement does not work for features with more complicated relationships, let alone relationships between networks of features.

Mechanism

Each specialist has two components: its model (a set of features), and a vocabulary of up to *capacity* entries, mapping features to signals (Figure 5-11). One specialist is designated as the inventor and creates signals when a feature is not in its vocabulary; the other is the adopter, and uses signals created by the inventor. For now, we will simply assume such a communication channel and connection between vocabulary and sensory features can be built.

For each example, a specialist puts the features arriving from the environment into its model, then sends their corresponding signals to its partner. When its partner's signals arrive, it translates them into features and adds them to its model as well. If the two vocabularies agree on the signal that equivalent features map to, then this will produce the desired behavior.

The population of the vocabulary is controlled stochastically when it is more than a fraction *prune* full. When the number of vocabulary entries v is greater than

$prune \cdot capacity$, each new entry has a $\frac{v - prune \cdot capacity}{(1-prune)capacity}$ chance of replacing a random pre-existing entry. Notice that when $prune = 0$ this produces unconditional random allocation, and when $prune = 1$ this is equivalent to random cache replacement. Setting an intermediate value of *prune* allows a graceful transition between plentiful and scarce resource behaviors.

The remainder of the mechanism is devoted to creating and maintaining agreement on the signals for equivalent features. This is where we will use the codetector to make a pair of decisions. First, when the inventor creates a signal not in the adopter's vocabulary, the adopter guesses a mapping for the new signal (picking randomly from the sensory features currently present and not in its vocabulary) and decides whether its guess agrees with the inventor. Once the adopter decides on a mapping and starts using the signal back, the inventor must decide whether the adopter has gotten the mapping right or wrong.

Since each specialist has one decision to make for each vocabulary entry, we attach a codetector to each entry according to the interface and usage specifications for the codetector:

- Examples are **positive** if both signal and feature are present, **negative** is one is but not the other, and ignored if neither is present.
- When a codetector decides **accept**, a vocabulary entry is used to send signals. When it decides **reject**, the entry is discarded.
- The inventor treats a new entry as accepted for a grace period of the first $8 \cdot accept$ examples, allowing time for both decisions to complete. The first $4 \cdot accept$ examples are ignored, giving time for the adopter to decide; the rest of the grace period gives the inventor time to decide whether it agrees with the adopter. Without a grace period, new entries would be summarily rejected because the inventor would stop using the new signal before it had a chance to be adopted.

The last task is to configure the codetectors. We set $miss = -2$, to catch strong associations but not be prohibitively high. We have a lot of leeway to set the rest of the parameters, since the codetectors are essentially talking directly to one another; one reasonable set is $accept = 8$, $reject = -8$, $rail+ = 32$ and $rail- = -32$.

Configuration Parameters

There are two configuration parameters:

- *capacity* is a positive number.

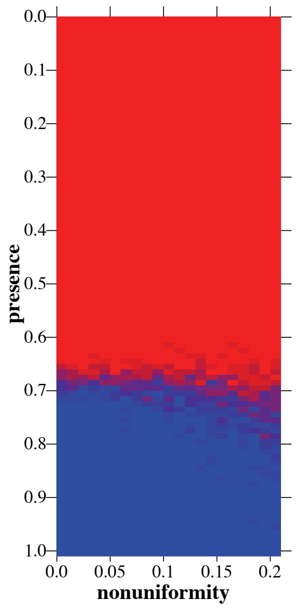


Figure 5-12: We predict the behavior of signal agreement using the codetector data sheet—in particular, the failure management survey's chart for $miss - 2$, $accept = 8$, and $rail = 32$.

- *prune* is a number between zero and one.

Dossier

We begin with predictions from limit cases. Consulting the codetector specification for failure management, we look at the chart for our chosen codetector configuration (Figure 5-12). From this, we see that, assuming vocabulary size is below $prune \cdot capacity$, we should expect that any pair of features whose probability of appearing together is about 0.8 or higher can create a sustainable agreement (though it may take several tries), and any pair whose joint probability is less than about 0.65 cannot. In between, agreements are likely to form and dissolve.

Now, to the question of whether the "right" features get paired. This is a matter of whether there are other features that appear often enough to be confused with the right feature. When a bad guess has been made, it is easier to reject, since positive examples only happen when both features are present. We can thus make an educated guess that only when the conditional probability of interference is about 0.9 or higher will incorrect pairings be sustainable. These two predictions together are illustrated in Figure 5-13.

When vocabulary size is above the $prune \cdot capacity$ threshold, some entries will be discarded. The limit case is when the entire vocabulary is full, in which case every new entry causes an existing entry to be discarded. From this, we can expect to

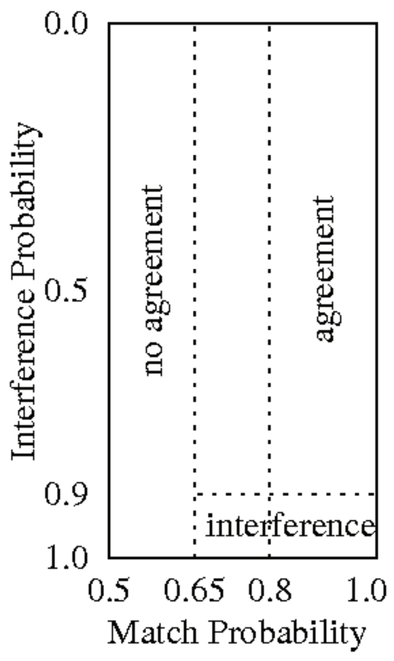


Figure 5-13: The codetector data sheet lets us predict the behavior of signal agreement with respect to the probability a feature will be matched or suffer interference.

see two different phases of behavior: when many features are seen in a short run of examples, the vocabulary will thrash, never reaching agreement. When few features are seen in a short run of examples, the vocabulary will mostly be stable, with a consistent small fraction of missing agreements.

To keep this example simple, I will only explore the behavior of the system when the vocabulary is below the $prune \cdot capacity$ threshold. Rest assured that the codetector data sheet can also be used to predict the transition between scarce resource behaviors.[9]

Initial Survey With these predictions in hand, we can begin to survey the behavior of signal agreement. We start with a survey using only two characteristics, probability of interference and probability of a match.

- **Conditions:** There are 10 features in the environment; each example contains on average a fraction k of the total features. To make an example, we choose a random number uniformly in the range $[-1, 1]$, add $10k$ and round to the nearest integer, then randomly select that many features. Each selected feature then has a chance c of going to both specialists; in the event that it does not,

[9]Example sequence properties give a thrash rate, adding convergence time tells us the spectrum of agreement durations, mapping back to signal properties gives a relation between feature properties and likelihood of agreement.

it goes to either one with equal probability. k ranges from 0.1 to 1.0 in steps of 0.01, and c ranges from 0.5 to 1.0 in steps of 0.01.

- **Configurations:** *prune* is set to 1 and *capacity* to 20.

- **Experiment:** For each combination of k and c, I run 10 trials. A trial starts with empty vocabularies and trains on 1000 examples. After each example, I collect the number of features that are training (in grace period or only one vocabulary), matched (mapped to the same signal in both vocabularies), or mismatched (mapped to different signals). The behavior of a trial is taken to be the sum of these statistics over the last 100 examples.

- **Results:** The results of this survey are summarized in Figure 5-14. The color of each pixel indicates the average behavior of the 10 trials, showing the fraction of training, matched, and mismatched as the blue, green, and red components of the pixel respectively.

  As can be seen, the results are fairly close to the prediction shown in Figure 5-13. The transition between agreement and no agreement takes place in the range $0.8 > c > 0.65$, except at very low probability, where the infrequency of examples means fewer proposals can be made—for example, at $k = 0.1$ each feature only appears in 100 examples, and a proposed signal cannot be rejected before the first 32-example grace period ends. The region of interference, where mismatches are common, is likewise almost entirely confined to the predicted region, where $k > 0.9$ and $c > 0.65$.

Extended Survey With this survey in hand, we can continue on to a more extended survey, where we ask what happens when features have different likelihoods of appearing in the environment or to a particular specialist.

There are lots of possible ways that the distribution of features can vary, and surveying them all is impractical. Instead, we will take a (hopefully) representative slice of behavior by splitting the features into two halves, a frequent half and an infrequent half, and controlling their relative likelihood of appearing with a parameter *skew*: when few features appear, the ones that appear will be from the high-frequency group with probability $0.5 + skew$ and from the low-frequency group with probability $0.5 - skew$. When many features appear, however, the high-frequency group may saturate, in which case the excess spills over into appearances from the low-probability group.

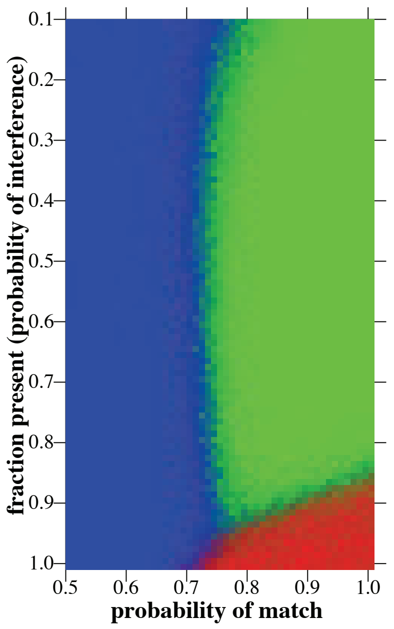


Figure 5-14: Chart showing the results of the initial survey. Probability of a match (a feature being observed by both specialists) ranges from 0.5 to 1.0 left to right, and the fraction of features present in each example ranges from 0.1 to 1.0 top to bottom. Blue indicates no agreement, red indicates mistaken agreement, and green indicates correct agreement.

That takes care of distribution of features appearing in the environment. For the likelihood of features appearing at a particular specialist, we introduce a *bias* parameter, so that when a feature appears to only one specialist, it goes to the inventor with probability *bias* and the adopter with probability $1 - bias$.

- **Conditions:** *skew* ranges from 0.0 to 0.5 in steps of 0.1 and *bias* ranges from 0.0 to 1.0 in steps of 0.1. As before, k ranges from 0.1 to 1.0 in steps of 0.01, and c ranges from 0.5 to 1.0 in steps of 0.01.

- **Configurations:** *prune* is set to 1 and *capacity* to 20.

- **Experiment:** For each combination, I run only one trial (due to the much larger number of combinations). The trial is run and data collected as for the initial survey.

- **Results:** The results of the extended survey are summarized in a tableau in Figure 5-15. We see that *bias* has little effect on behavior, while *skew*, by raising the likelihood that half of the features will be present, proportionally extends the interference range of that half of the features. We may note with relief that this region is half interference and half agreement, suggesting that even when high-frequency features are entangled and making mistakes, they do not interfere with agreement between the specialists for low-frequency features. The high *skew* also creates a region of half no agreement where k is low, but this appears to be merely an effect of there not being enough examples of the low-frequency features to complete the training of a signal.

Usage Specification

Next, we extract a usage policy from the dossier.

- **Configuration Policy:** The limited dossier I have constructed gives us little information, only that $prune \cdot capacity$ should be set well above the expected number of features.

- **Limiting Conditions:** Signal agreement fails when there are interfering signals with a very high conditional probability of being present, or when signals are not highly correlated between specialists.

- **Failure Simplification:** Interference manifests as extra features in a model. This can be mitigated by adding a third codetector, which compares predictions (made by a third party mechanism that uses the feature's presence) to future sensory input. Until this decision maker accepts the feature's presence as "correct," it may be assumed to be dubious information, possibly a result of interference, and ignored. This may be predicted to transform a large fraction of interference failures into no agreement failures, which are likely to be less disruptive.

- **Cost:** This naively designed device will be rather costly, since no attempt has been made to design interfaces suitable for development. In particular, the set of features and its interface to the vocabulary are quite expensive since they must be assumed to each be specially constructed in the absence of any other explanation. Besides this expense, there is also the question of how much the communication channel costs; we will simple add it as a separate factor to each cost.

  Finally, variation leads to dysfunctional features and variation in capacity, while execution error leads to missing features that should otherwise have been communicated.

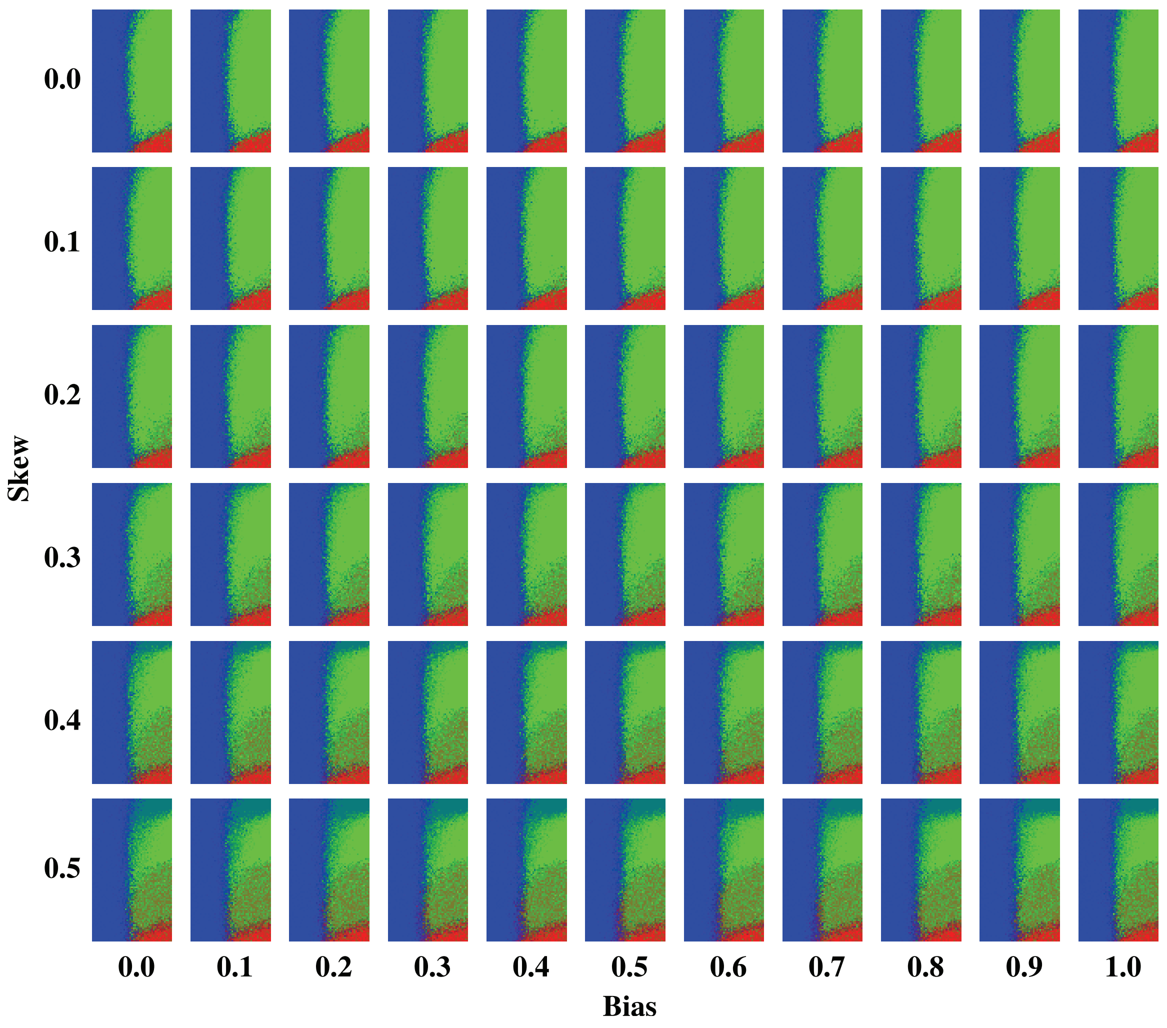


Figure 5-15: Tableau showing the results of the extended survey. Probability of a match (a feature being observed by both specialists) ranges from 0.5 to 1.0 left to right, and the fraction of features present in each example ranges from 0.1 to 1.0 top to bottom. Blue indicates no agreement, red indicates mistaken agreement, and green indicates correct agreement. Other colors are mixed behavior, with the mixture indicated by the component.

	Development	Mature
Time	$O(1 + \lvert channel \rvert)$	$O(1 + \lvert channel \rvert)$
Space	$O(\lvert features \rvert + \lvert channel \rvert)$	$O(capacity \cdot \lvert features \rvert + \lvert channel \rvert)$
Imperfection	unusable features, more/less capacity	false negative features

If we wish, we can now use signal agreement as a free-standing device, with no thought to the possibly complex behavior of the codetectors inside. We will not use it further in this document, however.

Chapter 6

A Design for Communication Bootstrapping

We now confront the question of how exactly communication bootstrapping might be used to support intelligence. Recall the definition of communication bootstrapping:

> A network of devices exhibits *communication bootstrapping* when they use shared experiences to reach agreement on a system of signals for communicating with one another.

We need a design for specialists in which communication bootstrapping occurs and produces agreement on useful signals. Further, once signals are agreed upon, their flow between specialists must help the specialists to cooperate.

In this chapter, I review the previous work on communication bootstrapping and present the general architecture we will use in following my roadmap for investigating learning by learning to communicate.

6.1 Communication Bootstrapping v1.0

The architecture used in [4] and [3] is the starting point for our investigation, because it is known to produce communication bootstrapping. It happens that the system in these papers was never given a name, so let us refer to it as *Communication Bootstrapping v1.0*.

This architecture, the original investigation into communication bootstrapping, was a proof of concept that a communication system could be self-organized on biologically inspired hardware. It was inspired by Kirby's work on language evolution[21], which is closely related to other work by Steels[38], Yanco[45], and Batali[2] among others.

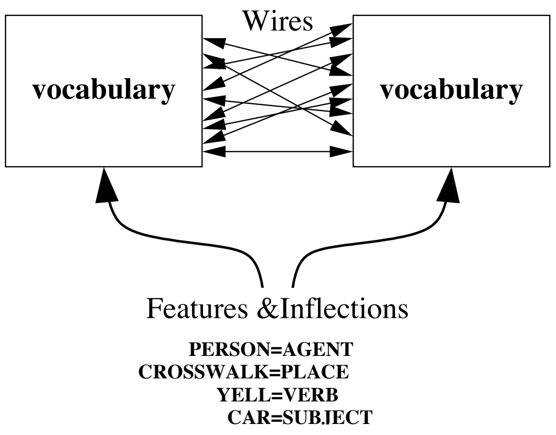


Figure 6-1: The v1.0 architecture for communication bootstrapping uses two vocabulary-building devices that learn to communicate from shared examples.

6.1.1 What v1.0 Does

Communication Bootstrapping v1.0 uses the simple architecture shown in Figure 6-1. In this design, two vocabulary-building devices are connected by a twisted bundle of wires[1] and presented with scenes digested into feature/role pairings. For example, "The person yelled at the car in the crosswalk" would be presented as a set of four pairs, `YELL=VERB`, `PERSON=AGENT`, `CAR=PATIENT`, and `CROSSWALK=PLACE`, where `YELL`, `PERSON`, `CAR` and `CROSSWALK` are the features and `VERB`, `AGENT`, `PATIENT`, and `PLACE` are the roles. The goal is for the two vocabularies to end up with equivalent entries for features and roles, such that when one is presented with a scene, its partner can reconstruct the scene from the signals it receives, even if that particular scene has never before been seen.

When a scene is to be used for training, it is presented to both sides simultaneously, they turn it into sparse encodings on the wires[2] and learn from the coincidences between features, roles and activity on the wires. When a scene is to be used for testing, it is given to one side, and the test is judged successful if the other reconstructs the scene perfectly.

Assuming that the features are used sparsely and mostly independently in training examples (e.g. `CAR` and `YELL` usually do not show up at the same time, and when they do it is not usually in sequential examples), there is little interference between encodings. As a result, the two devices quickly jump to the same conclusions and agree on encodings for all of the features and roles. Until the system begins to approach capacity, the expected time to learn a new feature is just over 10 exposures and the

[1]as might be grown cheaply with the path creation operations in Section 4.4.

[2]The sparse encoding works much like Mooers' Zatocoding[29] or Minsky's K-lines[28].

time to learn a new role is a constant that depends on the structure of scenes. Once learned, signals can be arranged combinatorially in order to communicate scenes that have never been seen before.

Moreover, the whole process is tolerant of both noise and parameter variation. There is a large range of parameter and noise values where the vocabularies appear to always converge to produce 100% success rate. Pushed outside this area, system performance degrades gradually, making it easy to tune.

This is all very satisfactory, but there are a few caveats: it assumes synchronized execution, a supply of largely independent examples, clear differentiation of training and testing, and learns only equality relations.

6.1.2 How and Why v1.0 Works

Using communication bootstrapping in a theory of intelligence will require violating many of the assumptions that v1.0 depends on. Let us open up the box and examine the mechanism to see how the v1.0 architecture produces communication bootstrapping, so that we can figure out how to get around its limitations. For the full details of the mechanism, see [4].

Encoding Scheme The wires used by v1.0 hold one of four values: **neutral**, **true**, **false**, and **conflict**. A wire is **neutral** by default, but may driven to **true** or **false** by any number of inputs. When all the drives are for the same value, the wire takes that value; when both values are driven, the wire takes the value **conflict**.[3]

Features are encoded as a sparse subset of the large number of wires connecting the two vocabulary-building devices. We will call one of these sparse sets a *symbol*. A device transmits a feature by driving those wires (Figure 6-2). Conversely, a feature is detected when most of its associated wires are non-neutral.

In v1.0, the role of a feature is encoded as the percentage of the wires in its symbol that are driven to **true**. We will call a signal carried on the wires of a symbol an *inflection*. When an inflection is decoded, **conflict** and **neutral** values are ignored, and only the numbers of **true** and **false** values on a feature's wires are used. The feature is then assigned the closest role whose encoding is no more than a few percent different than the arriving signal.

Notice that the role encodings carry very little information per wire. The reason is the twist in the bundle of wires, which is modelled as an arbitrary permutation.

[3]Some asynchronous electronics are built this way. They are typically implemented as a "dual-rail" logic, with one wire carrying a true bit and one carrying a false bit.

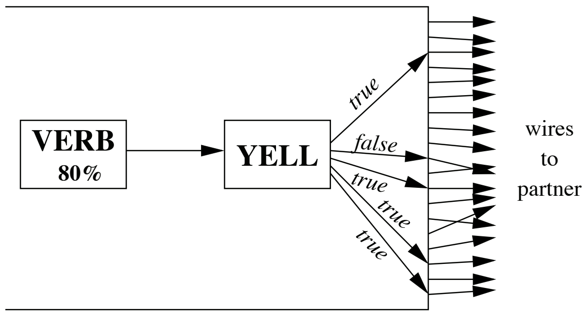


Figure 6-2: In the v1.0 architecture, a feature is encoded as a sparse set of active wires, and its associated role encoded as the percentage driven with **true**.

Since different symbols use different wires, the permutation affects each one differently. This means that if we are to use the same inflection with different symbols, it should not depend on the ordering of wires. The v1.0 inflections use resources very conservatively—too conservatively for our needs, since each symbol can be assigned only one inflection and it is costly to distinguish more than a few inflections. As a result, we will instead end up using an inflection encoding that is more like the symbol encoding.

Agreeing on Symbols In v1.0, the encoding for a feature has a four-stage life-cycle:

- **Creation:** When a device's input has a feature without an encoding, it proposes an encoding for the feature, choosing wires randomly. At the first opportunity, it also sets an initial guess for its partner's choice of encoding—every wire currently being driven.
- **Refinement:** During the next few appearances of a feature, the device refines its initial guess: the device transmits its own proposed encoding and eliminates any wire not driven by its partner. If there are ever too few wires remaining in the guess, the encoding is discarded and the process starts over.
- **Transition:** After a few rounds of refinement, the device assumes that any extraneous wires have been eliminated and begins transmitting on the wires of both its own proposed encoding and those wires remaining in its guess. Elimination of un-driven wires from the guess continues as before, and the encoding is discarded if there are too few.
- **Maturity:** After a few rounds of transition, the device assumes that its partner has either reached agreement or the attempt has failed for this encoding. The

device combines its proposal and guess, transmitting on both and eliminating from both, and the encoding is discarded if there are too few.

In effect, both devices make a proposal and try to guess the other's proposal. If one of the two picks up the other's proposal quickly enough, then they end up agreeing on an encoding.[4] If the attempt fails, then they try again with a different encoding.

This mechanism depends on three assumptions:

- Given two features, f and g, the conditional probability $P(g \in X | f \in X)$ of g appearing in an example X containing f is very small.
- Given that f and g appear in an example X together, and that f next appears in example Y, the conditional probability $P(f \cap g \in Y | f \cap g \in X)$ of g appearing in Y as well is very small.
- The probability of a feature in an example appearing to only one partner is very small.

The net effect of these three assumptions is that it is very easy to guess whether a wire is part of the partner's encoding proposal, since there is little overlap between encodings. In any short sequence of examples containing a feature, the wire will almost always be present if it is in the partner's encoding proposal for that feature, and almost always be absent if it is not.

We will not be able to depend on these input assumptions in more complex situations, such as our scenario of vision and hearing specialists observing a four-way intersection. The better we can approximate them, however, the more likely that we can induce communication bootstrapping.

Time Assumptions In v1.0, the wires in the bundle are used to send signals in both directions. When the two devices have agreed on an encoding for a feature, they will both drive the same wires. This was chosen to ensure that a feature only needed to be learned once, not once for each direction. It also leads to a hidden set of assumptions about time. The mechanism we will use later avoids this problem by using one communication channel for each direction and an abstraction that allows us to learn both mappings at once.

In v1,0, both devices are potentially sending signals on the same wires, so once a device begins driving the wires, its signals and the signals of its partner may become

[4]The encoding agreed upon turns out to always be either one of the device's proposals or the union of both proposals, plus occasionally a few wires added by interference.

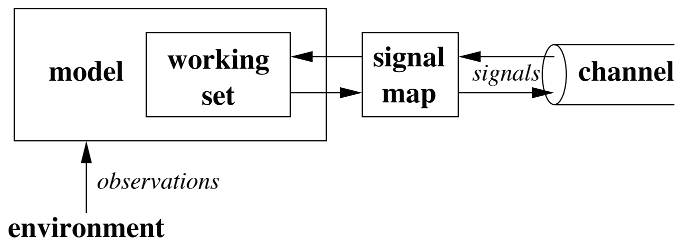


Figure 6-3: Core bootstrapping architecture for a specialist that participates in communication bootstrapping with one partner.

inextricably mixed. Indeed, the closer the devices come to agreement, the worse the mixing becomes. The v1.0 system handles this by making a set of timing assumptions:

- one of the devices starts signalling first
- the signal arrives at the partner before the partner starts to signal
- it is random which device transmits first
- there is time enough between examples for the wires to return to neutral

The net effect of these assumptions is that precisely one partner gets a clean view of the other's signal, and that they progress toward agreement at approximately the same rate.

We will not be able to retain any of these assumptions, since they depend on synchronization and transmission brevity that I am no longer willing to assume. We still need, however, to ensure that the partners get a clean view of each other's signals and that they progress towards agreement together.

6.2 Core Bootstrapping Architecture

I will generalize the original architecture to a more flexible design. Let us start by considering only a specialist that communicates with one other specialist (which we will call its communication *partner*). Figure 6-3 shows a core bootstrapping architecture for a specialist with one partner:

- A stream of *observations* arrive from the specialist's *environment*. For many specialists, these observations are samples of sensory input. For example, the vision specialist in the stoplight scenario receives observations in the form of sets of objects and their associated features and relations.

- The arriving observations are used to update the specialist's *model* of its environment. A specialist is likely to have a model and update mechanism specific to the type of observations it receives. For example, the vision specialist might incorporate Spelke's principles of cohesion, continuity, and contact[37].

- Communication between a specialist and its partner is carried on a bidirectional *channel* that connects them.

- At any time, a *working set* of model elements is designated. Signals describing the contents of the working set are sent frequently to the partner. Likewise, signals encoding the partner's working set arrive periodically over the channel. For example, the vision specialist might designate a few foreground objects as its working set and communicate only those objects and their relations.

- The translation between model elements and signals is handled by a *signal map.* The signal map creates signals to encode model elements and searches for relations between model elements and signals from the partner.

  For each potential relation considered between a model element and a signal component, the map finds positive and negative *examples* by comparing the times when the elements and the signal are present. When the positive examples dominate, the relation holds. Finding a relation indicates agreement with the partner that a signal is meaningful, though they may disagree about what it means. When meaningful signals arrive, the map sends an acknowledgement to the partner and may modify the working set according to the relations that have been discovered.

  For example, the vision and hearing specialists should come to agree on a signal that means "walk light" to the vision specialist and "cuckoo" to the hearing specialist. When the cuckoo sound starts up, the hearing specialist will send this signal, likely changing the working set of the vision specialist to include an expectation of seeing a walk light.

If we map the v1.0 architecture onto this core bootstrapping architecture, the v1.0 vocabulary becomes the signal map and the scenes are the observations delivered by the environment.

6.3 What Is a Good Agreement?

A good agreement on a signal is one that results in improved models of the environment. The improvement might be a direct result of making cross-modal information available: when the vision specialist sends a messages that a car is in contact with pedestrian, the hearing specialist predicts a scream.

Translating back and forth may also function as cooperative reasoning: after the hearing specialist predicts the scream, that signal might cause the vision specialist to predict the person moving downward, which leads the hearing specialist to predict sirens, which leads the vision specialist to predict the appearance of an ambulance. Though neither specialist understands accidents on its own, together they would have predicted that if a car hits a person, then an ambulance will come. The problem is how to measure whether the model has improved.

I will measure the quality of agreements produced by communication bootstrapping in two ways, prediction quality and relation analysis.

Prediction quality is the simpler measurement, but generally produces less insight. To measure prediction quality, we compare the predictions of a specialist's model to its observations. A simple metric captures aspects of both precision and accuracy: for a set of c changes in observation and p predictions, let u_i be 1 if the ith change was unpredicted and 0 if it was not, let s_i be 1 if the ith prediction was satisfied and 0 if it was not, and let t_i be the number of seconds time spanned by the ith prediction (e.g. if the the walk-light was predicted to turn on sometime in the next five seconds, that prediction's t_i would be five). We can then express prediction quality as

$$Q(p,c) = \frac{1}{p}(\sum_{i=1}^{p} \frac{s_i - k_1}{t_i}) - \frac{k_2}{c}(\sum_{i=1}^{c} u_i)$$

where the two ks are constants for weighting the relative importance of correct prediction, unfulfilled prediction, and failure to predict. If the prediction quality rises over time, then the model is likely to be improving.

The problem with prediction quality, however, is that it is not clear whether better prediction is a good indicator of the quality of agreements. A system's improving understanding of the world might actually lead it to make *more* mistakes, but have the type of mistake evince a deeper understanding. For example, if the car almost braked in time and tapped but did not injure the pedestrian, then a system that predicted screaming and sirens would appear to have lower prediction quality than one that knew nothing about accidents. If some specialists control actuators, then there are more ways the metric can fail: for example, a system might increase its prediction quality by avoiding exploratory behavior and repeating the same actions over and over. Finally, there is no clear way to distinguish between important and unimportant mistakes using prediction quality: predicting which direction a pedestrian crosses the street has the same value as predicting whether they will be hit by a car while doing so.

Relation analysis answers these problems, but is more labor intensive and subject to human error. For relation analysis, we crack open the signal maps and look to see what dynamics of the environment are captured when we compose the two maps together to get relations between model elements. For example, relation analysis could discover the accident to ambulance reasoning chain without having to observe it in action.

I will analyze systems using both of these techniques. In general, however, we will be more interested in what relations have been learned than the quality of predictions, because prediction quality is likely to be abysmal for a system as complex as our four-way intersection scenario. Prediction quality serves as a sanity check (making sure the trend is in the right direction) rather than a benchmark.

6.4 Signal Encoding

We now need to decide what sort of signals the specialists will be trying to agree on. This decision must be made carefully, as it will affect nearly everything that follows on from this point.

The requirements that must be taken into account are:

- Signals should be combinatoric. The individual signals in the signal map should be able to be combined to express a factorial number of situations. When new signals are agreed upon, they should be able to combine with those that already exist.

- Signals should be expressive. It should be the case that any portion of a model can potentially be communicated to a partner, and that the models be potentially able to approximate anything a specialist might need to represent.

- Signals should degrade gracefully. It should be difficult for a small amount of noise to change a signal into a different signal. If part of a compound signal is corrupted or not yet agreed on, it should not affect the interpretation of the rest of the signal.

- A large set of signals should be affordable according to the cost model from Section 4.4.

- Complicated compound signals should be able to be communicated quickly. Given the cost model in Section 4.4, this means that the channel must carry many signals at once and that these signals must be able to be recognized and interpreted independently.

- Signal structure should be highly constrained. The more options there are, the harder it is to produce communication bootstrapping.

The signal structure I have chosen is a sparse encoding of symbols marked with inflections, similar to the one in v1.0. There may be many other good signal structures that fit these requirements. For this sort of exploratory engineering, however, we just need *some* reasonable approach, and this structure is known to be workable.

6.4.1 Symbols and Inflections

In this symbol/inflection encoding scheme, symbols encode features of the specialist's model and inflections encode the relationships between features. The component signals are individual symbols and inflections. For example, the visual relations specialist has features like **CAR**, **RED** and **STREET** and relations like **TYPE**, **COLOR**, and **BELOW**. Directional hearing, on the other hand, has features like **CUCKOO** and >**100db** and relations like **TYPE** and **LOUDNESS**.

A compound signal is a set of symbols, marked with inflections to show their relationship. For example, the visual relations specialist might send a signal for a red car on the street as a set of three pairs **CAR=TYPE**, **RED=COLOR**, and **STREET=BELOW**.

We will assume that there are many more symbols than inflections, and that while the number of symbols may grow, the number of inflections has a strict upper bound (there are only so many relations used in the model). Potentially, any symbol might be paired with any inflection, though in practice some will never be paired up.

This encoding scheme places little constraint on the model, except that it be symbolic. Even that constraint is fairly weak, since a continuous value can be approximated with a set of discrete values (e.g. >**100db**). In this work, I choose to think only about models that are semantic networks containing objects and relations that connect to objects or other relations. Again, I am not claiming there is anything special about these models other than that they are easy for me to extract from my simulator and have been easy for me to think about as I explore communication bootstrapping.

6.4.2 Channel

In order to make the channel affordable, I assume that it consists of a large number of arbitrarily connected paths. That way, it can be grown using a small number of communication path operations, and complicated signals can be sent quickly on the parallel paths.

We encode symbols as a sparse subset of the paths, and inflections as a sparse sequence of pulses sent on a symbol's paths. Since the symbols are sparse subsets, any pair of symbols is likely to have few paths in common, and we can expect to be able to send many symbols simultaneously with little interference between them. Likewise, since inflections are encoded as a sparse pattern of pulses, we can send several inflections on each symbol and expect little interference between them.

Since inflections are encoded temporally, they are much more expensive than symbols (which take advantage of the massive parallelism in the cost model). We are assuming there is only a relatively small fixed number of inflections, however, so the greater expense is not problematic.

Encoding signals on the channel in this way meets all of the requirements specified in Section 6.4. Symbols are cheap, and signals can be communicated quickly. The sparse signals combine through superposition, and can be encoded and decoded independently, also ensuring graceful degradation. Any portion of a semantic network can be communicated if we add a pair of inflections to indicate the current and next focus and use these to traverse the semantic network one step at a time, as will be seen in Chapter 8.

All that remains is to determine how a system can produce communication bootstrapping with this encoding scheme, and whether the signals agreed upon can capture the structure of the system's environment. The rest of this document is devoted to teasing out the answer to those questions, one step at a time.

Chapter 7

Agreeing on Non-Equality Relations

Now that we have a general design for communication bootstrapping, we can start to fill it in and understand how to make it learn things. We will start with learning simple relationships between the models of two specialists.

For example, some reasonable things that our system ought to be able to learn in the stoplight scenario:

- When an engine is heard, a car will likely soon be seen.
- When an engine is heard, an SUV (a particular type of car) may soon be seen.
- Whenever a car is seen, an engine is heard.
- Seeing a yellow light is followed by hearing a cuckoo (the audible walk signal).

While none of these is universally true, they hold almost all the time in the scenario during the day because the vehicles are all cars, its streets are too narrow for parking, and there are a lot of pedestrians.

When we start trying to have our system learn about these sorts of relations between specialists, the assumptions that allowed communication bootstrapping in the v1.0 architecture no longer hold.

First, there may be many different relations per model feature. For example, engines are related to sedans, pickups, SUVs and vans, all of which are also cars. This means that the probability assumptions that v1.0 depends upon will be violated. On the one hand, features in a model may be correlated, violating the assumption that features rarely appear together: most cars are sedans, so the probability $P(sedan|car)$ is high. On the other hand, these more complicated relationships mean that a feature often appears in only one partner: most cars are not pickups, so the probability $P(pickup|engine)$ is low.

If the signal agreement mechanism looks at each communication path independently, the conditional probability of interference may be as high or higher than the probability of activity on the signal communication paths. For example, if there is an average 5% interference between signals, and only one out of every 20 engine noises is made by a pickup, then it will be hard for the directional hearing specialist to distinguish between interference and a signal about pickups: any given communication path is equally likely to be activated by either source.

Second, relations are no longer symmetric. For example, the cuckoo sound follows a yellow light, and is followed by people in the street. This means that the signal maps of communication partners will no longer be equivalent. For example, there may be no relationship between the yellow light and the sound of footsteps.

Finally, related events are separated in time. For example, a car's engine begins to be heard long before it appears the field of view. In order to learn relationships between such events, they must be brought together into a single example.

All together, these three things mean it is no longer reasonable to assume that experiences come pre-digested into examples, and that it is no longer feasible to learn the meaning of a signal in terms of individual communication paths. Indeed, it is not clear that it is even possible to segment the stoplight scenario into examples without pre-judging what should be learned. Consider a pickup truck that runs the light toward the end of a walk signal: should an example derived from this situation stop when the green light begins, when the pickup disappears, when the light changes from green to yellow, or some other time entirely?

I address these challenges with a two part solution:

- Separate the process of agreeing on signals from the process of agreeing on their meaning. This eliminates the problems of interference in relations where one side has a low conditional probability.

- Find examples using time interval relations between pairs of symbols or inflections.

Figure 7-1 shows my design for a signal map that uses this solution. By the end of this chapter, we will see how to build each component and how they fit together in this design.

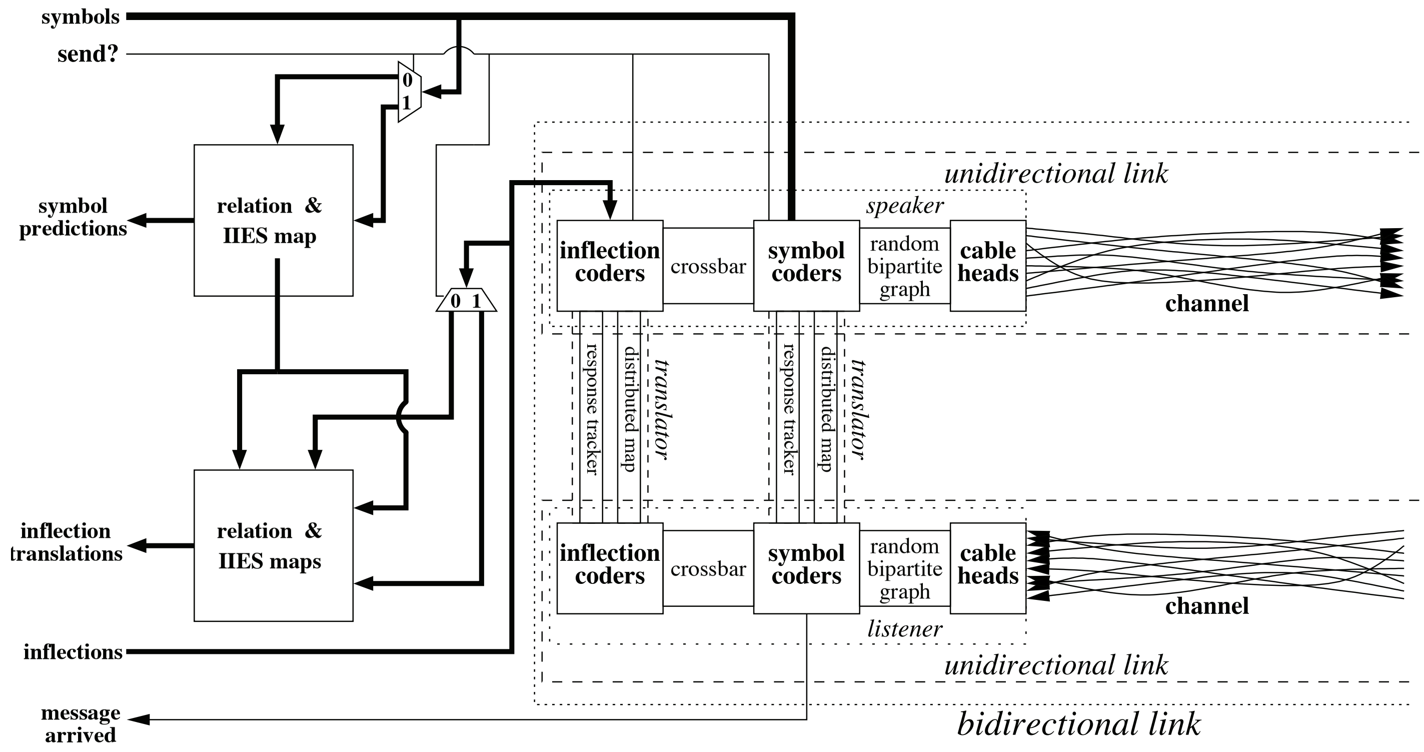


Figure 7-1: Design for a signal map separating agreement on signals from their interpretation. Thick black lines are distributed maps (defined in Section 7.1.1) used as wiring busses.

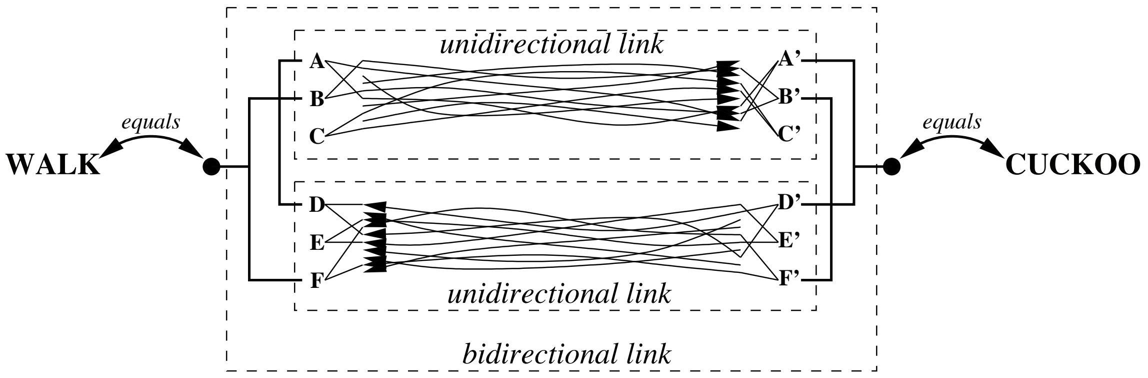


Figure 7-2: Two abstractions separate agreement on signals from agreement on their meaning: one-way bundles self-organize into a unidirectional link and aligned encodings for symbols and inflections. The one-way encodings are then paired up to form a bidirectional link carrying abstract symbols and inflections.

7.1 Separating Form and Meaning

In the v1.0 architecture, probability assumptions make it possible to use a simple process to agree on both the encoding and the meaning of a signal. Now that we cannot depend on these assumptions, I separate the two processes so that there is less opportunity for them to interfere with one another.

I separate agreement on signals from agreement on their meanings by means of a pair of abstractions (Figure 7-2). At the lowest level, a twisted bundle of communication paths self-organizes into a *unidirectional link*. One specialist is designated as the *speaker*, the other as the *listener*, and they cooperate to create a pool of "meaningless" symbols and inflections that can be reliably transmitted from the speaker to the listener. No interpretation is assigned to these signals: they are waiting for use to give them meaning. These pools are exposed for use differently on the two sides: on the speaker side, symbols and inflections can be allocated and used to compose messages, while the listener side simply reports the contents of the most recent message.

Two unidirectional links, one in each direction, are then bound together to form a *bidirectional link* between two specialists. Symbols from the unidirectional link pools are paired up to produce a pool of bidirectional symbols; the same is done for inflections. Again, there is no meaning attributed to the signals, but they are made available with an interface that allows each specialist to allocate and deallocate them, and to send and receive signals simultaneously.

Using this design, it is possible to build an agreement mechanism that looks for pairwise relations between the contents of the working set and the set of arriving symbols and inflections. The problems of encoding and interference are hidden beneath an abstraction barrier, simplifying the agreement problem. Indeed, it does not even matter that the signals are different in each direction.

This design eliminates both the problems of communication bootstrapping with individual communication paths and the problem of interference between incoming and outgoing signals.

7.1.1 Three Building Blocks

There are three structures that I use as building blocks, both here and elsewhere. The first is a *competition* between elements of a set, which I use for symmetry breaking. The second is a *random bipartite graph*, which connects two sets together with a random set of links. The last is a *distributed map*, which uses two random bipartite graphs to create arbitrary one-to-one functions between two sets.

Competition We will often need to choose a single device from a set, breaking symmetry in a group of otherwise identical devices. Given random number generation and communication with nearby devices in the set, there are many ways for a set of competing devices to break symmetry and elect a winner.

One that is particularly useful is the sorting algorithm presented by Butera in the discussion of streaming audio in [7]. Each device starts with a unique number, then devices exchange numbers in a parallel bubble sort towards a defined location. This sorting algorithm is highly resistant to variation and error in devices and the network that connects them: numbers simply flow around the fault.

If we modify this sorting algorithm to start with random numbers and sort toward the lowest number (identified by gossip) rather than a defined location, then a set of n devices will select the lowest number as the winner in at worst $O(n)$ time. A small further modification allows the modified algorithm to run in a subset of the original space. The sorting algorithm leaves the devices nearest the winner containing the runner-up, so if the winner withdraws from the contest, the next winner can be selected in (amortized) constant time.

Competition can often take place offline, allowing even the first winner to be chosen in constant time. For example, when we allocate symbols, the choice of the next symbol to be allocated can be made offline through an incrementally evolving competition. Finally, if we bias the initial random number generation, then we also bias the competition, giving priority to some devices.

This algorithm is not necessarily the best for the job, but serves to show that the task can be accomplished quickly and reliably.

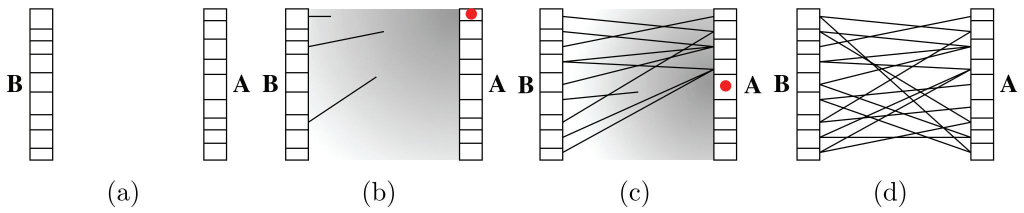


Figure 7-3: One way of creating a random bipartite graph between two sets, A and B (a) is to have devices of B randomly start growing communication paths toward A, while the devices of A compete to signal and the winner (red dot) guides the growing paths towards itself (b). When k paths have arrived at the winning element of A, it withdraws from the competition and a new device becomes the winner. This continues from device to device (c) until there are no devices left in the competition and the graph is finished (d).

Random Bipartite Graphs The random bipartite graph is simple: given two sets of devices, A and B, link every device $a \in A$ to k randomly chosen devices of B.[1] Random bipartite graphs will be a useful tool for creating signals that do not interfere with one another, and can be used to implement references in data structures.

A random graph can be constructed using the operations in Section 4.4 by converting random number generation into randomness in space. The devices of one set each send a signal for a communication path to grow toward (either different signals or the same signal at different times), while the devices of the other set randomly choose which signal to grow paths toward.

For example, we might have the devices of B randomly choose times to start growing communication paths toward A (Figure 7-3). On the other side, the devices of A compete with one another, and one winning device emits a "grow toward me" signal. When k paths have arrived at the winner, it withdraws from the competition and a new device becomes the winner and begins emitting the signal. This continues from device to device until there are no devices left in the competition and the graph is finished.

Implemented simply, growing a random graph in this manner requires a constant amount of encoding and time proportional to the number of devices in A. The constant factor for the growth time can potentially be made quite low, however, through two techniques. First, if the growth of a path experiences random perturbations, then we can guide growing paths toward a small region of devices rather than a single device. Second, we can adjust the time constant so that there are many more than k communication paths growing at once, allowing partially grown paths to redirect to the new signal source. This effectively "pipelines" path creation, but if paths are

[1]It is acceptable if randomness leads some devices to be chosen twice.

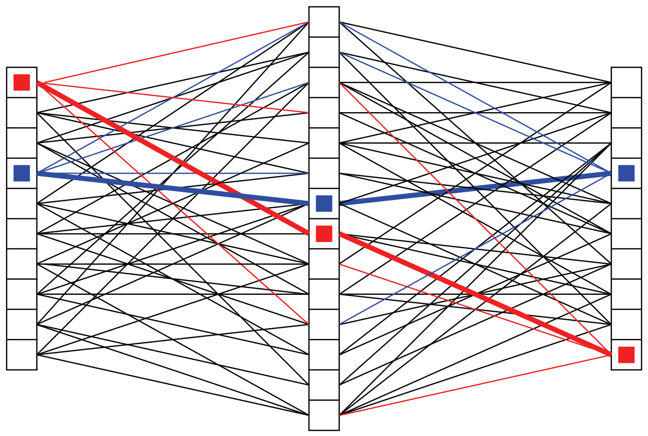


Figure 7-4: A distributed map connects two sets through an intermediate set of rendezvous points. If each set element connects to a sparse random subset of rendezvous points, then a small oversupply of connections and rendezvous points will make it highly probable that the rendezvous points can be configured to represent any one-to-one function mapping from one set to the other.

created too quickly, their distribution will be biased by the distance they must travel. The threshold where the bias becomes significant depends on the amount of random perturbation in the growth of a path, the geometry of the two sets, and the distance between them. Finally, variation during development results only in a small perturbation of distribution of graph edges—something that most systems using a random graph will tolerate handily. The mature random graph, unsurprisingly, takes $O(kA)$ hardware, relays communication in time proportional to the length of the links, and has error manifest as noise in the communication.

Distributed Map A *distributed map* is a device for dynamically creating communication paths between elements of a set A and elements of another set B. These communication paths act as a one-to-one mapping between subsets of A and B, carrying signals in either direction.[2] Appendix C.1 contains a full data sheet for this device.

This device is useful both directly as a map, and as a bus for connecting together sets that are created independently. We will use it two ways: first, to pair together symbols and inflections from the two unidirectional links, and second, to connect model elements to the signals allocated to represent them.

A distributed map connects together two sets through an intermediate set R of rendezvous points. The number of rendezvous points in the set R is equal to the size

[2]A crossbar gives us this same functionality and more. The advantage of a distributed map is that it is significantly cheaper than a crossbar.

of the smaller set, multiplied by a small oversupply constant o_s. Each set connects to the rendezvous points with a random bipartite graph.[3] In order for any two elements to have an expected rendezvous size r_s of shared rendezvous points, each set element connects to $k = \sqrt{|R| \cdot r_s}$ rendezvous points.[4]

To create a communication path between two elements, we send a special creation signal from each element to its rendezvous points. The rendezvous points that receive a signal from both sides compete (with preference given to rendezvous points that are not yet allocated). The winning rendezvous point then selects the paths that carried the creation signal: it will subsequently relay signals carried on these paths and ignore all signals on other paths except for creation signals. Finally, the rendezvous point sends an acknowledgement signal back along both paths to let the requesting elements know that creation succeeded.

As more paths are allocated, there is an increasing chance that all of the shared rendezvous points for a new path will already be allocated. In this case, one of the already allocated rendezvous points is the winner of the competition and the path it previously carried is unceremoniously terminated. Likewise, if a set element is connected to a new path, it disconnects from its previous path.

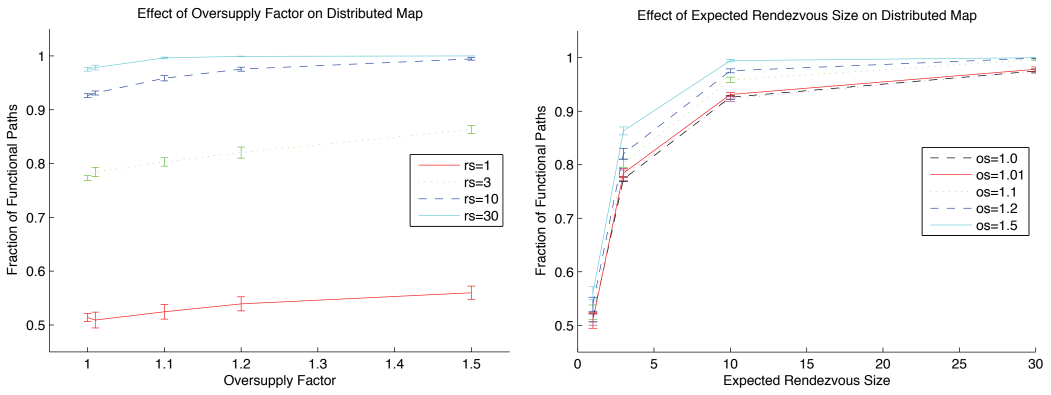
Communication paths can also be destroyed unilaterally: if an element sends a special deletion signal to its rendezvous points, then the rendezvous point for its communication path will reset, discarding its path selections on both sides.

Only a small oversupply factor and rendezvous size are needed in order to ensure that a distributed map can simultaneously maintain paths to nearly every element of the smaller set. Figure 7-5 shows graphs of success rate in creating a random permutation map between two sets of 1000 elements with various oversupply factors and expected rendezvous sizes. No element ever connects to the wrong element of the other set, and for even modest levels of o_s and r_s nearly every element connects with its pair in the other set.

The cost to create a distributed map between two sets A and B with oversupply multiplier o_s and rendezvous size r_s is the cost of creating the rendezvous points and the two random bipartite graphs. The encoding cost is constant, as is the time to create a path, delete a path, or send signals across all the existing paths. Letting $|R| = min(|A|, |B|)o_s$, it takes $O(\sqrt{r_s|R|}(|A| + |B|))$ hardware to implement the map, and $O(min(|A|, |B|))$ time to grow that hardware. Variation during development will result only in a small change in the number of links or rendezvous elements. In effect,

[3]This design is similar to that used by butterfly graphs[10] and other interconnects based on expander graphs.

[4]A single link has a $k/|R|$ chance of connecting to a shared rendezvous, so k of them have an expected size $k^2/|R|$. We then solve $k^2/|R| = r_s$ to obtain the value of k.



(a) (b)

Figure 7-5: A small oversupply factor and rendezvous size are all that is necessary to ensure that a distributed map can simultaneously maintain paths to nearly every element of the smaller set. These graphs show the success rate for a random permutation map between two sets of 1000 elements (the same data is plotted on each graph, but with different axes to highlight the effects of the two variables).

this is no more than a small perturbation in the constants o_s and r_s, which we have already shown will have little effect on the behavior of the device.

Given the large capacity and low error of distributed maps, we can abstract this device as either a reliable bus or as a map that has a small chance p_d of deleting a random mapping whenever a new mapping is added. Doing this will allow us run simulations much faster.

Using distributed graphs and competition, we can organize consistent mappings in any graph of sets—meaning that if we send a message along an arbitrary loop in the graph, it will return to the source unaltered (except by noise in transmission). Each link in the graph is a distributed map, and each set supports competition. To align a single element across all the sets, we start with a competition that picks a single element in each set, then connect all those elements in the distributed map. Once all the connections are finished, the process repeats until enough elements have been selected. Finally, a designated seed set selects elements to test, one by one, and any that do not successfully propagate over every link are discarded. The surviving elements form a consistent mapping over the graph.

7.1.2 Unidirectional Link

A unidirectional link connects two specialists, allowing messages composed of abstract symbols and inflections to flow from one to the other. A unidirectional link differs from a distributed map in that it tolerates noise, handles more complex messages, and

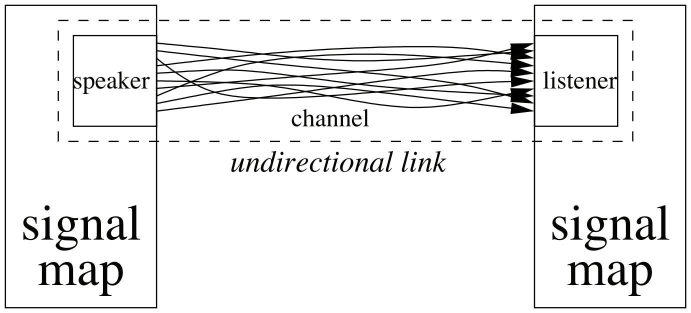


Figure 7-6: A unidirectional link connects two specialists. Messages are given to the speaker component of one specialist's signal map. The message is encoded into signals on the channel connecting the two partners, then decoded back into a message by the listener component of its partner's signal map.

can send over a long distance channel that connects arbitrary pairs of elements.[5] The link consists of a component in each specialist's signal map and a channel between the two components (Figure 7-6). In ordinary use, a message is given to the speaker, which encodes it into signals on the channel. The listener then decodes the signal back into the original message.

The unidirectional link assumes that there are many more symbols than inflections, and that each message contains only a small number of symbols and inflections.

The description of the unidirectional link can be broken into three parts:

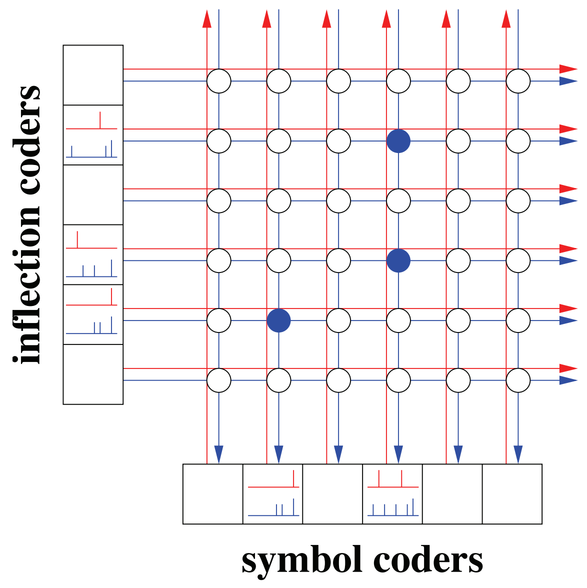
- how messages are encoded and decoded
- how the rest of the signal map knows what symbols and inflections to use
- how the speaker's encoding map self-organizes to match the listener's decoding map

The complete data sheet for this device is included in Appendix C.2.

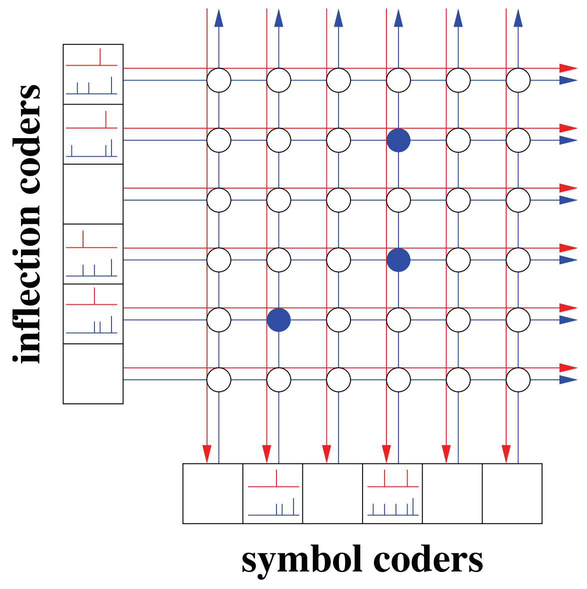
Encoding and Decoding The speaker and listener in a unidirectional link are mirror images of one another. Each maps signals to messages using two sets of coders—one for symbols and one for inflections—connected by a crossbar.[6] Each communication path in the channel is connected to a *cable-head* device at each end, which does some processing to support self-organization. The mapping between cable-heads is almost all one-to-one, but arbitrary within that constraint. Finally, the set of symbol coders is connected to the array of cable-heads with a sparse random set of communication paths.

[5]Arbitrary is *worse* than random, because random at least guarantees a particular distribution.

[6]In this case, a crossbar is affordable because we are assuming that the number of inflections is much smaller than the number of symbols.



(a) Sending



(b) Receiving

Figure 7-7: A message is represented by the set of active junctions (blue dots) in the crossbar connecting symbol and inflection coders. The crossbar communication paths carry two signals: selection (red) and code (blue). In the speaker (a), the message is specified with simultaneous selection pulses to set the junctions, then code pulse sequences combine as they flow from the inflection coders to the symbol coders. In the listener (b), the junctions are set when code pulse sequences from the inflections intersect with those arriving through the symbol coders. The message may then be read by sending a selection pulse on each inflection and observing which symbols it arrives at.

A message is represented as the set of active junctions in the crossbar: each junction represents a particular symbol/inflection pairing. Notice that this means that a symbol cannot be transmitted without any inflections.

The communication paths of the crossbar carry two types of signal: selection signals used to send or receive the message, and code signals used to carry the message from the speaker to the listener.

To send a message (Figure 7-7(a)), the user first clears the crossbar, then sends a set of selection pulses from both the inflection and symbol coders. Wherever the pulses intersect, the junction is activated. Since there are many more symbols than inflections, it will usually be most efficient to pulse the inflections of the message one by one, and pulse all the symbols with a given inflection together.

For example, consider sending the message **{PERSON=AGENT & BENEFICIARY, RED=PATIENT, WALK=PATIENT, CAUSED=VERB}**, which we could render into English approximately as "The person caused a red light and walk signal for their own benefit."[7] This message would be encoded into the crossbar with four selection pulses, one for the inflection **PATIENT** and symbols **RED** and **WALK**, another for the inflection **AGENT** and symbol **PERSON**, and so on.

A message is encoded as a short burst of activity on the channel. Each inflection coder holds a sparse pattern of p pulses scattered through b time slots in a burst, plus a synchronization signal. Figure 7-7 shows the synchronization signal as a strong initial pulse, but it might also be a change in base activity level or some other signal. When self-organization is complete, all of the inflection patterns will be different.

Once the crossbar has been set, all the inflection coders send their pattern as a code signal, which flows through the active junctions to the symbol coders. When a symbol has more than one inflection, their patterns superimpose as they flow through the crossbar. The symbol coders relay these patterns to the subset of cable-heads they have chosen to encode the symbol (Figure 7-8). The sequences once again superimpose at each cable-head, then propagate along the channel from speaker to listener.

In the listener, the process is reversed (Figure 7-7(b)), beginning with the cable-heads relaying the signals they receive onto the symbol coders. Each symbol coder monitors a chosen subset of cable-heads, and if at least d_s are active, it relays the consensus pattern (pulses appearing from at least d_c cable-heads) as a code signal into the crossbar. At the same time, the listener's inflections transmit their patterns as code signals, and whenever at least d_i pulses coincide at a junction, that junction becomes active.

[7] I have chosen this somewhat awkward sentence in order to have a symbol with two inflections.

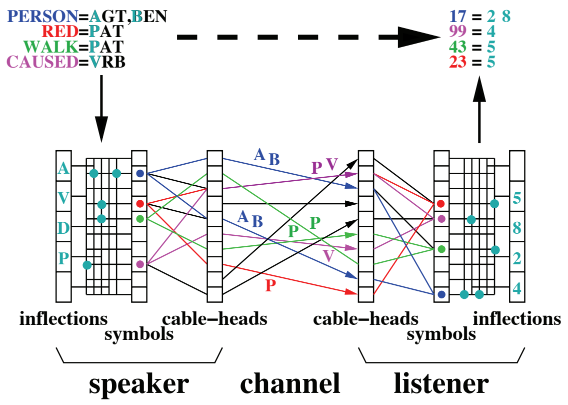


Figure 7-8: A unidirectional link is implemented using sparse coding on top of a random wiring pattern. Symbols are encoded as sets of active communication paths and relations as the pattern of activity on a symbol. There is initially no interpretation provided for signals, however, as represented by the names on the left becoming numbers on the right.

The message can now be extracted from the crossbar using selection signals. A selection pulse is sent through each inflection in turn and flows through the active junctions of the crossbar. The message may then be read by observing which symbol coders each selection pulse arrives at.

Note that the ability of the listener to decode the message reliably does not mean that it knows anything about what the symbols and inflections are intended to mean. Interpreting the message is a problem that the user of the link must handle.

Allocating Symbols and Inflections The unidirectional link provides only a limited number of symbols and inflections that can be used to compose messages. At most, there is one symbol for each symbol coder and one inflection for each inflection coder.

Unless the speaker and listener have an aligned encoder/decoder pair for a particular symbol or inflection, however, it is worthless for communication. Thus, generally only a fraction of the coders will be available for use.

With regards to usability, each coder in the speaker has four states. When *disabled* or *immature*, a coder is not part of an aligned encoder/decoder pair, and cannot be used. When *mature*, a coder is part of an aligned encoder/decoder pair and is part of the pool of usable coders. When *allocated*, a coder is part of an aligned pair and has also been provided to the user of the link in response to a request for a symbol or inflection.

Initially, all coders are disabled. When given time to self-organize (while the link is not being used for communication), the coders mature at a linear rate until most of the coders are mature. At that point the process slows down. I will explain how coders mature in the next section.

When the user of the speaker needs a new symbol or inflection, it activates a request line into the appropriate set of coders. The mature coders compete with one another over a shared link, and the winner becomes allocated and signals back to the user, creating a connection in the distributed map connecting the user to the set of coders. If there are no mature coders available, then the user receives no response. This serves to notify the user that the attempt at allocation has failed, so that the user may adjust its behavior appropriately.

The user may also deallocate a coder, once the user is done with it. When a coder is deallocated it resets itself, returning to its original disabled state, in order to avoid confusion between its last use and its next use. This is necessary because the deallocation takes place only on the speaker side and the listener cannot distinguish between a coder that has become unaligned and a coder that is simply not being used. When the speaker returns a coder to the disabled state, it is likely to be paired with a different coder in the listener when it next becomes mature.

Aligning the Speaker and Listener When the hardware of a unidirectional link is first created, the encodings for symbols and inflections in the speaker do not match those in the listener. Self-organization produces an aligned set of symbols and inflections, which may then be used for communication.

Self-organization is interlaced with ordinary use of the link. During times when the link is idle, the speaker selects immature coders as targets for alignment and trains them to maturity. The alignment of coders is constantly refined, both during these training periods and while the link is being used for communication.

A reasonable way we might expect the link to operate, then, is long periods of communication in which mature coders are allocated and then reset through deallocation or dealignment, interspersed with periods of training when the collection of mature coders is replenished. In general, we would also expect the inflections to change rarely once established, since the type of relationships in a specialist's model is likely to be stable, while the collection of symbols is likely to range from ones that never change to ones that are allocated and discarded almost immediately, as the specialist's model changes over time.

As soon as the first symbols mature, they can be allocated and used; once mature, a symbol is rarely interfered with by the ongoing self-organization (and a mechanism could be added to further protect critical symbols). Eventually, self-organization reaches a stable point, in which almost no changes occur except in response to external

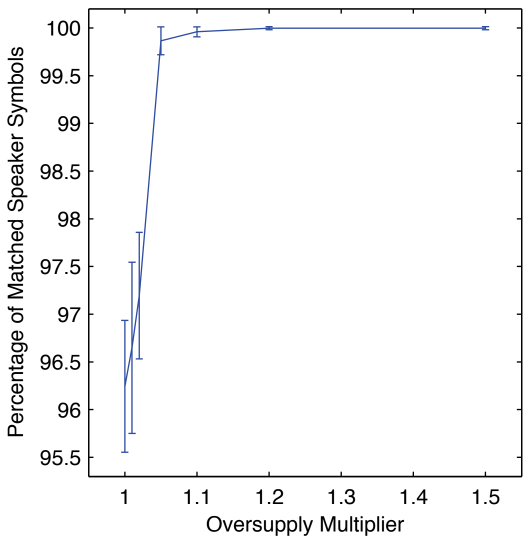


Figure 7-9: A small constant factor of coder oversupply is sufficient for efficient self-organization. This graph shows the percentage of 1000 symbols that have found matches after 20,000 rounds of self-organization at various levels of oversupply (vertical bars show twice standard deviation). Inflections are always fully matched.

perturbation.

Self-organization proceeds one coder at a time, beginning with all symbol coders and inflection coders disabled in both the speaker and the listener. Whenever a new coder is needed, the disabled coders compete to be the next to become immature. In the speaker, there is one immature coder of each type: together they serve as a target for alignment[8]. In the listener, coders are allocated in response to unrecognized activity on the communication paths or in a pattern, and there may be many immature coders at the same time.

The self-organization process is driven by babble generated by the speaker: the babble includes the current target, but is otherwise randomly generated from mature coders, growing in complexity as the number of mature coders grows. As the collection of mature coders grows, the babble exposes conflicts that are corrected by dropping communication paths or reallocating coders. Because the speaker chooses which coders are used, there is no way for the listener to distinguish between an unmatched coder and one that is merely rarely used. Accordingly, mature coders will occasionally reallocate when the supply of disabled coders is low, and the listener needs an oversupply of coders compared to the speaker in order to avoid thrashing when almost all coders are mature.

[8]When there are no immature coders remaining, the target is placed on a mature coder but does not disrupt its alignment.

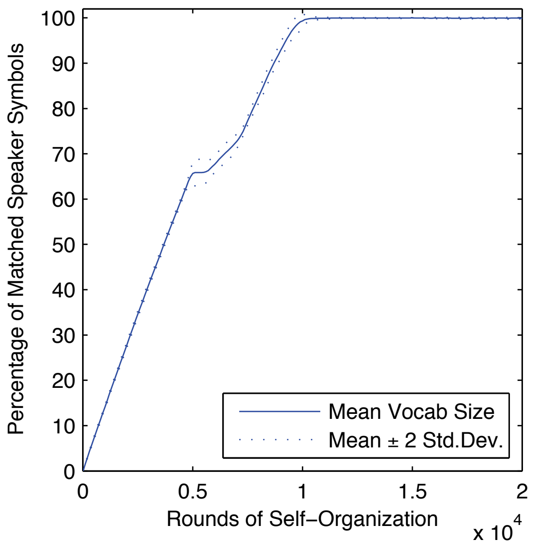


Figure 7-10: When vocabulary size is far from capacity, the mature vocabulary grows rapidly. After an initial period of fast linear growth, symbols that failed to rendezvous in their first attempt find complements at a slower rate.

The speaker inflections adapt stochastically. Each starts with a random sparse pattern. When one transmits, it sends its pattern to all the others over an all-to-all network, and when it overlaps too much with another active inflection, one of the two reinitializes with a new random pattern. All other coders adapt by comparing patterns of activation, using the codetector from Chapter 5 to decide whether particular elements are part of the pattern. When enough components of an coder's pattern are accepted, it matures.

Inflection alignment is easy, since the incoming pattern is filtered by consensus in its carrier symbol. A symbol coder, on the other hand, is attempting to rendezvous with its complement on a small subset of its communication paths; listener symbol coders push a single bit of feedback up their chosen communication paths to the speaker to enable this rendezvous. The rendezvous is generally very small compared to the number of cable-heads a symbol coder connects to, since it must be possible to connect any arbitrary pair of symbol coders.

As a result, we must take steps to avoid symbols becoming entangled. When feedback arrives at a speaker cable-head, it relays the pattern it transmitted so that symbol coders can ignore contaminated feedback information. Also, when a symbol coder makes rendezvous on too many communication paths, it must prune them slowly to ensure there is precisely one symbol coder connected to the other side.

Finally, any symbol coder that connects to too few communication paths deal-

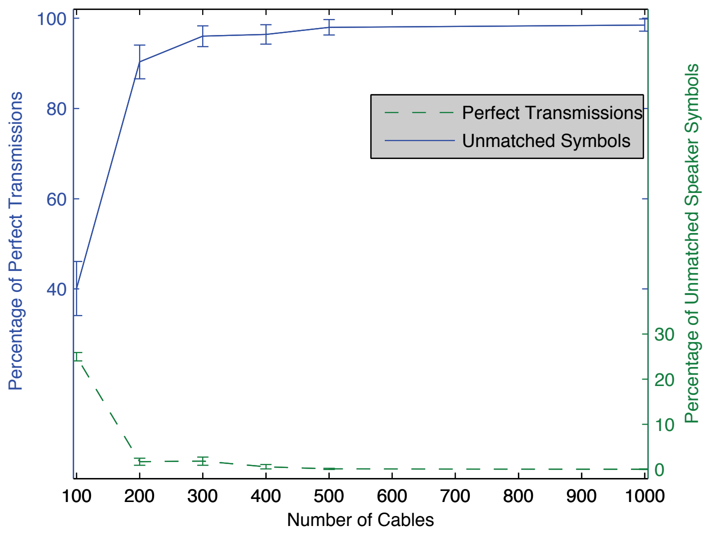


Figure 7-11: As the number of communication paths in the channel decreases, self-organization gradually declines in effectiveness, eventually degrading badly. This graph shows the final behavior for 20,000 rounds of self-organization.

locates itself and reset. The oversupply in the listener means that there are many opportunities for any speaker coder to find a complement, so if one attempt at alignment fails, the next is likely to succeed.

Behavior of Unidirectional Link Experiments in simulation show that the link's self-organization is fast, requires little oversupply, and is resilient against noise and small variations in the parameters.

Except where otherwise noted, simulation will be performed with the same set of parameters. The speaker has a potential vocabulary of $s = 1000$ symbols and $i = 20$ inflections; the listener has an oversupply multiplier of $o_s = 1.1$ times as many of both. The channel has $c = 1000$ communication paths, and each symbol connects to a random subset of $k = 100$ of them. Inflections are encoded using $p = 11$ pulses in a burst with $b = 60$ time slots. Messages may have up to $m_s = 5$ symbols, each with up to $m_i = 2$ inflections. The maximum permitted size of a symbol encoding is $s_+ = 9$, and thresholds are $d_i = 10$ for inflection detection, $d_s = 8$ for symbol detection, and $d_c = 7$ for symbol consensus. Each measured value comes from 10 trials of 20,000 rounds each, evaluated for vocabulary size and percent perfect message transmission once every 100 rounds. Behavior at the end of a trial is taken to be the behavior during the final 2,000 rounds.

A small constant factor of coder oversupply in the listener is sufficient for efficient

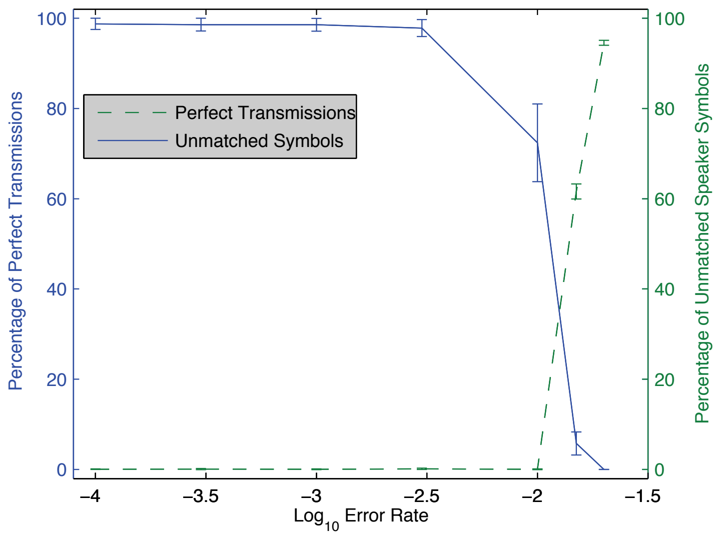


Figure 7-12: Noise has minimal impact until around 1% noise, when noise is high enough to disrupt rendezvous during self-organization. This graph shows the final behavior for 20,000 rounds of self-organization.

self-organization: Figure 7-9 shows the final symbol vocabulary size for oversupply ranging from none ($o_s = 1.0$) to $o_s = 1.5$; inflections are always fully matched, even at $o_s = 1.0$. Perturbations during morphogenesis mainly affect populations, so the oversupply also provides resilience against all small perturbations besides crossbar defects.

When the system is far from capacity, self-organization is fast. Figure 7-10 shows that vocabulary growth is approximately linear, slowing as symbols that failed to connect on their first try begin to retry (inflections connect almost immediately). When the channel is smaller, symbols interfere with one another more: past a critical threshold, self-organization runs more slowly and gradually collapses (Figure 7-11).

Finally, a small amount of noise on the communication channel does not affect either self-organization or normal message transmission. Figure 7-12 shows that noise has minimal impact at low levels, then causes a dramatic collapse in performance at around 1% noise, where the noise bits begin to inhibit symbol rendezvous during self-organization.

7.1.3 Bidirectional Link

Once we have built a unidirectional link, a bidirectional link is relatively straightforward to create. We simply take two unidirectional links—one in each direction—allow them to self-organize, then pair mature symbols and inflections together with a dis-

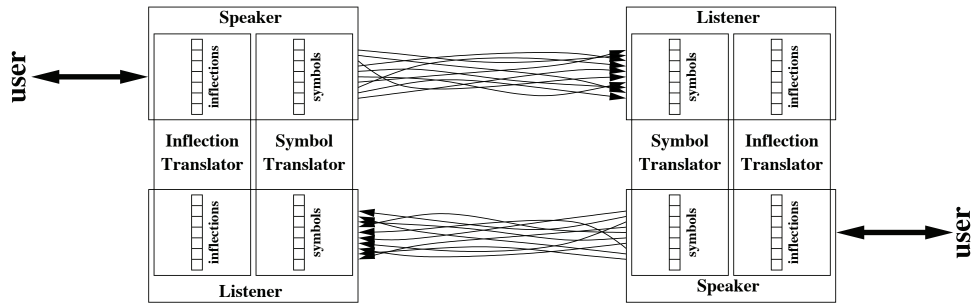


Figure 7-13: A bidirectional link connects together two unidirectional links with self-organizing translators that pair up speaker and listener coders as they mature in the unidirectional links. Because there are two links, messages can move in both directions simultaneously without interference. The user interacts only with speaker symbols and inflections: incoming messages are translated on their way to the user.

tributed map used for translation (Figure 7-13). The only trick is to ensure that both sides pair up equivalent elements.

The user of a bidirectional link then interacts only with the speaker version of each symbol or inflection. These (once paired) are made available for allocation and deallocation exactly as mature symbols and inflections are made available by the unidirectional link. To send a message, the bidirectional link passes the interaction straight through to its speaker link. To receive a message, on the other hand, the link simply remaps each signal through the distributed map connecting the speaker and listener signals together.

Appendix C.3 contains a full specification of this device.

Translating Symbols and Inflections Like the unidirectional link, the bidirectional link self-organizes during times when the channel is otherwise idle. This means that a bidirectional link has three distinct modes of operation: ordinary use, unidirectional self-organization, and bidirectional self-organization. One parameter of the bidirectional link is thus the ratio r_{ub} of self-organization time devoted to the unidirectional link versus the bidirectional link.

The mechanism I use to align pairs of symbols or inflections in a bidirectional link is relatively simple (Figure 7-14). Each side of the link allocates one symbol and inflection at a time from its speaker-side unidirectional link: these will serve as the target for pairing. This target is used in a "call and response" pattern: the device sends a message containing just that symbol/inflection pair, and waits for a response from its partner.

Upon receiving a message, the partner tries to translate the symbol and inflection to its speaker-side link. If an element is not already in its translation map, then it

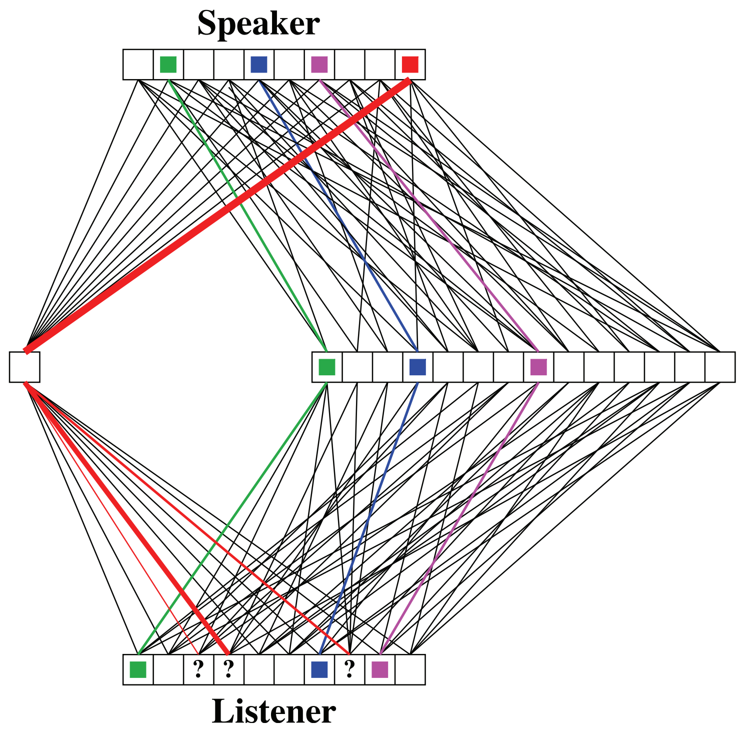


Figure 7-14: A bidirectional link translates between speaker and listener elements using a distributed map (right-hand bundle of connections). The map self-organizes by targeting one speaker element at a time (red square). The current target is sent to the partner, which allocates a new element to pair with it and sends the new element back on the other link. Tracking responses (left-hand bundle of connections) reveals the partner's choice quickly, and the appropriate translation can be added to the distributed map.

adds it, linking it to a newly allocated element from the speaker-side link. Finally, it sends the translated symbol/inflection pair back to the originator.

The originating side tracks the responses it gets, using a link that connects every speaker and listener element to a single rendezvous point. Each element that appears in a response is a proposed match, and the proposal is decided upon using the codetector from Chapter 5: responses containing the element are positive examples and responses that do not contain it are negative examples.

When one element's proposal is accepted and all others are rejected, the pair is added to the translation map and a new element is allocated to be the target for pairing. If something goes wrong and all proposals are rejected, the element is deallocated and a new element is allocated to be the target.

Rather than assign fixed roles, I allow both sides of the link to initiate and to respond. If each element tracks whether its pairing was initiated locally or in response to a signal from the partner, then we can ensure that the two sides (almost) never

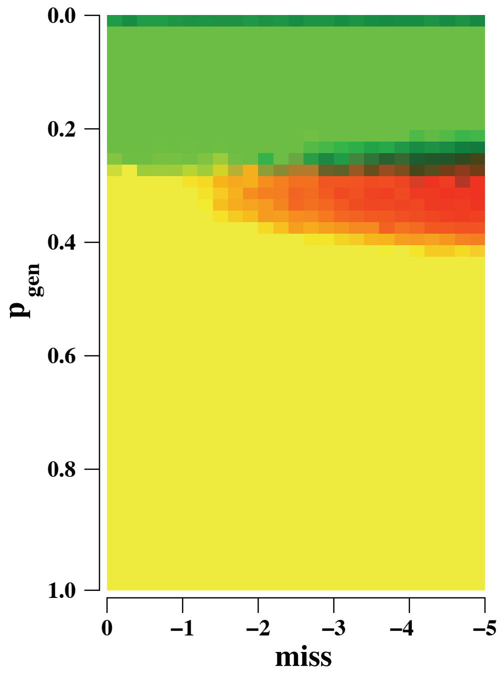


Figure 7-15: Simulation of a bidirectional link with idealized unidirectional links shows that a low p_{gen} leads to successful self-organization, and that the $miss$ penalty is largely irrelevant. Red indicates significant transmission errors, green indicates fast acquisition of vocabulary, yellow indicates fast acquisition of faulty vocabulary, and black indicates failure to communicate.

allocate the same element for different purposes by having them only allocate elements whose pairing was initiated locally.[9]

Having both sides play both roles complicates things, because each side needs to distinguish between a response to its own message and a new message. My solution is to have each side only play the role of initiator while it waits for a response for the time of one round trip max_{rtt}; the rest of the time, it plays the role of responder. If each side generates new messages with only a low probability p_{gen} in each unit of time when not waiting for a response, then most of the time precisely one side will be acting as the initiator, and repeated collisions leading to confusion are unlikely.

Behavior of Bidirectional Link I use experiments in simulation to determine reasonable values for the the generation probability p_{gen} and the $miss$ penalty for negative evidence, and the distribution of self-organization time between the unidirectional links and the symbol pairing process for the bidirectional link. The mechanism is not highly sensitive to parameter values and there is a broad range in which

[9]The exception is a rare double coincidence: if both simultaneously choose a pairing that was created when the two sides initiated a new pairing simultaneously.

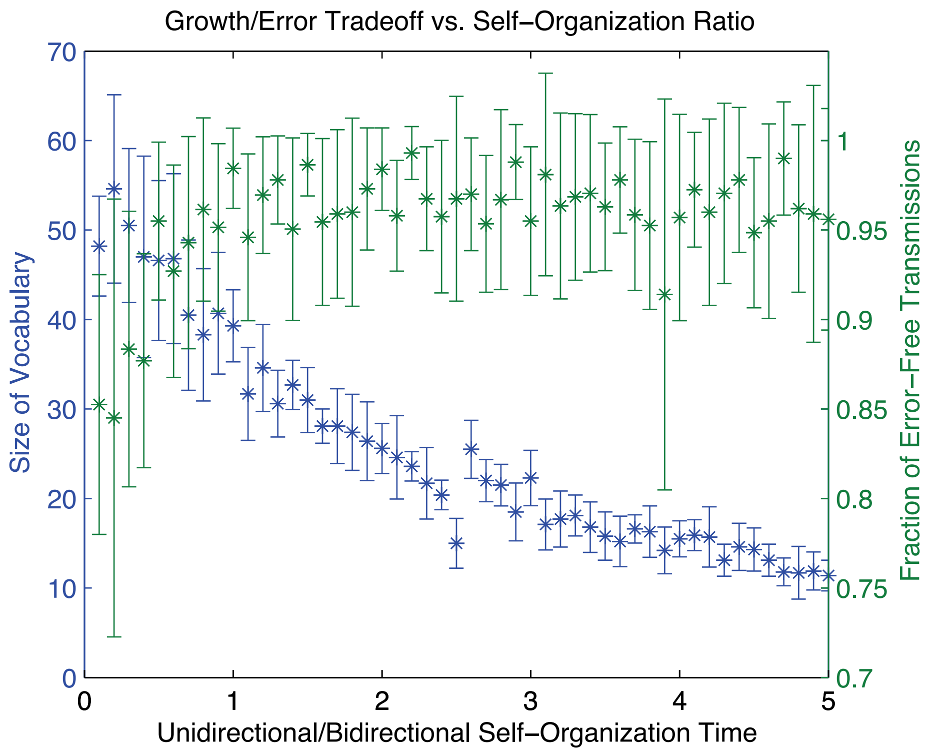


Figure 7-16: More self-organization time needs to be devoted to the unidirectional links than to the bidirectional link: the lower the ratio, the faster the bidirectional link vocabulary grows, but below around 2:1 unidirectional to bidirectional ratio, the error begins to increase.

self-organization quickly produces a pool of reliable symbols and inflections.

Figure 7-15 shows the effect of p_{gen} and $miss$ penalty on a bidirectional link using idealized unidirectional links that contain an unlimited number of elements organized in advance, using the parameters $max_{rtt} = 3$, $accept = 5$, and $reject = -5$ and ranging p_{gen} from 0.01 to 0.99 in steps of 0.02 and $miss$ from -5 to -1/5 in steps of 1/5. Red indicates significant transmission errors, green indicates fast acquisition of vocabulary, yellow indicates fast acquisition of faulty vocabulary, and black indicates failure to communicate. As can be seen, a p_{gen} of approximately 0.2 or lower leads to successful self-organization and $miss$ is largely irrelevant.

To see how the distribution of self-organization time affects the behavior of the system, I interleave chunks of unidirectional self-organization with chunks of bidirectional self-organization, setting the minimum chunk size to 100 time units and letting the ratio of unidirectional to bidirectional r_{ub} range from 0.1 to 5.0. Thus, for example, a ratio of $r_{ub} = 2.0$ means the system trains unidirectional for 200 time units, then bidirectional for 100, and then goes back to unidirectional.

Figure 7-16 shows the rate and quality of self-organization for various distributions of training time running for a total of 5000 time units. The unidirectional links use the same parameters as in their characterization above, while the bidirectional link uses $miss = -2$, $max_{rtt} = 3$ and $p_{gen} = 0.1$. We see that the lower the ratio, the faster the bidirectional link vocabulary grows, but below around 2:1 unidirectional to bidirectional ratio, the error begins to increase.

7.1.4 Do Brains Separate Form and Meaning?

Interestingly, connections between regions of the brain have a similar bidirectional structure: there is plentiful neural fiber running in both directions even where one might expect information to flow mostly in one direction[39]. It is not unreasonable to think that the connections within a brain might use a similar self-organization process.

This is not at all necessary to my communication bootstrapping hypothesis. My need to separate form and meaning may simply be due to the limitations of my engineering ability, rather than anything inherent about communication bootstrapping. If connections between regions of a brain do self-organize abstract signals for communication, however, that surprising congruence would be at least circumstantial evidence in favor of my hypothesis.

What evidence could there be as to whether a self-organization process of this sort is taking place in the brain? Although most of the details might vary widely, the two types of self-organization signals must always be separated both from one another and from normal use of the signals, because they are not compatible with one another or with normal communication.

Thus, if self-organization of this sort is taking place on a bidirectional connection between two regions of brain, we would expect to see three distinct modes of behavior:

- **High activity, mix of repetitive and fast-changing patterns, moderate correlations between flow directions:** this behavior would indicate normal use, where the signals are conveying real information. The speed at which patterns change should thus be correlated with sensory input and perhaps also introspection.

- **Low activity, fast changing and apparently random patterns, no correlation between flow directions:** this behavior would indicate independent self-organization of the two unidirectional links.

- **Very low activity, fast changing and apparently random patterns, strong correlations between flow directions:** this behavior would indicate pairing of unidirectional elements into bidirectional elements.

The latter two modes might be hard to detect and distinguish, since there would be so little activity.

The pool of signals and inflections needs to be constantly replenished, particularly if deallocated symbols are discarded. As a result, one would expect to see all three modes appear at least once every few days. One prime candidate for when self-organization would run is, of course, during sleep. The brain-wide selection of various modes of sleep could separate the three modes safely. If the training bursts are brief, however, they might take place scattered throughout the day.

Even if these predictions are all borne out experimentally, it does not mean that the brain is self-organizing connections between regions. It would, however, be a surprising similarity worth more investigation.

7.2 Learning From Message Sequences

Sending messages back and forth between specialists is not enough: once a message arrives, a specialist needs to be able to relate its partner's message to its own model. In this section, we will discuss how a specialist can discover relations between its model (proxied by its use of symbols and inflections) and its partner's use of symbols and inflections.

This is an unsupervised learning problem: there no trustworthy teacher labelling the data with categories or providing feedback on whether the relations a specialist discovers are correct or even sane. All that is available are two sequences of messages, one being sent, the other received.

What sort of relations should we look for between these two sequences of messages? I will choose to look for relations that allow the contents of one sequence to predict the future contents of the other.[10] As always, there may be other relations between sequences that would be interesting to learn, but prediction has been a good place to start and is directly related to the goal of predicting observations.

Since the number of possible relations to consider goes up nearly exponentially with the number of elements in a relation, I will consider only relations between pairs of like elements: one incoming symbol to one outgoing symbol or one incoming

[10]It may seem odd that I am only looking for relationships between sequences, rather than looking for ways that a sequence can predict its own future behavior. The reason is that the messages are an expression of a model's working set, and that I am leaving the discovery of relations within a model to the specialized reasoning hardware associated with each specialist. There is, however, no reason we cannot apply the mechanisms we create here to such internal learning, if they turn out to be suitable to the task.

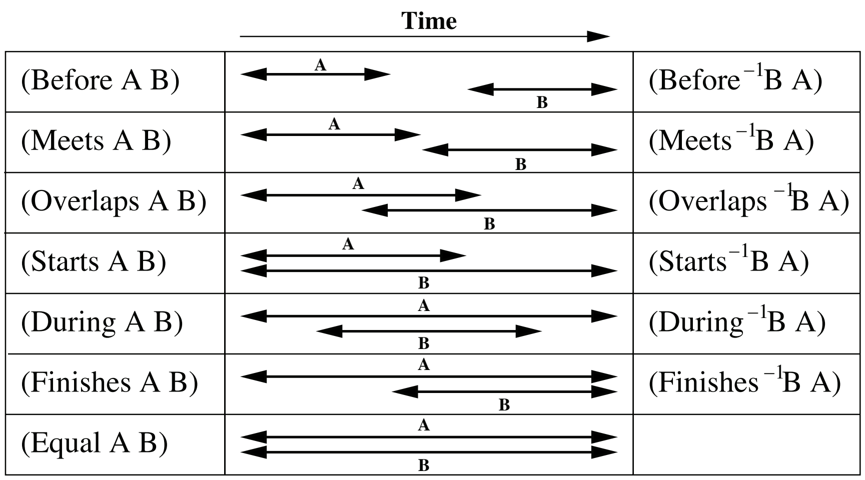


Figure 7-17: Allen's 13 time relations[1] compare intervals by comparing their start and end times.

inflection to one outgoing inflection. For example, the vision specialist should learn that when its partner uses the symbol for "engine noise," it will soon use the symbol for "car." In order to learn a relation between several elements, a specialist must create symbols that name groups of elements, reducing it down to a relation between two abstract elements. For example, the vision specialist can learn that the symbol for the "cuckoo" audible walk signal happens after a pedestrian contacts a pole if it invents and uses a symbol for the situation "pedestrian contacts a pole."

I will build up a mechanism for identifying predictive relations between pairs in three stages. First, we will see how Allen's time interval relations[1] can be used to combine sequences of messages into examples that connect together events separated in time. Second, we will see how these examples can be interpreted as evidence for or against various predictive relations. Finally, we compile this into an incremental example detector that learns relations aggressively.

7.2.1 Time Interval Relations

We can turn a sequence of messages into a collection of intervals by tracking when each element of interest is present in a message. An interval starts when an element appears, continues as long as it is present, and ends when it vanishes. We can then compare intervals by comparing the times at which they start and end, using the 13 time relations identified by Allen[1] and shown in Figure 7-17.

Allen's time relations presume continuous time values, while our sequences of messages are discrete. We can handle this discrepancy by interpreting the messages as samples of the specialist's working set. This means that we do not know precisely

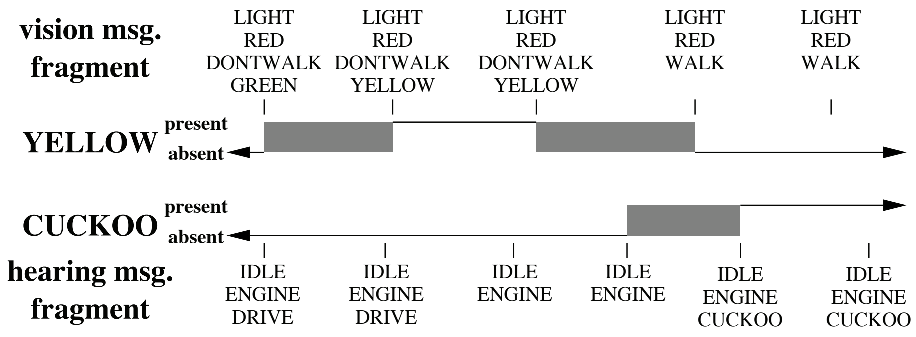


Figure 7-18: Messages are samples of a specialist's working set, so intervals start and end at an unknown point between messages. When the ranges of two end-points overlap, they are considered to happen at the same time. Thus, the sequence shown above has the time relation **(MEETS YELLOW CUCKOO)**.

when an interval starts or ends, only that it happens between two samples. When comparing intervals, then, we will consider two times to be equal if their ranges overlap at all. This also allows us to use specialists whose execution is not tightly synchronized.

Figure 7-18 shows an example comparison of two intervals, one for the **YELLOW** symbol being sent from vision, and one for the **CUCKOO** symbol being sent from hearing. The **YELLOW** interval starts before **CUCKOO**, but ends when **CUCKOO** starts, so the two intervals have the time relation **(MEETS YELLOW CUCKOO)**.

Using this interval representation allows us to avoid scaling problems, and to bring events together in time: it does not matter how many messages say **CUCKOO** and how many say **YELLOW**, only their relative order.

The interval representation also suggests a scale-invariant means of segmenting the sequences into examples: each **BEFORE** relation (in either direction) marks the start of a new example. The reason to choose the **BEFORE** relations for this purpose is that these are the only time interval relations where there is a gap where neither feature is present, and that the **BEFORE** relations thus offer the greatest chance that two subsequent examples will be independent of one another. Finally, when there is more than one time relation in an example, we need to reduce the set down to a single representative time relation. I will do this by choosing the first time relation, which will mean that learning can take place as soon as any piece of evidence is available.

7.2.2 Predictive Relations

Allen's time interval relations are not quite what we need in order to make predictions. Instead, I will hypothesize explicitly predictive relations, and categorize time relations into evidence for, against, or ambiguous regarding each predictive relation.

Take the example from the beginning of the chapter, "seeing a yellow light is followed by hearing a cuckoo." I capture this with a **SEQUENCE** relation, which says that **YELLOW** predicts **CUCKOO** will appear, and **CUCKOO** predicts **YELLOW** will disappear.

Since we need to assume that most elements are not in the working set most of the time (else our messages would be too large), it is only reasonable to make predictions based on the presence of an element. Absence of an element is simply too weak a signal for general use (the absence of elephants should not affect ordinary thinking). Thus, for a relationship between elements A and B, the presence of A has five possible predictive relationships to B:

- A predicts B will appear.
- A predicts B may appear.
- A does not anything about predict B.
- A predicts B may disappear.
- A predicts B will disappear.

The distinction between "will" and "may" statements traces back to the definition of good agreement in Section 6.3. A "will" relation promises both to help predict changes and that its predictions will be satisfied, while the weaker "may" relation promises only that it will help predict changes.

With two elements, there are 25 possible combinations. Of these, I will look for eleven: the six shown in Figure 7-19 and the inverses of all except **EQUAL**, which is symmetric. For any pair of elements, I will search for all eleven relations independently, and may find that several hold simultaneously. For example, **SEQUENCE** will usually be accompanied by **CAUSE**, **ENABLE**, and **DISABLE**$^{-1}$.

Why not search for all twenty-five? First off, I will want clear evidence for or against any relation, and the sparseness assumption (most elements are not in the working set) means that it is not possible to gather clear evidence for "may disappear." A "may disappear" relationship is like elephant repellent: you know it is working because there are no elephants nearby. Eliminating "may disappear" eliminates nine of the fourteen remaining possible relationships.

Name	**Predictions**	**Time Relations**
		`BMOSDFEfdsomb`
EQUAL	**A,A**	`0-111111111-0`
SUBCLASS	**A,a**	`----111--1--0`
SEQUENCE	**A,D**	`011-------000`
CAUSE	**A,-**	`011--1-0--00-`
ENABLE	**a,-**	`-11111100100-`
DISABLE	**-,D**	`-11-0----000-`

Figure 7-19: I will search for eleven predictive relations between two elements: the six shown above and the inverses of all but **EQUAL**, which is symmetric. Predictions are shown as *A predicts B,B predicts A*, with **A**=“will appear”, **a**=“may appear”, **D**=“will disappear”, **-**=“no prediction”. Time relation evidence is shown in 13 slots, one for each relation, identified by its first letter with inverses in lower case. The evidence for relations is shown as **1**=positive evidence, **0**=negative evidence, **-** is ambiguous evidence.

Of the remaining five, one is the symmetric “no prediction” relationship, which is the same as no relationship, and therefore can be discarded. The others are the four combinations of “may appear” and “will disappear.” These can be learned as pairings of **ENABLE** and **DISABLE**, and I did not see a way in which learning the combinations would be different than learning both individual relations, unlike the case for **SEQUENCE**, **SUBCLASS**, and **EQUAL**.

Finally, we need to determine, for each of the predictive relations, which time relations are clear evidence for or against and which are ambiguous. I will add one further constraint: since the likelihood of two interval end-points being equal depends on sampling rate, and since I want relation detection to be scale-independent, it is important that no predictive relation depend critically on a time relation that contains an equality, such as **MEETS** or **FINISHES**. As such, I will require that each equality-containing time relation provide the same type of evidence as one of the time relations it can become when the timing of the interval end-points is perturbed slightly. Thus, for example, **FINISHES** must give the same evidence as either **DURING** or **OVERLAPS**.

Figure 7-19 shows how I have chosen to interpret time relations as evidence for each predictive relation. There may be other reasonable ways of interpreting the time relations as evidence, but the ones shown here have served me well so far.

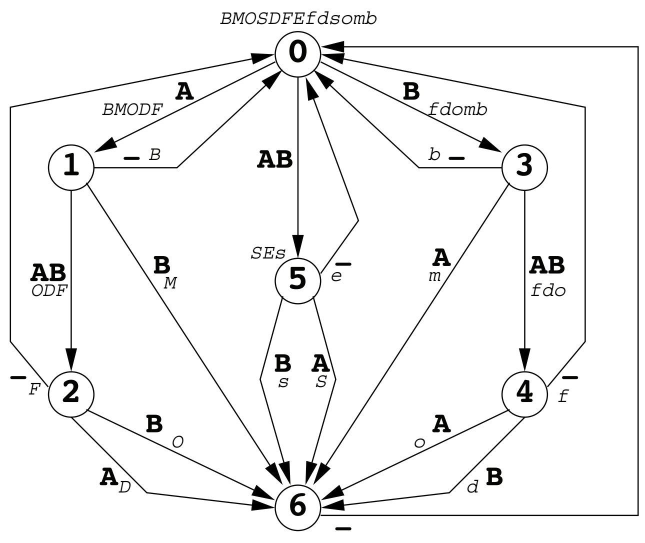


Figure 7-20: Incremental Interval-based Example Segmentation (IIES) uses a finite state machine for incremental detection of a time relation example between element A and element B. Bold text on edges show the combination of currently present elements that triggers a transition; italic text shows which time relations might hold following that transition.

7.2.3 Incremental Interval-Based Example Segmentation

Our last step is to take these lists of positive and negative evidence and transform them into a finite state machine for incrementally detecting positive and negative examples of each type of predictive relation. I call this mechanism *Incremental Interval-based Example Segmentation* or *IIES*.

Here, we will be aggressive in detecting examples. Rather than waiting for a relation to resolve completely, we will produce an example as soon as it is clear what the example will be. For example, a positive example for an **EQUAL** relation can be detected as soon as both elements appear at the same time. This means that we do not need to wait for an element to go away in order to learn about its relations, which will be important for learning about long-lived phenomena when a shifting focus of attention is introduced in the next chapter.

Figure 7-20 shows a finite state machine for incremental extraction of examples from time relations (omitting self-transitions). State zero is the beginning of an example, which might start with any of the 13 time relations. As the two elements appear and disappear, the range of possible relations narrows, until only one remains and the system ends up either in state zero (waiting for a new example) or in state six (ignoring further interval relations while waiting for a moment when both elements are absent).

For each predictive relation, we compile this general state machine into a relation-specific incremental example detector that reports an example on the first unambiguous transition away from state zero. For each pair of elements, we run all eleven machines in parallel, each feeding its output into a codetector (Chapter 5) that decides whether the relation holds. We now have a device, IIES, that turns two sequences of messages into predictive relations between a pair of message elements.

7.2.4 Experimental Verification

I have used input from the four-way intersection scenario to verify that this mechanism behaves as expected. For this test, I take each set of sensory input, and flatten it into a list of whether each feature or relation is present *anywhere* in the input. Thus, for example, there is a slot for the **CAR** feature that holds **TRUE** if there is at least one car visible and **FALSE** if there are no cars visible. I then search for relations between all pairs of inputs, using codetector parameters $miss = -2$, $accept = 10$, $reject = -10$, $rail+ = 50$, and $rail- = -50$.

To test this system, I recorded four flat-sense movies from the simulator, sampling once every 0.5 simulated seconds for 5000 seconds (approximately 80 simulated minutes). The four movies are:

1. Starting at noon (moderate traffic)

2. Starting at midnight (low traffic)

3. Starting at 8am (morning car rush hour)

4. Starting at 3pm (afternoon school rush hour)

If IIES is working correctly, we should expect to see the following:

- A small number of relations like **(EQUAL CUCKOO WALKLITE)** that capture true relationships between the senses.

- Many relations due to the coincidences of the senses observing for a long time from a fixed position and orientation.

- Spurious **EQUAL**, **ENABLE** and **SUBCLASS** relations between things like pedestrians and cars that are usually present at least somewhere in the scene.

- No relations involving features that are never absent from the scene, like the visual feature **OFFICE** and relation **ABOVE**.

- No relations involving features that only appear a few times, like the **CRASH** sound.

- No nonsensical relations like **(CAUSE IDLE YELLOWLIGHT)**.

Note that in this impoverished representation, much of the interesting behavior in the scenario is simply not possible to learn. For example, the light cycle cannot even be observed because there is no way to tell which light object the always-present **BRIGHT** feature is associated with (the representation will be enriched in the next chapter).

Are Reasonable Relations Learned? First, let us look at what relations are learned from the noon movie. The full summary of the movie and the relations learned from it (as well as the ones for other six that will be discussed) is listed in Appendix B.1.

Not all features appear: 59 of 64 vision features and 24 of 28 hearing features are present, meaning there are 1416 possible pairs and 15576 possible relations. All told, a total of 156 relations are actually learned between 91 pairs of features. A full 34 of the visual features are continually present (e.g. **SIDEWALK**, **ABOVE**, and **GREEN**) and 7 others appear less than ten times (e.g. **AMBULANCE**): none of these should appear in relations, and none of them do. Likewise, the 8 hearing features with less than ten intervals (rare ones like **YELL** and too-common ones like **STEPS**) do not appear in relations.

Now let us look at the 156 relations that are learned. There are 78 **ENABLE** relations, 47 **SUBCLASS** relations, 18 **EQUAL** relations, 9 **DISABLE** relations, 2 **CAUSE** and 2 **SEQUENCE** relations. These relations capture many interesting properties of the scenario, including:

- **CUCKOO** and **WALKLITE** are the same thing.

- **DONTWALK** sometimes leads to **CUCKOO**, then disappears.

- A moderately loud sound (around **70** decibels) is always followed by the appearance of a **CAR**, then disappears. It is sometimes followed by the **CAR** subclasses **TRUCK**, **VAN**, or **SUV**, or by backwards motion (**B**) as nearby cars cross the intersection.

- A **CAR** is always moderately loud (**70** decibels).

- The **WALKLITE** and upward motion (**U**, from pedestrian crossing) only happen when engines **IDLE**, which in turn happens only when there is a **CAR**.

- Seeing a **CAR** of any sort (e.g. **SEDAN**, **TRUCK**) or motion across the intersection (**L**, **R**, **F**, **B**) happens only when engines **DRIVE**.
- Sounds directly in front come from a **CAR** of any sort (e.g. **SEDAN**, **VAN**) or the **WALKLITE**.

There are only six apparently strange relations learned: **(DISABLE IDLE L)**, **(EQUAL FR CAR)**, **(SEQUENCE FR CAR)**, **(CAUSE FR CAR)**, **(DISABLE CAR FR)**, and **(DISABLE IDLE BLUE)**. Each one, however, has a likely situation-dependent explanation. Strongly leftward motion (**L**) appears to come mainly from cars, so when a car's engine is **IDLE** waiting for the light, the cars that come up behind it stop their leftward motion. Forward-right sound **FR** comes mainly from cars waiting at the light, so when it is heard, after the light changes the cars will go through, appearing to vision and moving their sound to a different octant. The last appears to be a pure fluke of the scenario: the **BLUE** feature often comes from pedestrian pants, so when cars are waiting **IDLE** for a red light, pedestrians can cross and take their blue pants away. None of these explanations are certain, but they are consistent with detailed inspection of short segments of input.

We can thus see that the relations learned from the noon movie are in line with our predictions. The relations learned involve many time spans, some short like cars stopping for a light, some moderate like walk lights, and some fairly long like all the times when some car is driving nearby. The relations also involve both common features like hearing a car idle, which happens in 72.5% of the samples, and rare ones like hearing the cuckoo sound, which happens in only 5.5% of the samples.

What Happens When There is More or Less Activity? To confirm that IIES is not sensitive to the distribution of sensory events, I compare results of learning from all four movies, where the relative activity spans a large range: the arrival rate of cars and pedestrians during their respective rush hours is 20 times higher than in the middle of the night.

My prediction was that the midnight movie might break up some of the too-common features so that we could learn about things like the sound of engines, while the rush-hour movies might make rare events more common and allow us to learn about children and emergency vehicles. By and large, however, I expected the relations learned to be approximately the same. Let us see how these predictions were borne out.

For the midnight movie, only 97 relations were learned between 55 pairs of features. Many of the significant relations learned were also learned in the noon scenario, with several notable exceptions. First, with such sparse activity in the intersection, not one pedestrian actually used the walk light, so all 13 relations involving **CUCKOO** or **WALKLITE** were missing. Less activity also meant that more features appeared too rarely for relations to be learned, including **ADULT** and **SUV**.

As predicted, however, the sparseness of activity meant that previously intermingled features could be separated enough to learn from. Among the interesting new relations that were learned:

- A **CAR** never appears unless a moderate sound (**60** decibels) or **DRIVE** is heard. These sounds often vanish after the car appears.

- A **CAR** and **ENGINE** sound always go together, and the sound comes first. Sometimes **ENGINE** leads to a **SEDAN** or **VAN**.

- Moderately quiet sounds (**50** decibels) transition to forward motion (**F**), which in turn transitions to moderately loud sounds (**70** decibels). Backward motion (**B**) transitions to moderately quiet sounds (**50** decibels).

- Forward motion (**F**) leads to sounds in front (**F**) and right (**R**), leftward motion (**L**) leads to front-left sounds (**FL**), and rightward motion (**R**) leads to sounds in the front-right (**FR**) and right (**R**).

All of the behaviors leading to these observations were in the noon movie, but with superimposed events confused by the flat representation.

Besides these, a number of the likely spurious or situation-dependent **EQUAL**, **ENABLE**, and **SUBCLASS** relations are different, likely simply due to the decreased number of examples and to the different random events in the two movies.

For the rush-hour movies, the 8am rush-hour movie learns 118 relations on 74 pairs, while the 3pm movie learns 129 relations on 83 pairs. Again, most of the substantive relations learned were also learned in the noon movie. Here, however, the confusion induced by the flat representation starts to obliterate relations that were easily learned before, even as previously rare events are brought up to learnability.

For example, much less is learned about **CAR** in the 8am movie than in the noon movie, but many of the noon movie relations are learned for particular types of car like **SUV** or **VAN**. In the 3pm movie, the increased density of people leads to the discovery that hearing **TALK** or **VOICE** sometimes leads to a **WALKLITE**, as well as a number of other less interesting associations of visual features with conversations.

Again, the distribution of apparently low-content **EQUAL**, **ENABLE**, and **SUBCLASS** is different both between each rush hour movie and noon and between the two rush hour movies.

Finally, a few strange **DISAPPEAR** relations appear in each rush-hour movie—two in the 8am movie and three in the 3pm movie. The three in the afternoon movie were situational **ENABLE** relations in the other movies, and the change to **DISABLE** relations appears to be a result of the change in car versus pedestrian ratios. The two in the morning movie, claiming that **TRUCK** makes both **TALK** and **VOICE** disappear, appears to be a situational relation resulting from a coincidence in the distribution of trucks and the stuttering nature of conversations.

As can be seen, the relation learning adjusts as predicted, showing that although IIES is affected by the distribution of external events, the effect is as predicted and not strong.

Does Sampling Matter? To confirm that IIES does not depend strongly on sampling rate, I compare the results of learning from every frame of the noon movie to the results of learning from every second, third, or fourth sample.

As the resolution decreases, two things happen to the data. First, extremely brief events may disappear entirely. Since most events in the scenario take several seconds to complete, the difference between sampling every 0.5 seconds and every 2.0 seconds should not have a large effect here. Second, events that were previously separated are brought together. This can create new relations, connecting two related features that were previously separated by a short gap, or it can blur experiences, connecting previously separated intervals of a common feature.

Considering both these effects, I predict that there will be little change in substantive relations, but that the "coincidental" relations will be much affected. Let us see how these predictions are borne out.

To begin with, the three under-sampled movies produce approximately the same number of relations as the original: 164 relations on 93 pairs for every second sample, 162 relations on 90 pairs for every third sample, and 176 relations on 94 pairs for every fourth sample.

Nearly all of the relations in the original movie are learned in each of the other three under-sampled movies. Approximately 10% of the low-content **EQUAL**, **SUBCLASS**, and **ENABLE** relations differ. Regarding substantive relations:

- All of the under-sampled movies lost the stronger **CAUSE** and **SEQUENCE** relations between moderately loud sounds (**70**) and **CAR**. The slowest movie also lost two other relations with **70**. These differences are likely due to the brevity of the loud sounds as the cars transit the intersection.

- All of the under-sampled movies lost most or all of the six odd situation-dependent relations, **(DISABLE IDLE L)**, **(EQUAL FR CAR)**, **(SE-**

QUENCE FR CAR), **(CAUSE FR CAR)**, **(DISABLE CAR FR)**, and **(DISABLE IDLE BLUE)**.

- The slowest movie added **CAUSE** and **SEQUENCE** to the relations between **DONTWALK** and **CUCKOO**, likely because the sampling could often entirely skip the first blink of the **DONTWALK** light.
- The two slower movies added a **DISABLE** relation from **IDLE** to forward motion (**F**). I suspect this has a similar origin to the odd **(DISABLE IDLE L)**, which disappeared.

The results of learning from all four movies are quite similar, in accordance with our predictions, showing that although IIES is affected by the sampling rate, the effect is as predicted and not strong.

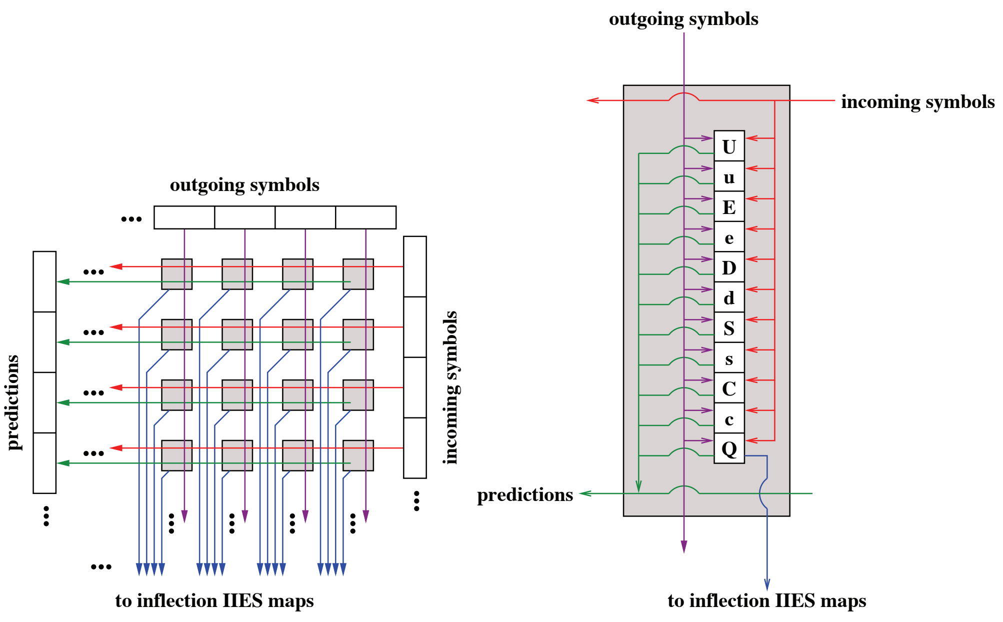
Summary We have seen that the IIES mechanism can find interesting predictive relations between pairs of features generated by our scenario. We will hold off on an examination of learning speed and predictive quality until the next section, when we place IIES in the context where it will actually be used, learning from messages between specialists.

7.3 Signal Map

We now have all the parts we need to construct a signal map following the design shown at the beginning of the chapter in Figure 7-1.

Recall that the purpose of the signal map is to translate between signals and model elements. The bidirectional map we developed in Section 7.1 translates between signals and symbolic messages. That will be all the signal map needs for outgoing messages, since I made the assumption in Section 6.4.1 that the model is a semantic network and messages express the working set. For incoming messages, however, there must be a means of relating the elements of a partner's message to a specialist's own model. This is as far as the signal map will go: deciding how to incorporate messages into the model is beyond the scope of this investigation.

Conceptually, the design is simple. The user sends and receives messages through the bidirectional link. As they flow between the user and the link, incoming and outgoing messages also flow to *relation maps*, which compare the two streams to find relations between pairs of symbols or pairs of inflections. Once the relation maps begin finding relations, they begin producing predictions for each message, incoming or outgoing, of how the complementary sequence of messages will change in the future.



(a) Symbol Relation & IIES Map (b) Junction Detail

Figure 7-21: The symbol relation map (a) compares symbols in incoming (red) and outgoing messages (purple). At each junction of the crossbar (b) these are fed to a paired IIES device and codetector for each of the 11 predictive relations. When a relation is accepted, it outputs predictions (green). The **EQUAL** relation, when accepted, connects to the inflection relation map and relays signals flowing from either incoming or outgoing symbols (blue).

7.3.1 Relation Maps

The implementation is somewhat more more complicated due to the difficulty of learning useful relations about inflections. The problem is that there are many fewer inflections than relations and that we may expect the average inflection to be used much more frequently than the average symbol. Thus, a specialist cannot learn useful relations between inflections merely by looking at what inflections are present in a message. We must consider inflections in the context of the symbols to which they are attached.

Fortunately, I intend to learn a simpler set of relations for inflections than for symbols. I am using inflections to encode the relations between model elements, and in the systems I build, these will generally be equivalent or untranslatable.

What this means is that I am most interested in quickly identifying a set of equality relationships between inflections, so that the universal relations can be translated from specialist to specialist. This can be done conservatively by comparing only inflections on symbols that already have an **EQUAL** relation between them. Arranging hardware to do this quickly is the main source of complexity in the design of the signal

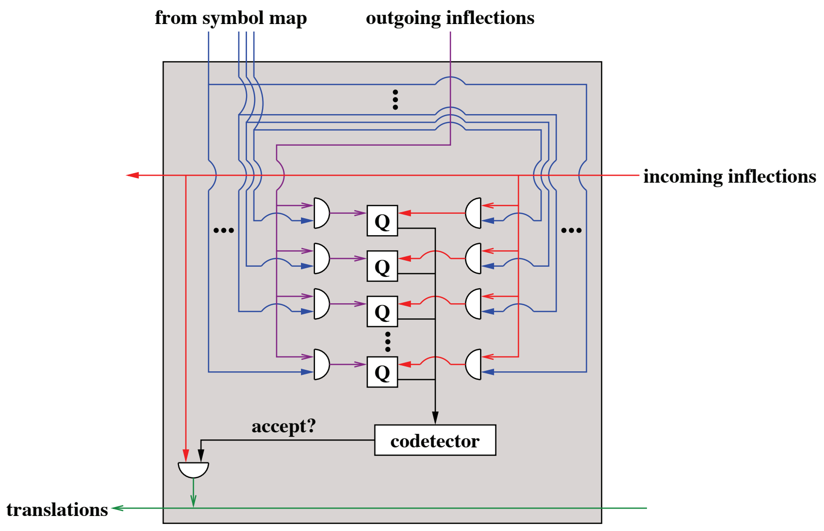


Figure 7-22: Each junction in the inflection relation map has several IIES devices that detect **EQUAL** relations and feed the same codetector.

map.

Figure 7-21 shows the map used for finding relations between symbols. The map is built around a crossbar connecting every incoming symbol to every outgoing symbol (the inputs to the map are connected to the set of coders by distributed maps serving as busses). Every symbol appears on both sides of the map, because a specialist may come to use a symbol differently than its partner does.

At each junction of the crossbar (Figure 7-21(b)), the incoming and outgoing signals are fed to a paired IIES device and codetector for each of the eleven predictive relations. If the relation is accepted by the codetector, then this device also outputs the appropriate prediction for each stimulus it receives.

When a junction's **EQUAL** relation is accepted, it also participates in inflection learning. The inflection relation map has not one, but many layers of IIES devices (all of them looking only for **EQUAL** relations): one layer serves one pair of matched symbols, so if there are k layers, then inflections may be learned on up to k pairs of **EQUAL** symbols at a time.

During any interval where the junction's symbols are appearing on both incoming and outgoing messages, the **EQUAL** relation connects with an IIES layer in the inflection relation map by means of a distributed map. When too many try to connect, using a distributed map for the connection results in the excess being persistently dropped. Upon connection, all of the IIES devices in the layer are reset to state zero, avoiding spurious example detection. When one of the symbols disappears, it disconnects after the IIES later makes one transition with no inflections on the vanished symbol.[11]

Pulses arriving at the junction are then routed onward to the inflection map, where they are intersected with the inflection pulses such that the IIES devices in each layer only receive pulses representing the inflection being carried on the pair of symbols connected to that layer (Figure 7-22). All of the layers at a junction feed the same codetector, and when that codetector accepts, incoming inflection pulses from the partner are translated to the specialist's own equivalents. Since the multiple layers means there are likely to be more example than for symbols, the codetectors' thresholds and rails are set to larger magnitudes than those of the codetectors deciding on symbol relations.

7.3.2 What Sorts of Signal Maps Could a Brain Afford?

If a mechanism similar to this signal map were used in the human brain, how many signals and inflections could specialists use to communicate? A coarse analysis of this design according to the developmental cost metric from Section 4.3 gives us a range of possibilities.

The numbers in this section should be thought of as collection of a Fermi estimates: they may be off by a few orders of magnitude, but will give us an idea of what the important factors are and serve as a sanity check that the design is not immediately implausible.

First off, notice that the parallel construction means that execution time is related to the number of inflections but not the number of symbols. If they are encoded as pulses, then the number of inflections depends on the ability to distinguish pulses within a signal map. With human reaction times on the order of a tenth of a second and neurons capable of acting on the order of a millisecond, we are looking at the ability to send messages containing somewhere between 10^1 and 10^3 inflections.

The time taken to self-organize is not a strong constraint because it is linear in the number of symbols or inflections. Figure 7-16 shows that symbols can safely mature at approximately 1 for every 200 time units—let us round it to 1/360. If the time unit is somewhere between a tenth of a second and a millisecond, then somewhere between 10^2 and 10^4 symbols and inflections can mature in each hour of self-organization. If one hour of an average night's sleep is used for self-organization, then this is the number of new elements that can be investigated in an average day.

Developmental cost and hardware cost are constrained primarily by the distributed maps and the symbol relation map. Although I have not verified that this is the case,

[11]We will discuss aborting IIES more thoroughly in the next chapter; the **EQUAL** relation between inflections happens to be a simple case.

I suspect that the symbol relation map can be created using two variant distributed maps, connecting s symbols using $O(s^2)$ hardware, $O(s)$ development time, and $O(1)$ encoding cost.

Our set-based construction methods mean that the entire signal map has a constant encoding cost: it does not depend on the number of symbols or inflections. I do not know enough about the constants to make an order of magnitude estimate on development time, but am not overly concerned because it is likely to be only linear in the number of symbols.

Finally, let us make order of magnitude estimates for the hardware. The distributed maps cost $O(s^{3/2})$, so if we assume each link is a single synapse we have an upper limit of 100 million symbols (0.5% of cortex dedicated to each map = 10^8 neurons, 10^4 graph links per neuron, $(10^8)^{3/2} = 10^{12}$). If we assume each link requires 1000 neurons, then we have a lower limit of 2000 symbols ($2000^{3/2}$ is about 90,000. 10^5 times 1000 neurons = 10^8 neurons).

The symbol relation map requires $O(s^2)$ junctions, but there is only one per signal map. If we assume that 5% of the cortex is dedicated to each signal map and that each junction is implemented using between 1 and 1000 neurons, then the number of symbols might range between 30,000 (square root of 10^9) and 1,000 (square root of 10^6).

Intriguingly, both of these sets of numbers are not dissimilar to the number of words in a human vocabulary, though this is likely to be a mere coincidence.

7.3.3 Experimental Verification

I have used input from the four-way intersection scenario to verify that the signal map behaves as expected. For this test, I abstract the bidirectional link as a reliable message passing device and unique signal creator whose misbehavior is modelled with four parameters.

- Each time a signal is created, there is a chance $p_{kill} = 10^{-5}$ that another signal will be destroyed, and a chance $p_{dup} = 10^{-6}$ that the new signal will be a duplicate of an existing signal.
- Each time a message is sent, there is a chance $p_{vanish} = 10^{-6}$ that each element of the message will have vanished from one specialist's table, and a chance $p_{gc} = 10^{-3}$ that the other specialist will garbage collect a message element unmatched by its partner.

I then set up a system of two specialists, one for hearing, one for vision. Each specialist's sensory input is flattened, as for the IIES test in Section 7.2.4, and a message sent for each sample. The message contains all the features present, each two inflection, one marking it **PRESENT** and another marking its type (the vision specialist has **TYPE**, **COLOR**, **SIZE**, **MOTION**, and **RELATION**; the hearing specialist has **TYPE**, **DIRECTION**, and **LOUDNESS**). Thus, a visual observation containing a red car would result in a message containing **CAR=TYPE,PRESENT** and **RED=COLOR,PRESENT**.

To test the system, I train it on each of the movies used for the IIES test. Since it will turn out that only the midnight movie appears to converge to a stable set of relations during the 5,000 second time period of the original movies, I also recorded a 20,000 second movie (about 5.5 hours) starting at noon and sampling every 0.5 seconds and trained on that movie as well. The output of these tests is listed in Appendix B.2.

If the signal map is working correctly, then each symbol relation map should contain a set of relations equivalent to those learned in the IIES test. The inflection relation maps, on the other hand, should detect that the two inflections for **PRESENT** are the same, but should not find any other equivalencies (though the impoverishment of the flat sense representation may trick it into finding a small number).

We will also look at the predictive quality and convergence of the set of relations that are learned. We should expect to see predictive quality improve over time, though precision of predictions is likely to be quite low and the measure may be entangled with variation in activity during the course of the simulation (e.g. the simulation produces different sensory experiences in the immediate aftermath of an accident). Remember, as noted in Section 6.3, prediction quality will serve as rough sanity check rather than a benchmark of performance.

The number of relations learned should rise quickly after a brief pause while examples begin to accumulate; over time, the rate of new relation learning should slow as learning shifts from strong relations with frequent examples to ones that are weaker or appear more rarely, with the number jittering up and down as evidence moves weak relations back and forth across the boundary of acceptance. Eventually, however, the number of relations should stabilize at an approximate ceiling.

Are the Same Relations Learned? In the case of symbols, six of the seven movies result in precisely the same set of relations being learned as in our test of IIES. The seventh, the 3pm movie, apparently has a problem with the symbol **VAN** close to the end of the sequence and the hearing specialist loses three relations from its symbol relation map.

In the long movie, the hearing specialist learns 330 relations and the vision specialist learns 338 relations (compare with 156 learned by each in the short noon movie). The vast majority of the additional relations are **SUBCLASS** and **ENABLE** relations, as we should expect weaker relations to produce.

Of the 13 strong relations in the original noon movie, (9 **DISABLE** relations, 2 **CAUSE** and 2 **SEQUENCE**) 9 are missing from the hearing relations and 4 from the vision specialist's relations. Only two are missing from both (one being the odd **(DISABLE IDLE BLUE)** relation), and five of the nine relations missing from the hearing specialist are also missing in the other movies, so it seems likely to be due to a peculiarity of the original noon movie. Going the other way, there are 7 hearing specialist and 8 vision specialist **DISABLE** relations not in the original noon movie, six of which are common to both.

All told, the relations learned are as expected: virtually identical for the four movies used previously, and within the expected range of variation for the additional long movie.

Does Prediction Quality Improve? The prediction quality measure from Section 6.3 is extremely noisy, due to the variation in what is occurring in the simulator at different times. Moreover, the predictions made by this simplistic mechanism are extremely imprecise, so the size of the terms in the prediction equation,

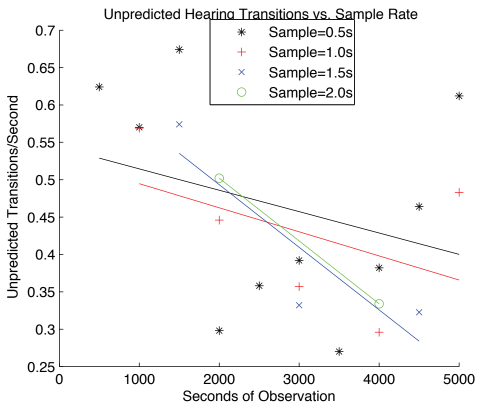
$$Q(p,c) = \frac{1}{p}(\sum_{i=1}^{p} \frac{s_i - k_1}{t_i}) - \frac{k_2}{c}(\sum_{i=1}^{c} u_i)$$

are very different—in particular, $\frac{s_i}{t_i}$ is generally much smaller than one. Instead of making a fairly arbitrary judgement about the constants to use, I will just consider the three components—correct predictions, incorrect predictions, unpredicted transitions—as a separate issue.

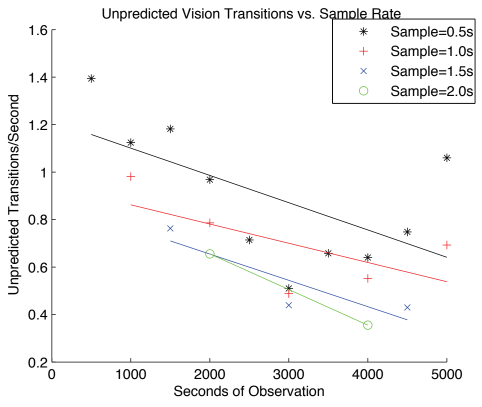
Inspecting the correct and incorrect predictions shows them to be hopelessly noisy due to the imprecision of predictions: the values are so small (given that dozens of predictions are being made during each time step) that I have no confidence that we can learn anything from them.

Unpredicted transitions, on the other hand, give us usable information despite the high rate of simulator behavioral noise. Figure 7-23 shows the rate of unpredicted transitions over time for each specialist and all eight movies, plotting the sampling and activity variations together for comparison. Each data point shows the average unpredicted transitions per second over the last 1000 samples, and a linear regression is calculated for each movie to get a rough gauge of the general trend.

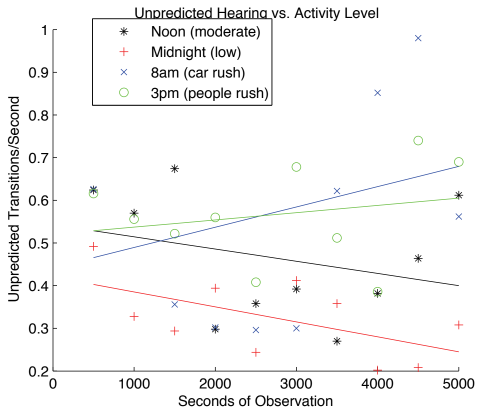
With the exception of the hearing specialist during rush-hour, every set of data



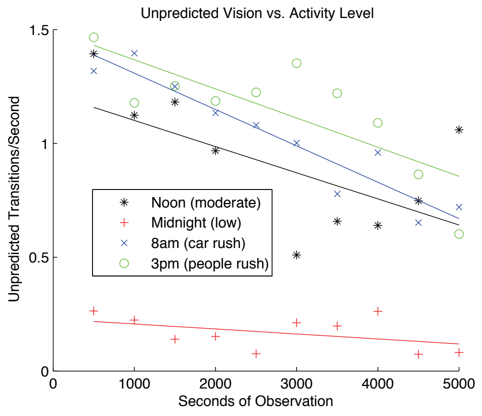
(a) Sampling Variation (Hearing)



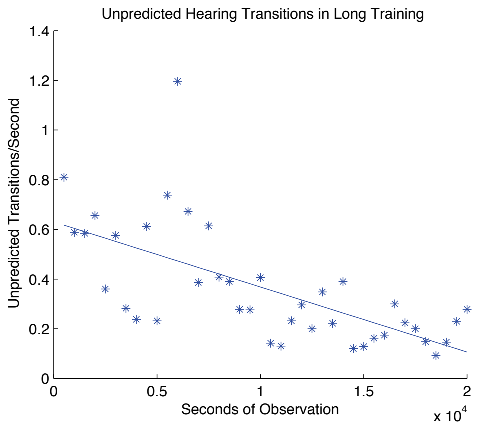
(b) Sampling Variation (Vision)



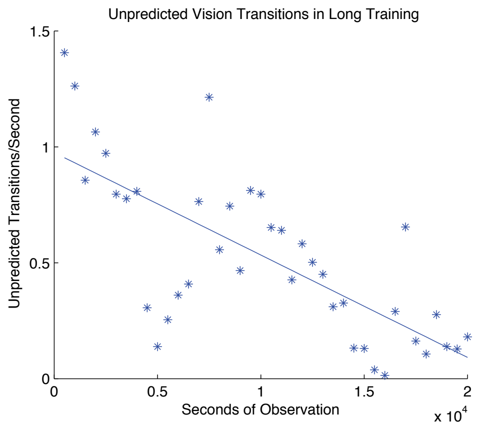
(c) Activity Variation (Hearing)



(d) Activity Variation (Vision)



(e) Long Run (Hearing)



(f) Long Run (Vision)

Figure 7-23: Prediction quality is an extremely noisy measure, but generally shows less unpredicted transitions over time. The graphs above show the average rate of unpredicted transitions in the set of 1000 samples preceding each point on the graph, plus a linear regression for each data set. Neither sampling rate (a,b) nor level of activity (c,d) appears to have a significant effect on the rate of improvement. Improvement continues even over a long training period (e,f), though with decreasing effect.

points has a downward trend. Neither sampling rate nor level of activity appears to have a significant effect on the rate of improvement, and improvement continues even over a long training period, though with decreasing effect.

Finally, judging by the higher level of scatter in the data points, the predictability of hearing observations is apparently less consistent than for vision observation, perhaps reasonable because the moving objects in the simulator are often heard before they are seen.

Thus, our measure of prediction quality gives at least a little bit of weak evidence that the relations learned in the signal map are capturing the dynamics of the simulator. We can expect no better of it given the indirectness of the measure and the lack of any serious attempt to use sensory data to model the environment. This increase in prediction quality is, in fact, due to nothing but communication bootstrapping.

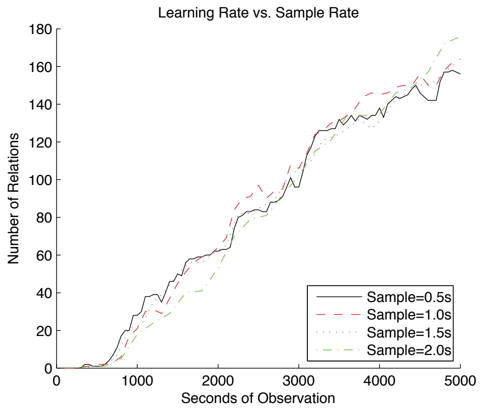
How Does the Number of Relations Change Over Time? In all of the movies yield, there is an initial pause while examples begin to accumulate, followed by a rapid climb as the strongest relations begin to be learned. The sampling rate appears to have no significant impact on the learning rate (Figure 7-24(a)).

The rate of activity, on the other hand, appears to have a significant impact (Figure 7-24(b)). After 5000 seconds, the noon and rush-hour movies are all still growing rapidly, though their rate of ascent appears to have slowed somewhat after the initial rush. The midnight movie, on the other hand, slows its rate of learning dramatically at around 2000 seconds, possibly due to the scarcity of training data. On the other hand, the rush-hour movies have a slower learning rate throughout, suggesting that there may be some intermediate rate of events that is best for learning.

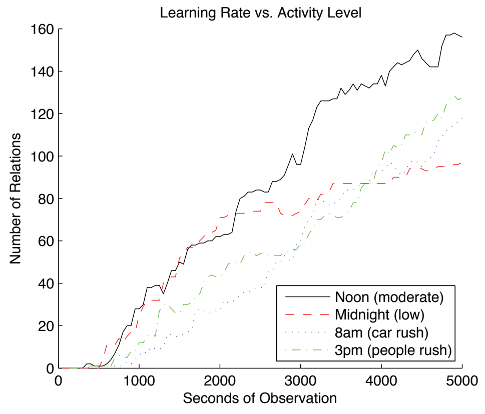
The long noon movie was motivated by the fact that all four of the original movies were still growing at 5000 seconds, to see whether the set of relations would stabilize. Indeed, after around 14,000 seconds learning from the long noon movie appears to have plateaued, though there is no way to tell from the graph whether another late set of weak or rare relations might still be building toward acceptance.

It is also notable that although the number of relations eventually plateaus, the growth of every set of relations is a jagged process marked by surges and reverses as relations slip back and forth across the line of acceptance. There is no reason to think that this will cease, nor should it given that fluctuations in the behavior of the world constantly present challenges to the existing relations.

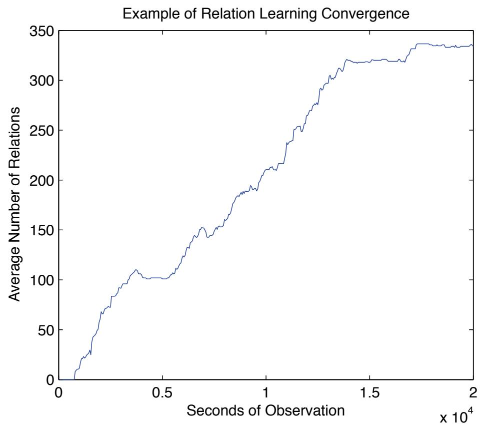
In sum, the signal map behaves as expected and each specialist learns a set of relations between the symbols it uses and the symbols its partner uses that capture structural information about the simulated world.



(a) Sampling



(b) Activity



(c) Long Training

Figure 7-24: The average number of relations in the two signal maps does not grow initially, as examples first begin to accumulate. Sampling rate does not appear to have a significant impact on the rate of learning, as shown in (a). Activity level (b) does appear to have an impact: higher activity yields slower learning, but in the midnight run, where activity is sparse, the learning slows markedly after the initial rush. Over a long period of training (c), learning slows as the frequent strong relations are exhausted and learning shifts to weaker and rarer relations, eventually appearing to stabilize to a consistent set of relations.

Chapter 8

Communicating Relations

A specialist should be able to use messages to tell its partner about the relations in its own model. We will not yet worry about what the partner does with this information, only that it can be sent.

Consider, for example, the stoplight hanging in the middle of our scenario's intersection. Figure 8-1 shows a close-up of the stoplight, and the relations regarding it that are observed by the vision specialist. It ought to be possible for the vision specialist to include these relations in its model and to inform the hearing specialist that of the six lights, only one green light and one red light are on.

I will show how relations like these can be communicated using inflections to mark some symbols as foci and others according to their relations to the foci. Doing so, however, forces us to accept that each message will only contain a small fraction of a specialist's model.

This threatens to undermine our mechanism for communication bootstrapping, unless the shared experiences of the specialists are reflected in their choice of foci. I therefore provide a *shared focus* mechanism and show how the IIES devices from the previous chapter can be extended to allow for changes in focus.

8.1 Encoding Relations With Inflections

The basic challenge in communicating relations is that inflections are applied to a single symbol, but relations connect two or more objects. Some additional structure must be imposed in order to connect together only the appropriate symbols—else we will not be able to tell whether the **BRIGHT** symbol goes with the green light or the yellow light. As always, there are many possible ways this could be done, and I claim only that my chosen mechanism is one that works.

I handle this problem by borrowing a trick from inflected human languages and

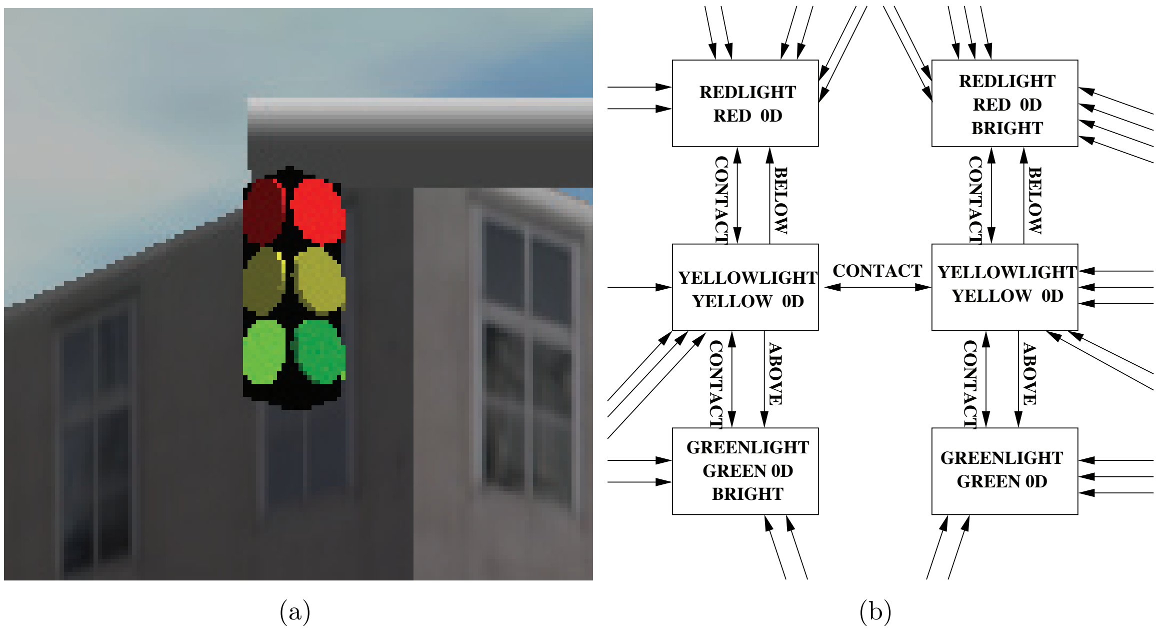


Figure 8-1: The fragment of visual observation containing the six lights of the stoplight (shown in close-up in (a)) is a network of 6 objects, 20 features, and 44 relations (b). The relations leaving the six lights go to the sky and pole above, the black box in which the lights are set, and the office building behind.

marking one object as the *focus* of the message—analogous to the subject of a sentence. That object is expressed in the message by its collection of features, each carrying a **FOCUS** inflection. Each of the objects it is related to is expressed in the message as well, with their features carrying an inflection encoding their relation to the focus.[1]

For example, if the focus is placed on the left-hand green light in Figure 8-1, then the message sent by the vision specialist would contain:

> **GREENLIGHT=FOCUS**
> **GREEN=FOCUS**
> **BRIGHT=FOCUS**
> **YELLOWLIGHT=ABOVE & CONTACT**
> **YELLOW=ABOVE & CONTACT**
> **0D=FOCUS & ABOVE & CONTACT**

(where **0D** is a feature marking a visual object as being small) plus a few more symbols and inflections for the building and light box.

[1]The original work on communication bootstrapping assumed its observations would already have this structure, neatly avoiding the issue.

Notice that this also means that an object need not be named with a symbol in order to be communicated from specialist to specialist: its participation in a relation implies its presence. Everything besides the focus, however, may be ambiguous in its identity. The message above, for instance, does not give any information about how many yellow lights are related to the green light: there may be one that is both **ABOVE** and in **CONTACT** (as is actually the case), one **ABOVE** and a different one in **CONTACT**, or even dozens of yellow lights that all have the same relation to the green light.

Larger fragments of the model can be conveyed in a single message by increasing the number of relations. One way to do this is with multi-step relations, like **ABOVE-THEN-LEFT** for an object that is **LEFT** of a second object, which is in turn **ABOVE** the focus. Another is to use multiple foci, each associated with an identical set of inflections, so that **FOCUS1** goes with **ABOVE1**, **FOCUS2** goes with **ABOVE2**, etc.

If there are r relations, then encoding s step relations for f foci requires $f\sum_{i=0}^{s} r^i$ inflections. A specialist whose model uses a moderate two dozen relations—eleven predictive relations and the rest peculiar to that specialist—takes 25 inflections per focus for one step and 601 inflections per focus for two. In Section 7.3.2, we estimated that a model of intelligence can afford a range of roughly 10 to 1000 inflections, so going more than one step in relations is simply not likely to be worth the cost. Going a single step, however, it is reasonable to assume that we can afford to maintain several foci.

Increasing the number of relations in this way does, of course, increase the time it takes for specialists to learn to interpret one another's inflections. The number of inflections is still much smaller than the number of symbols, however, so it should not concern us overly much. There is, however, a new alignment problem: if each specialist uses several **FOCUS** inflections and several **ABOVE** relations, it is important that they group them in the same ways: otherwise relations will end up connected to the wrong objects!

Finally, note that this method of encoding relations into messages largely preserves the error tolerance of symbol/inflection messages. A symbol or inflection that is lost or misinterpreted affects only those parts of the message that it is directly involved in. Better yet, since most objects have more than one feature, a symbol error does not necessarily mean an error in the relation that symbol supports. The only truly critical elements are the focus inflections, without which a message lacks context.

We now have a usable definition of the *working set* of a model: it is the set of objects marked as foci, the set of relations connected to the foci, and the set of objects at the other end of those relations.

8.2 Shared Focus

With this scheme for encoding relations, we are forced to admit that only a small fraction of a scene can be communicated at any one time. The vision input from which Figure 8-1 was excerpted, for example, contains a total of 64 objects, 242 features, and 515 relations. Worse, current sensations are only one of the sources of model complexity: we are also likely to want to incorporate memory, prediction, higher-level interpretations giving a deeper understanding of the scene, etc. It is simply not reasonable to assume that the whole model, or even a large fraction of it, can be communicated in a single message.[2]

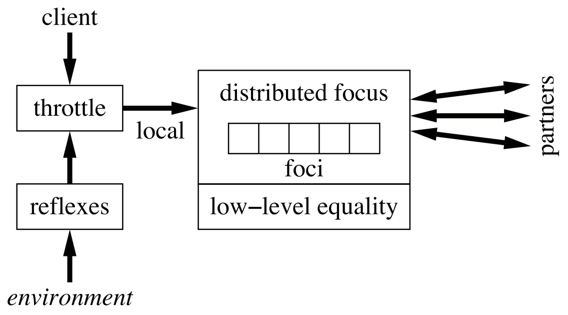
This is a threat to communication bootstrapping at its most basic level, since the signals of specialists no longer necessarily contain shared aspects of their experiences. If vision keeps its foci on the silent buildings and hearing keeps its foci on the sounds of cars and pedestrians off to the side and out of sight, then the messages they send have no shared experiences that can allow communication bootstrapping to occur. While some relations may still be learned, we cannot expect them to capture rich or interesting information unless the foci are placed on related objects.

Nor is this just a problem for learning: when a specialist receives a message from its partner, we would like the message to somehow affect its model. In a large, complicated model, however, the objects the message refers to are likely to be ambiguous. For example, if hearing mentions the sounds of a car idling close by in front (e.g. **IDLE=FOCUS2**, **70=FOCUS2** and **F=FOCUS2**), then which of the several cars waiting for the light does it refer to? While there may or may not be any good way of resolving such ambiguity, if a specialist matches one of its foci to the focus in the incoming message then it can at least ensure that the ambiguity is resolved consistently as long as neither focus moves.

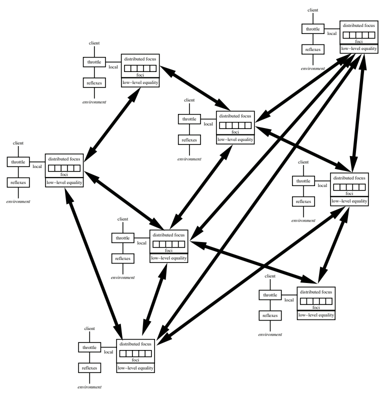
It may be helpful to think of foci as the "pronouns" in a conversation between specialists. With three foci, for example, a specialist can designate objects as "this," "that," and "the other." If we provide a way for specialists to agree on which object is "this" and to know when they are agreed, then it will make it much easier for us to build mechanisms that do something useful with the "conversation."

Let us call the subject of such an agreement a *topic*. Two objects in different specialists are thus the same topic if each specialist's object matches its interpretation

[2]While it is possible to design a scheme that would allow most or all of the model to be sent in a single message, my experiences lead me to doubt that such a scheme would be affordable.



(a) Shared Focus Architecture



(b) Example Focus Network

Figure 8-2: Specialists share their foci using a distributed mechanism that balances local requests against the goal of matching their partners' topics. Local requests may come either from some other device within the specialist or from reflex responses to environmental stimuli. The distributed focus mechanism I use will misbehave if local requests are too frequent, so they are throttled.

of the other's description. We will remain discretely silent on how exactly a specialist tests for a match: let it simply be stated that there are many plausible standards, and that the choice of standard is likely to be specialized for each specialist. For example, the hearing specialist may use the specialized knowledge that something that is loud is not also quiet.

I address these problems with the distributed mechanism shown in Figure 8-2. This mechanism attempts to balance four competing goals: dominance, fairness, agility, and longevity.

- **Dominance** means that a few topics occupy almost all of the foci on almost all of the specialists in the network. Without dominance, specialists are not participating in the same "conversation."

- **Fairness** means that any specialist in the network has an equal chance to propose a new topic and have it become dominance. Without fairness, the system cannot respond reliably to surprises.

- **Agility** means that when the dominant topic shifts, it shifts quickly to a new dominant topic. Without agility, the network of specialists cannot respond quickly to surprises.

- **Stability** means that when a topic has become dominant, it is likely to stay there for a long time. Without longevity, the "conversation" between specialists is unlikely to stick to a topic long enough to do anything useful with it.

The core of my mechanism is a *distributed focus* device that balances local requests against the goal of matching its partners' topics. The device misbehaves when local requests arrive too quickly, so I add a *throttle* that manages streams of incoming requests. Finally, I break up the chicken-and-egg problem of communication bootstrapping (no relations without dominance, no dominance without topics, no topics without relations) with a set of *reflexes* that direct each specialist's foci independently to objects likely to be focused on by other specialists, and a *low-level equality* device that uses sub-symbolic maps between senses to detect when two objects are the same topic.

Complete data sheets for the distributed focus and throttle devices may be found in Appendix C.4 and Appendix C.5, respectively. Low-level equality is implemented with mechanisms peculiar to each specialist, so it is not detailed.

8.2.1 Distributed Focus

Distributed focus operates on a network of n devices, each containing a set of f foci. There is no constraint on the structure of the network. The foci act as a set of registers, with each focus pointing to an object in the model (this may be implemented using a distributed map). Associated with each focus is a *privilege* rating, which will be used to break symmetry between competing topics. When a message is sent, the privilege rating for each focus could be added using a special set of inflections or conveyed on a parallel set of connections.

Changes in topic come from two sources: local requests and changes in a partner's foci. The local requests are a stream of references to objects that may or may not already be pointed to by a focus. Changes in a partner's foci are taken as implicit requests for a matching change in the device's own foci, but only if the topic can be interpreted.

The Purpose of Privilege To become dominant, a topic needs to invade devices throughout the network. When topics are competing for space, however, we find ourselves with a symmetry breaking problem: if the competing topics can invade one another, then they are likely to thrash, trading devices back and forth in a never ending tussle. If they cannot invade one another, then they are likely to end in deadlock, with neither becoming dominant.

I will avoid this type of thrashing with a solution based on insight from Rauch's work on spatial separation and group evolution[30]. Rauch uses a cellular automata host/disease model where each cell has three states: live, dead, and infected. There is a fixed death probability that an infected cell will die, a fixed growth probability that a live cell will spread to adjoining dead cells, and an evolvable infection probability that an infected cell will spread to adjoining live cells. Rauch discovered that, across a large range of parameters, the probability of infection evolved to sustain a stable but shifting population of all three states: local perturbations are damped out by the action of other areas, and changes in the global parameters change only the characteristic diameter of regions of cells with each state.

Mapping this back to the shared focus problem, we can consider devices where a competing topic is invading others to be in the infected state, devices where it is subject to invasion to be in the dead state, and devices lacking the topic to be in the live state. Rauch's work tells us that we can expect thrashing to be a very durable behavior across a wide range of algorithms. It also tell us how to prevent thrashing: stability breaks down when the characteristic diameter of any state's region approaches that of the network.

I will use this observation to break the stalemate between competing topics. At low probability, a spreading topic will become privileged for a short time, during which it cannot be invaded and almost always succeeds in invading. This will break symmetry, quickly spreading the topic throughout the network. Paradoxically, once the privilege ends, the topic will typically no longer be spreading, and can be quickly displaced by its competitors.

In terms of implementation, the normal privilege level is zero: when privilege rises above this level, it leaks away at a constant rate.

Update Mechanism A distributed focus device updates its foci before each message the specialist sends.

Topic requests come from two sources: the local request stream and changes in neighbor foci. Each device caches both the most recent values from each neighbor and the values used in the last update. At the update, these values are compared, and if either the value has changed or it has gained privilege, then that is treated as a request for focus and added to the collection of local requests (which may contain multiple requests for the same topic). When there is privilege in the neighbor, it carries along in requests, decremented by one to ensure dissipation.

To prevent communication delays leading to oscillation, elements of the cache will sometimes skip updates: each element has a probability p_{wait} of simply ignoring an update, neither taking a new value nor comparing against the old to produce a request.

Next, the device selects f requests to service (less if there are less requests total), giving preference to those with higher privilege (local requests are never privileged).

Each serviced request then has a small chance p_{priv} of its privilege spontaneously rising to the maximum privilege value t_{priv}. The chance of privilege is set so that the expected number of privileged topics in the network is expected to be very low: $p_{priv} = \frac{k_p}{n \cdot t_{priv}}$, where k_p is a small constant and n is the number of devices in the network.

Finally, the request's topic is compared with the current foci: if it is already in a focus, then the privilege of that focus is set to the maximum of its current value and that of the request. If the topic is not in a focus, then a target focus is selected according to some *policy* and its replaced with the request unless its privilege is higher.

Notice that there is no mention of "salience" or other ratings of relative topic value—instead, every local request for attention, no matter how trivial, should have a good chance of dominating the attention of the entire system. It is tempting to give preference to reflexively important topics like loud noises. I choose not do so, however, for several reasons:

- Systematically preferring some topics means systematically shying away from others. Those disregarded topics will sometimes be important, and I do not want distributed focus to add to the barriers they must overcome.

- There may be no clear way to compare preference strength across specialists.

- Preference can still be expressed clearly by repeated local requests.

Analysis There are four variables controlling the behavior of distributed focus: p_{wait}, k_p, t_{priv}, and *policy*. Using the characterization detailed in Appendix C.4, I will choose $p_{wait} = 0.5$, $k_p = 2$, $t_{priv} = 20$, and a policy of **follow**—meaning that when a partner changes from one topic to another, the device will make the same change if the old topic is in its foci, and replace randomly if the old topic is not.

With these parameters, distributed focus behaves as desired so long as the rate at which local requests for new topics appear is very low throughout the network. The number of specialists requesting a topic does not matter, only the number of topics being requested: when topics are being requested quickly, both longevity and dominance suffer because new topics are invading the network before the current topics can even spread throughout. Preventing this form of misbehavior will be the responsibility of the throttle.

8.2.2 Throttling Requests

Although I originally developed my throttle mechanism in order to support distributed focus, its turns out to be usable not only for focus requests, but for shaping any set of event streams.

The rate at which events occur in an intelligence can be all over the place, from slow and lazy to fast and furious. For example, a stray thought might touch off a cascade of brainstorming, or a turn onto a busy street might suddenly result in hundreds of cars flowing through the visual field on the other side of the dividing line.

If the events consume a limited resource (such as foci), we need to be able to control the consumption in both regimes, servicing every event when events come slowly and rationing the resource when events come quickly. For example, distributed focus only behaves well when the rate of requests for new topics is low: when requests come too quickly, topics cannot dominate for long enough to enable collaboration between participants. When requests are coming slowly, however, we want topics to become dominant quickly and reliably.

The challenge comes from the fact that events may come from multiple sources, and it is important that a previously silent source be able to quickly invade the flow and take control. For example, consider a vision specialist with two flows competing to be local requests in the distributed focus system: one from its observations, the other from internal manipulation of models. If the system is wandering along, deep in thought, and a car lurches in its direction, the observations need to be able to suddenly demand attention. On the other hand, driving along a highway with many cars going the other direction, a sudden insight should be able to invade the flow of information from observations.

These problems are further complicated when the desired rate of events is very slow (as is the case for distributed focus) since that makes it difficult to estimate the rate over short time spans.

I will handle this with a throttle mechanism closely related to a Token Bucket Filter[11], a widely available network traffic-shaping method. A Token Bucket Filter works by matching two rates of flow: a rate at which packets arrive and a rate at which tokens for service are generated. A packet is serviced whenever there is a token available for it, giving three regimes of operation. When packets are slower than tokens, tokens accumulate, "saving up" to allow a burst of transmission if the packet rate spikes. When packets are faster than tokens, the queue fills and packets begin to drop. On the boundary, when the arrival and service rates are the same, packets are serviced immediately and tokens do not accumulate. My mechanism is similar, but also mixes streams fairly.[3]

Mechanism Events arrive at a throttle from s different source streams. The rate and distribution of the incoming streams is unconstrained. The throttle emits a filtered stream of events, to be constrained to a sustained rate of no more than r_s and burst of no more than b_{max} events per burst.

The throttle mechanism is quite simple. Each source stream has an associated activity level. When an event arrives, the source's activity level is checked: if it is more than $b_{max} - 1$, the event is discarded, otherwise the event is serviced—sent to the output stream.

When an event is serviced, the activity level of the source that it came from is raised by one plus a random number in the range $[-c_{var}/2, c_{var}/2]$, where c_{var} is a constant range of cost variation. This variant cost works to prevent stuttering by keeping event servicing from falling into a regular rhythm.

The activity level of a source slowly drops. Every source activity level above zero decreases at a rate of r_s/n_{active}, where n_{active} is the number of sources with a positive activity level. Thus, the total activity level of the system decreases at rate r_s, no matter how many sources are active.

When a source is idle its activity settles toward zero, banking away the ability to transmit a burst when it next activates. When a source is active, the rate that activity leaks away dictates how often its events can be serviced, and that rate scales inversely proportional to the number of active sources, effectively controlling the overall rate.

The throttle behaves well under a wide range of conditions, suffering only during the transition between the sparse and rationed behavior regimes, and then only moderately. Full details are given in Appendix C.5.

8.2.3 Reflexes and Low-Level Equality

The remaining two components, reflexes and low-level equality, work together to provide a foundation for distributed focus. The reflexes provide a basic level of focus direction and low-level equality provides a basic test for topic matching. At first, these are all the network of specialists has to support distributed focus—they are, in fact, the bootstraps that communication bootstrapping will seize upon.

I will not go into mechanisms for either component, because both are likely to be highly specialized for particular specialists. It is enough to note that the reflexes should respond to the sort of things that human reflexes respond to, like change

[3] I would not be surprised if I have reinvented a mechanism that already exists somewhere in the field of computer networks.

and motion, and that the low-level equality can be built using ordinary cross-modal learning (for example, a Kohonen map[26], Coen's slices[12], or Roy's cross-modal approach[33] would all be appropriate).

For example, the vision specialist's reflexes might focus on a **WALKLITE** that has just appeared while the hearing specialist's reflexes focus on a new **CUCKOO** sound. The low-level equality map between the vision and hearing specialists connects the coordinates of the two senses and suggests that these are the same thing. Now the two specialists' signal maps can begin to learn that they are **EQUAL**.

Once the specialists know that **WALKLITE** and **CUCKOO** are **EQUAL**, then they can start using them to learn about other features and objects that relate to them. For example, when **PERSON** is in **FRONT** of the **WALKLITE**, the specialists can learn that the **CUCKOO** only happens when there are people around. Another example: each **WALKLITE** plays its **CUCKOO** sound at a consistent volume, so specialists can learn that to associate the **WALKLITE** with those decibel levels. Likewise, in future work when we look to have specialists invent abstractions, the focus needed to support this process of invention can come from the existing relations that the abstractions incorporate.

The reflexes and low-level equality need not be particularly smart or complicated mechanisms, just enough to get the first few relations up and running. When the specialists' models are better developed, they will still be running, and it is important that they not interfere too much with the more sophisticated understanding that the specialists develop. The reflexes keep driving the foci, but begin to compete with other drives such as curiosity and prediction (which we will not explore in this document). The low-level equality becomes just one more factor in the test for topic matches, available only for immediate observations.

8.3 Focus and Relation Learning

We have been talking about how signal maps can only safely learn relations from matched foci, but have not yet explained exactly why or how. Both, fortunately, are fairly straightforward, and have already been seen in the simpler case of **EQUAL** relations between inflections in Section 7.3.

If we allow a signal map to learn relations between unmatched foci, then there is no reason to expect that the messages will reflect any sort of shared experience. If the hearing specialist has the **CUCKOO** sound in a focus while the vision specialist is considering the buildings, and we allow learning from unmatched foci, then the relationship between **CUCKOO** and **WALKLITE** will be degraded because

WALKLITE is not related to the buildings that the vision specialist is focusing on.

Thus, we must only allow the symbol relation map to run IIES on symbols related to matched foci, just as we must only allow the inflection relation map to run IIES on pairs of **EQUAL** symbols. We still do not, however, need multiple layers in the symbol relation map.

If we knew which symbols were associated with which objects, all we would need to do is add a one-round cache and modify IIES to handle four special cases:

- **An object obtains focus and a period of matched focus begins:** If a pair of symbols are both present before the period begins, it is not possible to tell which appeared first, so the IIES finite state machines detecting examples for the pair should be placed in a superposition of states 2, 4, and 5 from Figure 7-20. An **EQUAL** relation gains positive evidence immediately, and any other predictive relation must wait for one or both of the symbols to vanish before it can begin running the IIES normally. If only one symbol is present, the IIES runs normally, transitioning from state zero based on the current input.

- **An object obtains focus as it appears:** IIES starts with a transition from state 0 to the cached input, then continues with the current input.

- **An object loses focus and a period of matched focus ends:** IIES is disabled and returns to state zero to await the next period of matched focus.

- **An object loses focus as it disappears:** IIES transitions based on the current input (possibly producing an example), then is disabled and returns to state zero to await the next period of matched focus.

With these modifications, IIES should be able to learn safely from periods of matched focus and remain idle in between.

Unfortunately, the chicken-and-egg problem is still with us, for until learning begins, there is no way to tell which symbol is associated with which object. This is particularly problematic since symbols with the same inflection may refer to different objects, when the inflection expresses a relation rather than focus.

What this means is that we cannot actually implement the modifications to IIES without opening up a broad new area of questions about assumptions and design decisions, including:

- Should relation learning be restricted to symbols related to the same foci, which reduces the opportunity for serendipity, or should the symbols for any foci be able to interact, which allows interference between unrelated foci?

- How should learned relations be incorporated in the decision about whether two objects match? Especially, what is the role of **CAUSE**, **SEQUENCE**, and **DISABLE**?
- If any two foci can be matched, how can relations between inflections be learned effectively, since evidence matching **FOCUS1** to its partner's **FOCUS2** is evidence against matching its partner's **FOCUS3**?
- Should focus be allowed to shift to a partner's object if there is no clear translation into a specialist's own model?
- To what degree are we willing to alleviate these problem through (expensive and possibly fragile) hard-wired focus connections or a sequenced deployment of focal mechanisms?

I will not attempt to resolve these questions in this document, particularly given that it would involve making decisions about models, and I am not yet comfortable that I can navigate those decision safely.

What I will ask instead is this: can we learn enough to start matching symbols to objects *without* modifying the signal map in any way, merely allowing the shared focus mechanism to determine the contents of the messages?

This is actually a purer Communication Bootstrapping approach to the problem: we simply rely on the world to have enough structure that some relations will stick out like a sore thumb and attract our attention if they are allowed to.

To test this idea, I once again use input from the four-way intersection scenario. As before, I abstract the bidirectional link and set up a system of two specialists. This time, however, I do not flatten the sensory input, but keep it intact so that I may send relations.

For the shared focus mechanism, I use 4 foci per specialist, and set the throttle to limit the rate of requests to 1 per message. Reflexes will request focus for objects that appear, add features, or begin to move, but not those that disappear, lose features, or halt. Low-level equality is taken from object unique identifiers provided by the simulator, or by the direction to an object in hearing space being within 10 pixels of the center of an object in vision space. Only those objects that are matched, however, are transmitted in the messages. No matching is done besides via low-level equality.

I train the system on simulated input, starting at noon and sampling once per half second for 5,000 seconds (about 80 minutes), then examine what the system has learned. If our hopes are borne out, then there should be strong sets of relations learned between closely related objects.

Examining the results (shown in Appendix B.3), we see that both vision and hearing have acquired 448 relations. A full 240 of these relations are ones that could not have been learned in the flat representation, as they involve features like **STEPS** or **DARK** that are nearly always present somewhere in the input. The set of relations appears to have largely stabilized as well, changing little during the last 1000 seconds.

Most important, however, are the 79 relations connecting together four clumps of strongly associated symbols—nearly one in every five relations learned. I will rate a pair of symbols a member of this set if it has an **EQUAL** relation and at least three other **SUBCLASS** or **ENABLE** relations. The four clumps correspond to:

- **Cars:** Mid-level loudness (**60**), **DRIVE**, and **ENGINE** connect to moderately small (**8D**), **DARK**, and **CAR**. In addition, **DRIVE** and **ENGINE** connect to **SEDAN** and **ENGINE** connects to small (**0D**) objects.

- **People: STEPS** connects to **PERSON** and **ADULT**

- **Things in front:** Sounds directly in front (**F**) connect to moderately small (**8D**) and **DARK** objects (apparently either cars or people).

- **Close things:** Moderately loud sound (**70**) connects to mid-sized images (**16D**).

As can be seen, these strong sets of relations could be used to pair up the hearing and vision specialist symbols for either people or cars. This shows that the chicken-and-egg problem of bootstrapping and symbol/object mapping might be averted through an underlying shared focus device and strong relationships between streams of sensory input.

8.4 Potential Benefits from Shared Focus

Learning with a focus of attention has the potential to transform the system of specialists in a profound manner, provided we can overcome questions related to the symbol/object dichotomy.

First, the problems of saturation we encountered in the last chapter simply vanish. It will no longer be impossible to learn about red lights just because there is always some light that was red: the foci separate the objects from one another. Even unvarying objects like the sky can be learned about, as the foci move near it and away from it, cutting a single interval of object presence into a series of focal encounters.

Second, a specialist need not be distracted by the complexity of its environment. The throttle limits the rate at which even the wildest environmental stimuli can affect the shared focus. More importantly, if specialist's foci are partially guided by communications from its partners, then it will tend toward topics on which it can communicate to its neighbors, and that therefore have the most potential for additional incremental learning.

Finally, a specialist will only be able learn about things that are closely related to things it already knows. To learn a relation between symbols, they must be associated with objects that can be matched, which means they must have low-level equality or else involve symbols whose relations are already known.

All together, this implies that our system of specialists may be able to naturally adjust its interpretation of its environment to match its level of understanding. In the beginning, surrounded by an uninterpretable "blooming, buzzing confusion," the system would necessarily focus on the simplest and most tractable elements. As it begins to understand its environment, it can focus on more subtle parts of the scene, building off its knowledge of the simpler.

Chapter 9

Contributions

I began this dissertation with a simple question: how might the various parts of an intelligence learn to work together as a unified whole? I propose that this question is key to our understanding of intelligence.

I hypothesize that the specialist parts making up human intelligence learn to cooperate as a byproduct of learning to communicate, and that they learn to communicate by exploiting the phenomenon of communication bootstrapping. This also suggests the more radical hypothesis that human intelligence may arise largely from the struggle of the various specialists to understand one another.

I have presented a roadmap for a serious investigation of these hypotheses, a venture that will require far more than a few years of work by a single graduate student. In this dissertation, I have laid a solid foundation for work in this area, and have taken the first few steps following my roadmap. As I have progressed, I have checked that my ideas are not unreasonable by testing them against a running example: two specialists, vision and hearing, observing a simulated four-way intersection.

My work in this dissertation contains three key technical contributions: developmental cost (Chapter 4), failure simplification (Chapter 5), and extension of communication bootstrapping (Chapter 6, 7 and 8).

Developmental Cost Investigation of intelligence through exploratory engineering has suffered from a lack of useful constraints. I note that engineering and morphogenesis labor under similar constraints, so adding constraints from biological development may help us to discover new organizational principles similar to those exploited by biology.

To that end, I introduce the notion of developmental cost, so that the cost of a device includes not only the hardware and time to run the mature device, but also the cost of encoding and running a program that grows the device and its response to defects in development. We may thus measure the quality of a device in terms of asymptotic complexity using cost assumptions derived from biology, yet likely to be undisturbed as our understanding of biological details continues to change.

My communication bootstrapping designs take developmental cost into account, ensuring that they are within the envelope of biological plausibility and allowing us to make rough estimates of how they might be employed in a human brain (Section 7.3.2).

Failure Simplification Building devices for a use in models of intelligence presents a major software engineering challenge: the specification for a device is often contradictory, hard to define precisely, and subject to violation by other devices' misbehavior.

Failure simplification embraces these problems rather than attempting to avoid them. As we develop a device, we create a dossier that describes its major phases of good behavior and misbehavior. Failure simplification means that we recognize that we cannot prevent misbehavior, and instead look to damp its impact. Typically, this is done by modulating the behavior of a device to select only misbehaviors that are easy to detect and respond to. A common technique for this is pre-emptive failure, which sacrifices some good behavior in order to avoid complexity in the boundaries between phases.

Each of the devices I use in the extension of communication bootstrapping is analyzed with a dossier documenting its major phases of behavior—the codetector as the example in Chapter 5, the rest in Appendix C. Their use is consistent with the philosophy of failure simplification, taking into account the likely modes of misbehavior.

Besides aiding our study of intelligence, failure simplification might be applied to the more pragmatic field of software engineering. As our programs grow in complexity beyond the ability of any single human to comprehend, techniques like failure simplification will become ever more important.

Extension of Communication Bootstrapping Previous work on communication bootstrapping was essentially a proof of concept that the phenomenon could take place in a brain. In this dissertation, I dig into the practical details of what is actually needed for communication bootstrapping to be useful in a model of intelligence.

In Chapter 6, I generalize the original communication bootstrapping architecture. I then propose a standard for judging the success of communication bootstrapping and a new means of encoding inflections that allows more complicated messages to be sent.

Chapter 7 deals with the failure of the assumptions that the previous work relied on. I first build up a complicated mechanism that allows me to abstract the messages away from the details of their encoding (Section 7.1). Along the way, I describe what sort of brain activity would be evidence that a similar process is being carried out in human brains. I then describe how a specialist can learn to interpret message elements, using an incremental mechanism to detect examples that provide evidence for and against a set of eleven predictive relations (Section 7.2).

Finally, Chapter 8 shows how messages can be combined with a notion of focus to communicate model fragments from one specialist to another. This chapter also deals with the problem of getting specialists to agree on a set of topics to focus on, and shows how this apparent difficulty might actually aid learning by allowing the system to ignore things that it is not close to understanding.

The result of all this work is a design that learns to communicate between two specialists and in the process sifts interesting knowledge from a cluttered environment. Nor is it a closed design: the design nearly begs for the next steps in its development—a proper discussion of models, more investigation of focus, a means of building abstractions, and a higher-level attentional drive to compete with its reflexes.

9.1 The Larger Architectural Vision

Allow me to indulge, for a moment, in speculation about how a human-level intelligence based on communication bootstrapping might work.

The whole would consist of several large parts, one for each major specialist contributing to intelligence—vision, hearing, sensorimotor, language, social, etc. Each of these, in turn, would be composed of several smaller specialists: the vision specialist, for example, might be composed of one specialist for shape, another for color and texture, another for spatial relations, and so on, each with its own peculiar model and reasoning system. All told, I would expect there to be around a dozen large specialists and somewhere between 20 and 100 smaller specialists. Each specialist connects to several related specialists, so the whole forms a mesh-like network a few hops in diameter.

These specialists rest atop a supporting infrastructure of sub-symbolic processing, devices that handle the initial processing of sensory input and the routine parts of motor control. Simple, information-intensive behaviors like hand-eye coordination are carried out at this level, and self-organizing maps between domains give the low-level equality relations needed for communication bootstrapping to start learning symbolic relations. Predictions in the symbolic specialists cause anticipation in the senses and actuation in the motor controls.

At first, each of the specialists is lost in a sea of meaningless input, unable to communicate with its partners. As the lower level infrastructure organizes itself, it begins to drive the symbolic specialists and they begin to learn to interpret one another's messages. As the specialists start to learn to communicate, the shared focus mechanism starts to points them at the same topics, and more and more is learned as the specialists start to have more experiences in common. Since two specialists may interpret the same symbol in different ways, translation may in fact become extrapolation, and the circulation of a structure through the network of specialists may effectively carry out cooperative reasoning upon it.

Now there is a danger that the learning will senesce, as the specialists get driven only towards things that they already understand. The next stage of learning, once some basic competence has been established, is driven by surprise. When a specialist fails to predict something, or when a specialist's prediction fails, the failure is an irritation that requests the focus, drawing the system to poke and prod at things it does not understand.

At the same time, a surprised specialist begins to propose explanations, inventing abstractions that combine objects and relations together into an abstract feature. The explanation is honed on a specialist's own experiences, then judged by whether any partner can relate the new feature to its own features or proposed explanations. These explanations might even contain strategies for shifting the focus about, or for building explanations, so that a specialist becomes better at cooperating and learning as it matures.

Thus we may envision a system that starts simple and adds to its understanding of the world one layer of explanation at a time, each layer patching mysteries in the layer below it and introducing newer and more subtle forms of confusion. The growth of understanding is regulated by the ability to communicate, keeping any specialist from surging off into unsupported flights of fancy—or at least from going very far.

Moreover, each additional insight comes in two parts: an educated guess from a specialist's own experience, and an unpredictable discovery of how that guess relates to the experiences of other specialists. I see, in this duality, the opportunity for true creativity on the part of a communication bootstrapping system. An artist begins by doing, then produces a great work by recognizing and exploiting the potential in what has been done. So too may a specialist's proposal become something more when other specialists are asked to interpret it.

In this vision, communication bootstrapping is not a silver bullet that solves all problems—the work that must be invested in the infrastructure and the reasoning systems for each of the specialists far outweighs it. Rather, I see communication bootstrapping as the teaspoon of baking powder that allows the cake to rise.

9.2 Next Steps

The next steps that I intend to take toward this larger vision are laid out in the roadmap all the way back in Section 1.2. The most immediate targets are:

- A clear definition of what makes up a specialist's model, and an affordable hardware design to support it.
- Actuation through prediction, and the addition of a motor specialist to the system, which can then try to learn how to safely cross the street.
- Surprise, from unpredicted changes or unfulfilled predictions, as a driver for focus.
- Invention of abstractions in response to surprise, to be judged by whether a partner can relate something in its experience to the abstraction.

Besides these, there are a number of open problems where important contributions could be made, including:

- refinement of developmental cost metrics by incorporating more information from synthetic biology, developmental biology, and neuroscience.
- refinement of the various devices to improve costs or capabilities. For example, the cost of the relation maps in Section 7.3 can likely be greatly reduced.
- development of specialized models and reasoning systems, both improving the existing specialists and adding new specialists such as language or social reasoning.
- integration of more effective and/or realistic processing methods for sensory input.
- improving the theoretical analysis of the conditions for communication bootstrapping.
- construction of a theoretical model for failure simplification.

- investigation of the relationship between language acquisition and the development of cooperation between specialists, including the question of whether an intermediate proto-language specialist might simplify the cooperation problem.

9.3 Wider Implications

I will be so bold as to propose that this dissertation may represent an important new direction in the study of human-level intelligence.

Despite a fabulous increase in knowledge, the study of human-level intelligence has been largely stalled for the past few decades. Neuroscience and cognitive science have discovered a great many facts about how brains and minds work, but a broadly integrated quantitative model has stubbornly failed to emerge. Artificial intelligence has produced a great many clever systems that do remarkable things, but such systems are notoriously hard to build on or combine, and it is unclear which of these systems actually represent progress toward the larger goal.

I believe that work changes this landscape in two ways. First, I point out that it is not even clear how our brains integrate their parts into a computational whole. The analogy between the problems of development and the problems of engineering make this largely neglected problem a tantalizing field for investigation. I personally hope that it will be a case, like Waltz's shadows[40], where adding complexity simplifies the problem. This is my own personal hobby-horse, and where I have placed my own bet that a key part of the answer will be found.

Perhaps more importantly, however, developmental cost and failure simplification have the potential to improve the standards for exploratory engineering research on intelligence. Developmental cost leads to very different designs than thinking about conventional computer hardware, as the reader will have noticed while reading Chapter 7. Better yet, it also gives us a way to judge the plausibility of a device without knowing how it will be used or getting embroiled in the fine details of biology. Failure simplification, on the other hand, is a basically different philosophy of device integration. If widely adopted, it may make it much easier for researchers to build on one another's progress.

As a scientist, I am greatly excited about the prospects for progress on our understanding of human-level intelligence, one of the fundamental mysteries of our universe. The road is long, but perhaps we can once again see the way ahead.

Appendix A

Glossary

I have trouble keeping my terminology and definitions fixed as I go through a document—I forget the terminology I've invented before and reinvent it, or as my understanding of my topic becomes clearer the meanings of words change and I forget to change the old usages. This list is to help me keep myself consistent, as well as to help the reader.

Formatting

- *italics* indicate the introduction of a mathematical variable or technical term, which will be given a precise definition. Occasionally, it will be used for emphasis or foreign language—these cases will be clear in context.
- **bold face** indicates a variable name, defined value in a program, or title of a list entry.

Definitions

- **allocated (coder state):** the state of a *coder* in a *unidirectional link* when it is aligned with a complement and has been provided to the link's client for use in building *messages*. This state only exists in the *speaker*.
- **bidirectional link:** a *device* that connects two *specialists* and allows *messages* to be simultaneously sent and received using the same set of *symbols* and *inflections*. Composed of two *unidirectional links* whose *coders* are paired up via a *distributed map* as they become *mature*.
- **cable-head:** a *device* in the *unidirectional link* that bridges between random links to the *symbol coders* and a *communication path* in the *channel*.
- **channel:** a thick, twisted bundle of *communication paths* over which *specialists* send *messages* to one another.

- **coder, symbol coder, inflection coder:** a *device* that encodes or decodes *signals* in a *unidirectional link*. There are two varieties: a coder for *symbols* designates a set of communication paths, a coder for *inflections* stores a pattern of pulses.
- **codetector:** an incremental decision-making *device* that indicates whether to accept, reject, or wait on a proposal.
- **communication bootstrapping:** a phenomenon exhibited when a network of *devices* use shared experiences to reach agreement on a system of *signals* for communicating with one another.
- **communication path:** a connection between two *devices* that can carry information.
- **competition:** a *device* that breaks symmetry in a set, selecting one element at a time as the current winner.
- **conditions:** the external *environment* affecting a *device.*
- **configuration:** a particular choice of values for the *configuration parameters* of a *device.*
- **configuration parameters:** a set of adjustable values controlling the behavior of a *device.*
- **configuration policy:** a description of how to obtain *desirable behavior* from a *device* given a range of *conditions.*
- **cost:** the asymptotic complexity of a *device* with regards to time and space for *development* and mature operation, plus expected types of variation and error.
- **desirable behavior:** criteria for determining whether a *device's* actions are appropriate.
- **development:** the growth of a biological system, including the construction of computational hardware.
- **developmental cost:** the complexity of encoding blueprints for a *device*, the time it takes to grow a *device* from those blueprints, and the structural variation expected in a *device* so produced.
- **device:** a part with a well-characterized interface and behavior.
- **dither (codetector behavior):** a major behavior phase for a *codetector*, in which it is slow to make any decision.
- **disabled (coder state):** the state of a *coder* in a *unidirectional link* when it is not in use.
- **distributed focus:** a *device* that allows a network of *specialists* to balance the goals of consensus on a set of *topics* and allowing each participant an equal chance to quickly steer the consensus.

- **distributed map:** a *device* that makes one-to-one connections between elements of two sets, usable either as a dynamic map or as a reliable multi-path bus.
- **dossier:** analysis and experimental surveys of a *device's* behavior.
- **environment:** the context in which a *device* acts.
- **example:** a unit of evidence for or against a proposal.
- **failure simplification:** a method for limiting the impact of *device* misbehavior by selecting those modes of misbehavior that are easiest to understand and cope with.
- **fast accept (codetector behavior):** a major behavior phase for a *codetector*, in which it decides **accept** quickly and firmly.
- **fast convergence (distributed focus behavior):** a major behavior phase for *distributed focus*, in which dominant *topics* emerge quickly.
- **fast reject (codetector behavior):** a major behavior phase for a *codetector*, in which it decides **reject** quickly and firmly.
- **feature:** an elementary description of an *object* (e.g. a color, a degree of loudness, a category).
- **focus, focus of attention:** a selection of a few *objects* in the *model* of a *specialist*.
- **immature (coder state):** the state of a *coder* in a *unidirectional link* when it is attempting to align with a complement.
- **incremental interval-based example segmentation, IIES:** a *device* that detects examples of predictive *relations* between two *message* elements.
- **inflection:** a *signal* carried on a *symbol* that ties *features* together into *objects* and *relations*.
- **intelligence:** the broad competence and flexibility exhibited by humans, or any system exhibiting these qualities.
- **interface specification:** the relationship between a *device* and its *environment*.
- **learning by learning to communicate:** disagreement on the interpretation of *signals* that captures dynamics of a system's environment.
- **limiting conditions:** a description of the *conditions* under which a *device* is likely to misbehave.
- **listener:** the receiving side of a *unidirectional link*.
- **low-level equality:** a test for whether *objects* in different *specialists* are the same *topic* that does not depend on the ability of to interpret *messages*.

- **mature (coder state):** the state of a *coder* in a *unidirectional link* when it is aligned with a complement. In the *speaker*, this state means it is ready to be *allocated.*
- **mechanism:** how a *device* implements its *interface specification.*
- **message:** a burst of communication sent by a *specialist* to describe its *working set* to its *partner.*
- **model:** a *specialist's* description of its *environment* in terms of *objects* and *relations.*
- **object:** a persistent bundle of sensory features (e.g. a sound source), represented as an element of a part's model with *features* and *relations* to other objects.
- **observations:** the stream of sensory input provided to a *specialist.*
- **oscillate (codetector behavior):** a major behavior phase for a *codetector*, in which it rapidly changes decision.
- **partner:** a *specialist's* peers, with whom it exchanges *messages.*
- **privilege:** an occasional random assertion of dominance used by *distributed focus* to prevent *thrashing* among competing *topics.*
- **random bipartite graph:** a device connecting two sets such that each element in one set connects to k random elements in the other.
- **range of behavior:** the set of observable actions that a *device* may exhibit.
- **reflexes:** a built-in *device* peculiar to each specialist for converting its *observations* into requests for *focus* on potentially interesting *topics.*
- **relation, predictive relation, time relation:** a descriptor of how two elements relate to one another. Relations in the *model* connect pairs of *objects*, time relations connect intervals, and predictive relations connect signals to expectations about how the *model* will change.
- **relation map:** a pairwise map of *predictive relations* between incoming and outgoing signals. Each *signal map* contains one for symbols and one for inflections.
- **shared focus:** a *device* that attempts to ensure that a *specialist* and its *partner* are sending *messages* about closely related *topics.*
- **signal:** a message element: a *symbol* or an *inflection.*
- **signal map:** a *device* that translates between the *working set* of a *specialist* and *messages* exchanged with its *partner.*
- **speaker:** the sending side of a *unidirectional link.*
- **specialist:** one of the major peer *devices* that cooperate to produce *intelligence*, such as vision, hearing, or motor.

- **symbol:** an encoding of a *feature* as a sparse set of *communication paths* in the *channel.*
- **thrashing (distributed focus behavior):** a major behavior phase for *distributed focus*, in which a set of competing *topics* continually sweep through the network of *specialists*, replacing one another.
- **throttle:** a *device* that fairly filters several streams of events (such as *topic* requests input to *distributed focus*) down to a single, rate-limited stream of events.
- **topic:** objects in different *specialists* that are judged equivalent, such that they form a *shared focus.*
- **unidirectional link:** a *device* that transmits *messages* composed of *symbols* and *inflections* from a *speaker* to a *listener.*
- **usage specification:** a description of how a *device* should be configured and used in order to minimize its misbehavior.
- **v1.0, communication bootstrapping v1.0:** The architecture from the original work on *communication bootstrapping* in [4] and [3].
- **working set:** the subset of its *model* that a *specialist* communicates to its *partners.*

Appendix B

Experimental Data

This chapter contains listings of the experimental data referenced elsewhere in this dissertation.

B.1 IIES Learning

These are the results of learning relations directly between flattened sense inputs from the simulator, used in Section 7.2.4.

When relations are shown between a pair of features **A** and **B**, they are abbreviated with single characters: **Q** is **EQUAL**, **U** is **SUBCLASS**, **S** is **SEQUENCE**, **C** is **CAUSE**, **E** is **ENABLE**, **D** is **DISABLE**. Upper case means the relation goes from **A** to **B**, and lower case means the relation goes from **B** to **A**. For example, **(CUCKOO DONTWALK ”eD”)** is short for **(DISABLE CUCKOO DONTWALK)** and **(ENABLE DONTWALK CUCKOO)**.

83 minutes, starting at noon, sampling every 0.5 seconds

```
Video features:
 Always: SIDEWALK BUILDING WALKBOX POLE LIGHT REDLIGHT YELLOWLIGHT GREENLIGHT
 GRASS ROAD SKY CROSSWALK HOUSE OFFICE ABOVE BELOW LEFT RIGHT FRONT BACK
 CONTACT RED YELLOW GREEN DARK BRIGHT 0D 8D 16D 24D 32D 40D 48D 56D
 Sometimes: WALKLITE DONTWALK CAR PERSON ADULT CHILD SEDAN TRUCK VAN SUV
 AMBULANCE TOWTRUCK CYAN BLUE MAGENTA R UR U UL L DL D DR F B
 Never: BLUELIGHT POLICE 64D 72D 80D
Audio features:
 Always:
 Sometimes: CUCKOO IDLE DRIVE HONK TALK STEPS YELL SIREN ENGINE VOICE F FL L
 BL BR R FR 40 50 60 70 80 90 100
 Never: BRAKES CRASH SCREAM B
Video interval counts: BLUE 522 CYAN 499 F 333 L 322 MAGENTA 305 R 304 B 275
 CHILD 182 DONTWALK 155 SEDAN 125 PERSON 88 VAN 72 ADULT 71 TRUCK 67 SUV 63
 CAR 63 WALKLITE 23 U 12 DL 8 UR 8 UL 4 DR 3 D 3 TOWTRUCK 3 AMBULANCE 3 56D 1
 48D 1 40D 1 32D 1 24D 1 16D 1 8D 1 0D 1 BRIGHT 1 DARK 1 GREEN 1 YELLOW 1 RED 1
 CONTACT 1 BACK 1 FRONT 1 RIGHT 1 LEFT 1 BELOW 1 ABOVE 1 OFFICE 1 HOUSE 1
 CROSSWALK 1 SKY 1 ROAD 1 GRASS 1 GREENLIGHT 1 YELLOWLIGHT 1 REDLIGHT 1 LIGHT 1
 POLE 1 WALKBOX 1 BUILDING 1 SIDEWALK 1 80D 0 72D 0 64D 0 POLICE 0 BLUELIGHT 0
```

```
Audio interval counts: 70 428 L 182 R 180 BL 121 BR 118 FR 115 VOICE 107
 TALK 106 IDLE 102 DRIVE 75 60 68 F 52 FL 47 CUCKOO 23 50 17 ENGINE 9 100 4
 90 4 40 3 SIREN 3 STEPS 2 80 1 YELL 1 HONK 1 B 0 SCREAM 0 CRASH 0 BRAKES 0
Results:
((CUCKOO WALKLITE "QUuEe") (CUCKOO DONTWALK "eD") (CUCKOO CAR "ue")
 (CUCKOO PERSON "ue") (CUCKOO ADULT "ue") (CUCKOO SEDAN "ue")
 (CUCKOO MAGENTA "e") (IDLE WALKLITE "UE") (IDLE CAR "ue") (IDLE BLUE "D")
 (IDLE U "UE") (IDLE L "D") (DRIVE WALKLITE "E") (DRIVE CAR "E")
 (DRIVE SEDAN "UE") (DRIVE TRUCK "UE") (DRIVE VAN "UE") (DRIVE SUV "UE")
 (DRIVE CYAN "E") (DRIVE BLUE "E") (DRIVE MAGENTA "E") (DRIVE R "UE")
 (DRIVE L "QUE") (DRIVE F "QUE") (DRIVE B "UE") (F WALKLITE "UE")
 (F CAR "Que") (F SEDAN "e") (F TRUCK "U") (F VAN "U") (F SUV "UE")
 (F CYAN "E") (F BLUE "E") (F R "QU") (F L "Q") (F B "Q") (FL CAR "Q")
 (FL SEDAN "Q") (FL SUV "E") (FL R "QE") (FL B "Q") (L PERSON "e")
 (L ADULT "e") (L CHILD "e") (BR CAR "u") (BR PERSON "e") (BR ADULT "e")
 (R CAR "u") (R PERSON "e") (R ADULT "e") (R CHILD "e") (R CYAN "e") (R R "e")
 (FR WALKLITE "UE") (FR CAR "QSCd") (FR SEDAN "U") (FR TRUCK "UE")
 (FR VAN "UE") (FR SUV "UE") (FR CYAN "E") (FR BLUE "E") (FR MAGENTA "E")
 (FR U "UE") (FR L "UE") (FR F "QUE") (FR B "UE") (60 WALKLITE "UE")
 (60 CAR "E") (60 SEDAN "UE") (60 TRUCK "UE") (60 VAN "UE") (60 SUV "UE")
 (60 CYAN "E") (60 BLUE "E") (60 MAGENTA "E") (60 R "QUE") (60 U "UE")
 (60 L "QUE") (60 F "QUE") (60 B "QUE") (70 WALKLITE "UEe") (70 DONTWALK "ue")
 (70 CAR "QSCud") (70 PERSON "e") (70 ADULT "e") (70 CHILD "e")
 (70 TRUCK "Ed") (70 VAN "Ed") (70 SUV "Ed") (70 BLUE "E") (70 B "Ed"))
```

83 minutes, starting at noon, sampling every 1.0 seconds

```
Video features:
 Always: SIDEWALK BUILDING WALKBOX POLE LIGHT REDLIGHT YELLOWLIGHT GREENLIGHT
 GRASS ROAD SKY CROSSWALK HOUSE OFFICE ABOVE BELOW LEFT RIGHT FRONT BACK
 CONTACT RED YELLOW GREEN DARK BRIGHT 0D 8D 16D 24D 32D 40D 48D 56D
 Sometimes: WALKLITE DONTWALK CAR PERSON ADULT CHILD SEDAN TRUCK VAN SUV
 AMBULANCE TOWTRUCK CYAN BLUE MAGENTA R UR U UL L DL D DR F B
 Never: BLUELIGHT POLICE 64D 72D 80D
Audio features:
 Always:
 Sometimes: CUCKOO IDLE DRIVE HONK TALK STEPS YELL SIREN ENGINE VOICE F FL L
 BL BR R FR 40 50 60 70 80 90 100
 Never: BRAKES CRASH SCREAM B
Video interval counts: BLUE 310 F 304 CYAN 287 L 271 R 271 B 238 MAGENTA 186
 DONTWALK 151 CHILD 121 SEDAN 112 VAN 68 TRUCK 62 CAR 62 SUV 60 PERSON 56
 ADULT 47 WALKLITE 23 U 10 UR 4 UL 3 TOWTRUCK 3 AMBULANCE 3 DR 2 D 2 DL 2 56D 1
 48D 1 40D 1 32D 1 24D 1 16D 1 8D 1 0D 1 BRIGHT 1 DARK 1 GREEN 1 YELLOW 1 RED 1
 CONTACT 1 BACK 1 FRONT 1 RIGHT 1 LEFT 1 BELOW 1 ABOVE 1 OFFICE 1 HOUSE 1
 CROSSWALK 1 SKY 1 ROAD 1 GRASS 1 GREENLIGHT 1 YELLOWLIGHT 1 REDLIGHT 1 LIGHT 1
 POLE 1 WALKBOX 1 BUILDING 1 SIDEWALK 1 80D 0 72D 0 64D 0 POLICE 0 BLUELIGHT 0
Audio interval counts: 70 409 L 181 R 174 BL 119 BR 117 FR 105 IDLE 93 VOICE 88
 TALK 87 DRIVE 69 60 63 F 49 FL 42 CUCKOO 23 50 17 ENGINE 9 100 4 90 4 40 3
 SIREN 3 STEPS 2 80 1 YELL 1 HONK 1 B 0 SCREAM 0 CRASH 0 BRAKES 0
Results:
((CUCKOO WALKLITE "QUuEe") (CUCKOO DONTWALK "eD") (CUCKOO CAR "ue")
 (CUCKOO PERSON "ue") (CUCKOO ADULT "ue") (CUCKOO SEDAN "ue")
 (CUCKOO MAGENTA "e") (IDLE WALKLITE "UE") (IDLE CAR "ue") (IDLE SUV "E")
 (IDLE U "UE") (DRIVE WALKLITE "E") (DRIVE CAR "E") (DRIVE SEDAN "UE")
 (DRIVE TRUCK "UE") (DRIVE VAN "UE") (DRIVE SUV "UE") (DRIVE CYAN "E")
 (DRIVE BLUE "UE") (DRIVE MAGENTA "UE") (DRIVE R "UE") (DRIVE L "QUE")
 (DRIVE F "QUE") (DRIVE B "QUE") (F WALKLITE "UE") (F CAR "Que") (F SEDAN "Q")
 (F TRUCK "UE") (F VAN "UE") (F SUV "UE") (F CYAN "UE") (F R "Q") (F L "Q")
 (F B "Q") (FL CAR "Q") (FL SEDAN "Q") (FL SUV "E") (FL CYAN "E") (FL R "QE")
 (FL B "Q") (L PERSON "e") (L ADULT "e") (L MAGENTA "e") (L F "e")
 (BL DONTWALK "u") (BR DONTWALK "e") (BR CAR "u") (BR ADULT "e") (R CAR "u")
 (R PERSON "e") (R ADULT "e") (R CHILD "e") (R R "e") (FR WALKLITE "UE")
 (FR CAR "QSCu") (FR SEDAN "UE") (FR TRUCK "UE") (FR VAN "UE") (FR SUV "UE")
 (FR CYAN "E") (FR BLUE "E") (FR MAGENTA "E") (FR R "QU") (FR U "UE")
 (FR L "QUE") (FR F "QUE") (FR B "UE") (60 WALKLITE "UE") (60 DONTWALK "e")
 (60 CAR "E") (60 SEDAN "UE") (60 TRUCK "UE") (60 VAN "UE") (60 SUV "UE")
 (60 CYAN "E") (60 BLUE "UE") (60 MAGENTA "UE") (60 R "UE") (60 L "QUE")
 (60 F "QUE") (60 B "QUE") (70 WALKLITE "UEe") (70 DONTWALK "ue")
 (70 CAR "Qud") (70 PERSON "e") (70 ADULT "e") (70 CHILD "e") (70 TRUCK "Ed")
 (70 VAN "Ed") (70 SUV "Ed") (70 BLUE "E") (70 F "e") (70 B "Ed"))
```

83 minutes, starting at noon, sampling every 1.5 seconds

```
Video features:
 Always: SIDEWALK BUILDING WALKBOX POLE LIGHT REDLIGHT YELLOWLIGHT GREENLIGHT
 GRASS ROAD SKY CROSSWALK HOUSE OFFICE ABOVE BELOW LEFT RIGHT FRONT BACK
 CONTACT RED YELLOW GREEN DARK BRIGHT 0D 8D 16D 24D 32D 40D 48D 56D
 Sometimes: WALKLITE DONTWALK CAR PERSON ADULT CHILD SEDAN TRUCK VAN SUV
 AMBULANCE TOWTRUCK CYAN BLUE MAGENTA R UR U L DL D DR F B
 Never: BLUELIGHT POLICE 64D 72D 80D UL
Audio features:
 Always:
 Sometimes: CUCKOO IDLE DRIVE HONK TALK STEPS YELL SIREN ENGINE VOICE F FL L
 BL BR R FR 40 50 60 70 80 90 100
 Never: BRAKES CRASH SCREAM B
Video interval counts: F 290 R 256 L 241 B 222 BLUE 218 CYAN 198 MAGENTA 136
 SEDAN 103 CHILD 95 VAN 70 DONTWALK 65 SUV 62 TRUCK 61 CAR 61 PERSON 42
 ADULT 34 WALKLITE 23 U 6 UR 3 TOWTRUCK 3 AMBULANCE 3 DR 2 D 2 DL 2 56D 1 48D 1
 40D 1 32D 1 24D 1 16D 1 8D 1 0D 1 BRIGHT 1 DARK 1 GREEN 1 YELLOW 1 RED 1
 CONTACT 1 BACK 1 FRONT 1 RIGHT 1 LEFT 1 BELOW 1 ABOVE 1 OFFICE 1 HOUSE 1
 CROSSWALK 1 SKY 1 ROAD 1 GRASS 1 GREENLIGHT 1 YELLOWLIGHT 1 REDLIGHT 1 LIGHT 1
 POLE 1 WALKBOX 1 BUILDING 1 SIDEWALK 1 UL 0 80D 0 72D 0 64D 0 POLICE 0
 BLUELIGHT 0
Audio interval counts: 70 399 L 175 R 171 BL 116 BR 113 FR 89 IDLE 88 VOICE 73
 TALK 72 DRIVE 65 60 57 F 49 FL 38 CUCKOO 23 50 16 ENGINE 8 100 4 90 4 40 3
 SIREN 3 STEPS 2 80 1 YELL 1 HONK 1 B 0 SCREAM 0 CRASH 0 BRAKES 0
Results:
((CUCKOO WALKLITE "QUuEe") (CUCKOO DONTWALK "eD") (CUCKOO CAR "ue")
 (CUCKOO PERSON "ue") (CUCKOO ADULT "ue") (CUCKOO SEDAN "ue")
 (CUCKOO R "e") (IDLE WALKLITE "UE") (IDLE CAR "ue") (IDLE SEDAN "u")
 (IDLE TRUCK "E") (IDLE F "D") (DRIVE WALKLITE "E") (DRIVE CAR "E")
 (DRIVE SEDAN "UE") (DRIVE TRUCK "UE") (DRIVE VAN "UE") (DRIVE SUV "UE")
 (DRIVE CYAN "E") (DRIVE BLUE "UE") (DRIVE MAGENTA "E") (DRIVE R "QUE")
 (DRIVE L "QUE") (DRIVE F "QUE") (DRIVE B "QUE") (TALK CYAN "E")
 (VOICE CYAN "E") (F WALKLITE "UE") (F CAR "Que") (F SEDAN "Qe")
 (F TRUCK "UE") (F VAN "UE") (F SUV "UE") (F CYAN "UE") (F BLUE "E") (F R "Q")
 (F L "Q") (F B "Q") (FL CAR "Q") (FL SEDAN "Q") (FL SUV "UE") (FL CYAN "UE")
 (FL R "E") (FL B "Q") (L L "e") (L F "e") (BL DONTWALK "e") (BR CAR "u")
 (R CAR "Que") (R PERSON "e") (R ADULT "e") (R CHILD "e") (R CYAN "e")
 (R R "e") (FR WALKLITE "UE") (FR CAR "QuE") (FR SEDAN "QE") (FR TRUCK "UE")
 (FR VAN "UE") (FR SUV "UE") (FR CYAN "E") (FR BLUE "E") (FR R "Q")
 (FR L "QUE") (FR F "QUE") (FR B "UE") (60 WALKLITE "UE") (60 CAR "E")
 (60 SEDAN "UE") (60 TRUCK "UE") (60 VAN "UE") (60 SUV "UE") (60 CYAN "E")
 (60 BLUE "UE") (60 MAGENTA "E") (60 R "QUE") (60 L "QUE") (60 F "QUE")
 (60 B "QUE") (70 WALKLITE "UE") (70 DONTWALK "ue") (70 CAR "Qud")
 (70 SEDAN "d") (70 TRUCK "Ed") (70 VAN "Ed") (70 SUV "Ed") (70 BLUE "E")
 (70 R "Ee") (70 F "e") (70 B "UEd"))
```

83 minutes, starting at noon, sampling every 2.0 seconds

```
Video features:
 Always: SIDEWALK BUILDING WALKBOX POLE LIGHT REDLIGHT YELLOWLIGHT GREENLIGHT
 GRASS ROAD SKY CROSSWALK HOUSE OFFICE ABOVE BELOW LEFT RIGHT FRONT BACK
 CONTACT RED YELLOW GREEN DARK BRIGHT 0D 8D 16D 24D 32D 40D 48D 56D
 Sometimes: WALKLITE DONTWALK CAR PERSON ADULT CHILD SEDAN TRUCK VAN SUV
 AMBULANCE TOWTRUCK CYAN BLUE MAGENTA R UR U UL L DL DR F B
 Never: BLUELIGHT POLICE 64D 72D 80D D
Audio features:
 Always:
 Sometimes: CUCKOO IDLE DRIVE TALK STEPS YELL SIREN ENGINE VOICE F FL L BL BR
 R FR 40 50 60 70 80 90 100
 Never: HONK BRAKES CRASH SCREAM B
Video interval counts: F 281 R 242 L 228 B 210 BLUE 173 CYAN 158 MAGENTA 112
```

```
 SEDAN 104 CHILD 71 VAN 66 TRUCK 59 CAR 59 SUV 58 PERSON 38 ADULT 29
 DONTWALK 25 WALKLITE 23 TOWTRUCK 3 AMBULANCE 3 DR 2 U 2 DL 1 UL 1 UR 1 56D 1
 48D 1 40D 1 32D 1 24D 1 16D 1 8D 1 0D 1 BRIGHT 1 DARK 1 GREEN 1 YELLOW 1 RED 1
 CONTACT 1 BACK 1 FRONT 1 RIGHT 1 LEFT 1 BELOW 1 ABOVE 1 OFFICE 1 HOUSE 1
 CROSSWALK 1 SKY 1 ROAD 1 GRASS 1 GREENLIGHT 1 YELLOWLIGHT 1 REDLIGHT 1 LIGHT 1
 POLE 1 WALKBOX 1 BUILDING 1 SIDEWALK 1 D 0 80D 0 72D 0 64D 0 POLICE 0
 BLUELIGHT 0
Audio interval counts: 70 367 L 176 R 168 BL 116 BR 113 IDLE 85 FR 74 VOICE 64
 TALK 63 DRIVE 58 60 50 F 46 FL 32 CUCKOO 23 50 15 ENGINE 7 100 3 90 3 40 3
 SIREN 3 STEPS 2 80 1 YELL 1 B 0 SCREAM 0 CRASH 0 BRAKES 0 HONK 0
Results:
((CUCKOO WALKLITE "QUuEe") (CUCKOO DONTWALK "sceD") (CUCKOO CAR "ue")
 (CUCKOO PERSON "ue") (CUCKOO ADULT "ue") (CUCKOO SEDAN "ue")
 (IDLE WALKLITE "UE") (IDLE CAR "ue") (IDLE TRUCK "E") (IDLE VAN "E")
 (IDLE SUV "E") (IDLE CYAN "E") (IDLE F "D") (IDLE B "E")
 (DRIVE WALKLITE "UE") (DRIVE DONTWALK "Q") (DRIVE CAR "E") (DRIVE SEDAN "UE")
 (DRIVE TRUCK "UE") (DRIVE VAN "UE") (DRIVE SUV "UE") (DRIVE CYAN "E")
 (DRIVE BLUE "UE") (DRIVE MAGENTA "UE") (DRIVE R "QUE") (DRIVE L "QUE")
 (DRIVE F "QUE") (DRIVE B "QUE") (VOICE PERSON "e") (F WALKLITE "UE")
 (F CAR "Que") (F SEDAN "Qce") (F TRUCK "UE") (F VAN "UE") (F SUV "UE")
 (F CYAN "UE") (F BLUE "E") (F R "Q") (F L "Q") (F B "QU") (FL VAN "E")
 (FL SUV "E") (FL CYAN "UE") (FL R "QE") (FL B "Q") (L CAR "Qu") (L L "e")
 (L F "e") (BL CAR "e") (BR CAR "u") (R CAR "Qu") (R ADULT "e") (R CHILD "e")
 (R MAGENTA "e") (R R "e") (FR WALKLITE "UE") (FR CAR "QuE") (FR SEDAN "QUE")
 (FR TRUCK "UE") (FR VAN "UE") (FR SUV "UE") (FR CYAN "E") (FR BLUE "UE")
 (FR MAGENTA "UE") (FR R "E") (FR L "QUE") (FR F "QUE") (FR B "QUE")
 (60 WALKLITE "UE") (60 CAR "E") (60 SEDAN "UE") (60 TRUCK "UE") (60 VAN "UE")
 (60 SUV "UE") (60 CYAN "E") (60 BLUE "UE") (60 MAGENTA "UE") (60 R "QUE")
 (60 L "QUE") (60 F "QUE") (60 B "QUE") (70 WALKLITE "UE") (70 DONTWALK "Que")
 (70 CAR "Qud") (70 PERSON "e") (70 CHILD "e") (70 TRUCK "E") (70 VAN "Ed")
 (70 SUV "Ed") (70 BLUE "E") (70 R "e") (70 L "e") (70 F "e") (70 B "UEe"))
```

83 minutes, starting at midnight, sampling every 0.5 seconds

```
Video features:
 Always: SIDEWALK BUILDING WALKBOX DONTWALK POLE LIGHT REDLIGHT YELLOWLIGHT
 GREENLIGHT GRASS ROAD SKY CROSSWALK HOUSE OFFICE ABOVE BELOW LEFT RIGHT FRONT
 BACK CONTACT RED YELLOW GREEN DARK BRIGHT 0D 8D 16D 24D 32D 40D 48D 56D
 Sometimes: BLUELIGHT CAR PERSON ADULT CHILD SEDAN TRUCK VAN SUV AMBULANCE
 POLICE CYAN BLUE MAGENTA R UR U UL L DR F B
 Never: WALKLITE TOWTRUCK 64D 72D 80D DL D
Audio features:
 Always:
 Sometimes: IDLE DRIVE STEPS SIREN ENGINE F FL L BL BR R FR 40 50 60 70 90 100
 Never: CUCKOO HONK BRAKES CRASH TALK YELL SCREAM VOICE B 80
Video interval counts: L 81 BLUE 77 F 72 CAR 72 R 63 B 56 SEDAN 48 CYAN 45
 MAGENTA 37 PERSON 35 CHILD 28 VAN 12 TRUCK 10 ADULT 7 SUV 6 DR 2 UL 1 U 1 UR 1
 56D 1 48D 1 40D 1 32D 1 24D 1 16D 1 8D 1 0D 1 BRIGHT 1 DARK 1 GREEN 1 YELLOW 1
 RED 1 CONTACT 1 BACK 1 FRONT 1 RIGHT 1 LEFT 1 BELOW 1 ABOVE 1 POLICE 1
 AMBULANCE 1 OFFICE 1 HOUSE 1 CROSSWALK 1 SKY 1 ROAD 1 GRASS 1 BLUELIGHT 1
 GREENLIGHT 1 YELLOWLIGHT 1 REDLIGHT 1 LIGHT 1 POLE 1 DONTWALK 1 WALKBOX 1
 BUILDING 1 SIDEWALK 1 D 0 DL 0 80D 0 72D 0 64D 0 TOWTRUCK 0 WALKLITE 0
Audio interval counts: 50 158 70 99 60 94 DRIVE 93 FR 82 FL 77 F 74 ENGINE 55
 R 45 BR 43 L 39 IDLE 39 BL 35 40 31 STEPS 12 100 1 90 1 SIREN 1 80 0 B 0
 VOICE 0 SCREAM 0 YELL 0 TALK 0 CRASH 0 BRAKES 0 HONK 0 CUCKOO 0
Results:
((IDLE CAR "ue") (IDLE SEDAN "e") (IDLE L "eD") (DRIVE CAR "QSCUE")
 (DRIVE SEDAN "UE") (DRIVE VAN "E") (DRIVE CYAN "E") (DRIVE BLUE "UE")
 (DRIVE MAGENTA "E") (DRIVE R "UE") (DRIVE L "UE") (DRIVE F "UE")
 (DRIVE B "UE") (ENGINE CAR "QUE") (ENGINE SEDAN "UE") (ENGINE VAN "UE")
 (ENGINE CYAN "E") (ENGINE BLUE "UE") (ENGINE MAGENTA "UE") (ENGINE R "UE")
 (ENGINE L "UE") (ENGINE F "UE") (ENGINE B "UE") (F CAR "Que") (F SEDAN "e")
 (F L "e") (F F "ce") (F B "e") (FL CAR "Q") (FL R "E") (FL L "eD") (L L "e")
 (R R "eD") (R F "eD") (FR CAR "Q") (FR BLUE "E") (FR R "eD") (FR L "E")
 (40 CAR "u") (50 R "E") (50 F "Ed") (50 B "D") (60 CAR "QSCUE")
 (60 SEDAN "UE") (60 VAN "E") (60 CYAN "E") (60 BLUE "UE") (60 MAGENTA "E")
 (60 R "UE") (60 L "UE") (60 F "UE") (60 B "UE") (70 CAR "Q") (70 F "eD")
 (70 B "Ed"))
```

83 minutes, starting at 8am, sampling every 0.5 seconds

```
Video features:
 Always: SIDEWALK BUILDING WALKBOX POLE LIGHT REDLIGHT YELLOWLIGHT GREENLIGHT
 GRASS ROAD SKY CROSSWALK HOUSE OFFICE ABOVE BELOW LEFT RIGHT FRONT BACK
 CONTACT RED YELLOW GREEN DARK BRIGHT 0D 8D 16D 24D 32D 40D 48D 56D
 Sometimes: WALKLITE DONTWALK BLUELIGHT CAR PERSON ADULT CHILD SEDAN TRUCK VAN
 SUV AMBULANCE TOWTRUCK POLICE CYAN BLUE MAGENTA R UR U UL L DL D DR F B
 Never: 64D 72D 80D
Audio features:
 Always:
 Sometimes: CUCKOO IDLE DRIVE HONK BRAKES TALK STEPS YELL SIREN ENGINE VOICE
 F FL L BL BR R FR 40 50 60 70 80 90 100
 Never: CRASH SCREAM B
Video interval counts: CYAN 494 F 391 MAGENTA 370 L 342 R 334 B 310 BLUE 287
 DONTWALK 184 CHILD 150 ADULT 127 PERSON 109 SEDAN 92 SUV 85 TRUCK 83 VAN 77
 WALKLITE 27 CAR 23 U 14 UR 13 UL 10 DR 5 D 5 DL 5 BLUELIGHT 4 POLICE 3
 TOWTRUCK 3 AMBULANCE 2 56D 1 48D 1 40D 1 32D 1 24D 1 16D 1 8D 1 0D 1 BRIGHT 1
 DARK 1 GREEN 1 YELLOW 1 RED 1 CONTACT 1 BACK 1 FRONT 1 RIGHT 1 LEFT 1 BELOW 1
 ABOVE 1 OFFICE 1 HOUSE 1 CROSSWALK 1 SKY 1 ROAD 1 GRASS 1 GREENLIGHT 1
 YELLOWLIGHT 1 REDLIGHT 1 LIGHT 1 POLE 1 WALKBOX 1 BUILDING 1 SIDEWALK 1 80D 0
 72D 0 64D 0
Audio interval counts: 70 436 VOICE 230 TALK 229 R 207 L 175 BR 119 BL 114
 IDLE 86 FR 58 60 40 DRIVE 34 F 32 FL 30 CUCKOO 27 90 11 100 10 50 10 HONK 7
 80 3 40 3 SIREN 3 YELL 2 ENGINE 1 STEPS 1 BRAKES 1 B 0 SCREAM 0 CRASH 0
Results:
((CUCKOO WALKLITE "QUuEe") (CUCKOO DONTWALK "eD") (CUCKOO CAR "ue")
 (CUCKOO PERSON "ue") (CUCKOO ADULT "ue") (CUCKOO SEDAN "ue")
 (CUCKOO SUV "e") (CUCKOO CYAN "e") (CUCKOO BLUE "ue") (CUCKOO L "u")
 (IDLE WALKLITE "UE") (IDLE CAR "Que") (IDLE SEDAN "Qu") (IDLE TRUCK "E")
 (IDLE SUV "U") (IDLE CYAN "E") (IDLE MAGENTA "E") (IDLE L "D")
 (DRIVE WALKLITE "UE") (DRIVE TRUCK "UE") (DRIVE VAN "UE") (DRIVE SUV "QUE")
 (DRIVE CYAN "QU") (DRIVE MAGENTA "E") (DRIVE R "U") (DRIVE L "QU")
 (DRIVE F "QUE") (DRIVE B "QUE") (TALK TRUCK "d") (VOICE TRUCK "d")
 (F TRUCK "U") (F SUV "UE") (F CYAN "E") (F MAGENTA "E") (F R "Q")
 (FL SUV "UE") (FL CYAN "E") (FL R "E") (FL F "Q") (L PERSON "e")
 (L ADULT "e") (L BLUE "e") (BL SEDAN "Q") (BR SEDAN "e") (R ADULT "e")
 (R SEDAN "Que") (R R "e") (R U "E") (R F "e") (FR TRUCK "UE") (FR VAN "UE")
 (FR SUV "UE") (FR MAGENTA "E") (FR L "U") (FR F "U") (60 WALKLITE "UE")
 (60 TRUCK "UE") (60 VAN "UE") (60 SUV "UE") (60 CYAN "U") (60 MAGENTA "E")
 (60 R "U") (60 L "UE") (60 F "UE") (60 B "UE") (70 WALKLITE "UE")
 (70 DONTWALK "ue") (70 CAR "Q") (70 TRUCK "UE") (70 VAN "Ed") (70 SUV "E")
 (70 CYAN "E") (70 MAGENTA "E") (70 B "QE"))
```

83 minutes, starting at 3pm, sampling every 0.5 seconds

```
Video features:
 Always: SIDEWALK BUILDING WALKBOX POLE LIGHT REDLIGHT YELLOWLIGHT GREENLIGHT
 GRASS ROAD SKY CROSSWALK HOUSE OFFICE ABOVE BELOW LEFT RIGHT FRONT BACK
 CONTACT RED YELLOW GREEN DARK BRIGHT 0D 8D 16D 24D 32D 40D 48D 56D
 Sometimes: WALKLITE DONTWALK BLUELIGHT CAR PERSON ADULT CHILD SEDAN TRUCK VAN
 SUV AMBULANCE TOWTRUCK POLICE CYAN BLUE MAGENTA R UR U UL L DL D DR F B
 Never: 64D 72D 80D
Audio features:
 Always:
 Sometimes: CUCKOO IDLE DRIVE HONK BRAKES CRASH TALK STEPS YELL SIREN ENGINE
 VOICE F FL L BL BR R FR 40 50 60 70 80 90 100
 Never: SCREAM B
Video interval counts: CYAN 518 BLUE 479 MAGENTA 410 L 376 F 368 R 358 B 319
 DONTWALK 234 CHILD 165 SEDAN 82 TRUCK 78 PERSON 78 SUV 65 VAN 62 ADULT 47
 WALKLITE 35 CAR 34 U 19 UR 15 DL 10 D 9 UL 9 DR 8 POLICE 7 BLUELIGHT 6
 TOWTRUCK 5 AMBULANCE 4 56D 1 48D 1 40D 1 32D 1 24D 1 16D 1 8D 1 0D 1 BRIGHT 1
 DARK 1 GREEN 1 YELLOW 1 RED 1 CONTACT 1 BACK 1 FRONT 1 RIGHT 1 LEFT 1 BELOW 1
 ABOVE 1 OFFICE 1 HOUSE 1 CROSSWALK 1 SKY 1 ROAD 1 GRASS 1 GREENLIGHT 1
 YELLOWLIGHT 1 REDLIGHT 1 LIGHT 1 POLE 1 WALKBOX 1 BUILDING 1 SIDEWALK 1 80D 0
 72D 0 64D 0
Audio interval counts: 70 414 VOICE 254 TALK 253 R 151 L 143 BL 121 BR 99
 IDLE 94 DRIVE 82 60 55 FR 51 CUCKOO 35 F 26 FL 15 50 13 90 12 100 10 SIREN 5
 HONK 5 80 4 40 3 ENGINE 3 YELL 3 STEPS 2 CRASH 1 BRAKES 1 B 0 SCREAM 0
Results:
((CUCKOO WALKLITE "QUuEe") (CUCKOO DONTWALK "eD") (CUCKOO CAR "ue")
 (CUCKOO PERSON "ue") (CUCKOO ADULT "ue") (CUCKOO CHILD "e")
 (CUCKOO SEDAN "ue") (CUCKOO MAGENTA "ue") (CUCKOO L "E")
 (CUCKOO B "E") (IDLE WALKLITE "UE") (IDLE CAR "ue") (IDLE SEDAN "e")
 (IDLE TRUCK "U") (IDLE VAN "UE") (IDLE SUV "U") (IDLE CYAN "D")
 (IDLE BLUE "D") (IDLE UR "UE") (IDLE U "E") (IDLE L "D")
 (DRIVE WALKLITE "UE") (DRIVE SEDAN "QE") (DRIVE TRUCK "UE") (DRIVE VAN "UE")
 (DRIVE SUV "UE") (DRIVE CYAN "D") (DRIVE BLUE "E") (DRIVE R "QUE")
 (DRIVE UR "E") (DRIVE U "E") (DRIVE L "QUD") (DRIVE F "QUE") (DRIVE B "QUE")
 (TALK WALKLITE "E") (TALK TRUCK "E") (TALK VAN "E") (TALK UR "E")
 (TALK U "E") (TALK B "E") (VOICE WALKLITE "E") (VOICE TRUCK "E")
 (VOICE VAN "E") (VOICE UR "E") (VOICE U "E") (VOICE B "E") (F CAR "Qe")
 (FL CYAN "E") (L DONTWALK "e") (L PERSON "e") (L ADULT "e") (L L "e")
 (BL DONTWALK "e") (BR SEDAN "e") (BR L "e") (R WALKLITE "E") (R PERSON "e")
 (R UR "E") (R U "E") (FR WALKLITE "UE") (FR TRUCK "UE") (FR VAN "E")
 (FR SUV "UE") (FR F "Q") (60 WALKLITE "UE") (60 TRUCK "UE") (60 VAN "UE")
 (60 SUV "UE") (60 BLUE "UE") (60 R "QUE") (60 U "E") (60 L "UE") (60 F "QUE")
 (60 B "UE") (70 WALKLITE "UE") (70 DONTWALK "ue") (70 CAR "Q") (70 CHILD "e")
 (70 TRUCK "E") (70 VAN "Ed") (70 SUV "Ed") (70 B "E") (100 DONTWALK "e"))
```

B.2 Signal Map Learning

These are the results of two specialists learning to communicate using flattened sense inputs from the simulator, used in Section 7.3.3. The relations between symbols in each signal map are omitted (with one exception), as they are precisely identical to that learned in the IIES test above.

83 minutes, starting at noon, sampling every 0.5 seconds

```
Audio Relations: [Omitted]
Video Relations: [Omitted]
Inflections: ((PRESENT PRESENT) (PRESENT MOTION))
             ((PRESENT PRESENT) (MOTION PRESENT))
Predictions:
T=1000 A=[C=1.57 I=0.32 U=312.0] V=[C=0.63 I=0.00 U=697.0]
T=2000 A=[C=4.02 I=8.51 U=285.0] V=[C=1.35 I=3.70 U=562.0]
T=3000 A=[C=3.08 I=13.86 U=337.0] V=[C=2.24 I=10.49 U=591.0]
T=4000 A=[C=3.12 I=14.63 U=149.0] V=[C=0.68 I=10.59 U=484.0]
T=5000 A=[C=2.72 I=11.76 U=179.0] V=[C=4.12 I=9.95 U=357.0]
T=6000 A=[C=1.87 I=13.95 U=196.0] V=[C=1.02 I=4.71 U=255.0]
T=7000 A=[C=1.03 I=6.79 U=135.0] V=[C=1.24 I=2.19 U=329.0]
T=8000 A=[C=0.88 I=12.16 U=191.0] V=[C=1.53 I=2.97 U=320.0]
T=9000 A=[C=1.26 I=16.54 U=232.0] V=[C=0.87 I=5.80 U=374.0]
T=10000 A=[C=0.95 I=24.20 U=306.0] V=[C=0.46 I=1.59 U=530.0]
Number of Relations:
(1 0 0) (2 0 0) (3 0 0) (4 0 0) (5 0 0) (6 0 0) (7 2 2) (8 2 2) (9 1 1)
(10 1 1) (11 1 1) (12 2 2) (13 4 4) (14 7 7) (15 11 11) (16 17 17) (17 20 20)
(18 20 20) (19 28 28) (20 28 28) (21 30 30) (22 38 38) (23 38 38) (24 39 39)
(25 39 39) (26 35 35) (27 40 40) (28 46 46) (29 46 46) (30 50 50) (31 49 49)
(32 56 56) (33 58 58) (34 58 58) (35 59 59) (36 59 59) (37 60 60) (38 60 60)
(39 62 62) (40 62 62) (41 63 63) (42 63 63) (43 64 64) (44 74 74) (45 80 80)
(46 81 81) (47 83 83) (48 83 83) (49 84 84) (50 84 84) (51 83 83) (52 83 83)
(53 88 88) (54 88 88) (55 89 89) (56 91 91) (57 96 96) (58 101 101) (59 96 96)
(60 96 96) (61 104 104) (62 113 113) (63 117 117) (64 123 123) (65 126 126)
(66 126 126) (67 126 126) (68 127 127) (69 127 127) (70 132 132) (71 129 129)
```

```
(72 131 131) (73 134 134) (74 131 131) (75 134 134) (76 133 133) (77 132 132)
(78 134 134) (79 134 134) (80 138 138) (81 133 133) (82 140 140) (83 142 142)
(84 144 144) (85 143 143) (86 144 144) (87 145 145) (88 148 148) (89 150 150)
(90 146 146) (91 144 144) (92 142 142) (93 142 142) (94 142 142) (95 152 152)
(96 157 157) (97 157 157) (98 158 158) (99 157 157) (100 156 156)
```

83 minutes, starting at noon, sampling every 1.0 seconds

```
Audio Relations: [Omitted]
Video Relations: [Omitted]
Inflections: ((PRESENT PRESENT) (PRESENT MOTION))
             ((PRESENT PRESENT) (MOTION PRESENT))
Predictions:
T=1000 A=[C=2.33 I=4.30 U=569.0] V=[C=4.77 I=6.79 U=981.0]
T=2000 A=[C=11.48 I=16.39 U=446.0] V=[C=5.07 I=34.35 U=786.0]
T=3000 A=[C=5.10 I=25.96 U=357.0] V=[C=6.95 I=29.42 U=488.0]
T=4000 A=[C=7.01 I=28.61 U=296.0] V=[C=2.77 I=7.88 U=552.0]
T=5000 A=[C=4.08 I=30.53 U=483.0] V=[C=2.75 I=4.09 U=693.0]
Number of Relations:
(1 0 0) (2 0 0) (3 1 1) (4 1 1) (5 2 2) (6 2 2) (7 3 3) (8 6 6) (9 17 17)
(10 21 21) (11 30 30) (12 31 31) (13 29 29) (14 37 37)  (15 45 45) (16 51 51)
(17 56 56) (18 59 59) (19 60 60) (20 64 64) (21 69 69) (22 84 84) (23 90 90)
(24 91 91) (25 97 97) (26 90 90) (27 94 94) (28 95 95) (29 108 108)
(30 106 106) (31 114 114) (32 124 124) (33 126 126) (34 130 130) (35 131 131)
(36 134 134) (37 138 138) (38 144 144) (39 146 146) (40 145 145) (41 146 146)
(42 149 149) (43 150 150) (44 150 150) (45 156 156) (46 150 150) (47 150 150)
(48 158 158) (49 162 162) (50 164 164)
```

83 minutes, starting at noon, sampling every 1.5 seconds

```
Audio Relations: [Omitted]
Video Relations: [Omitted]
Inflections: ((PRESENT PRESENT) (PRESENT MOTION))
             ((PRESENT PRESENT) (MOTION PRESENT))
Predictions:
T=1000 A=[C=6.81 I=24.43 U=861.0] V=[C=10.26 I=32.55 U=1144.0]
T=2000 A=[C=25.02 I=33.85 U=498.0] V=[C=12.36 I=76.55 U=659.0]
T=3000 A=[C=13.71 I=40.54 U=484.0] V=[C=5.47 I=17.15 U=645.0]
Number of Relations:
(1 0 0) (2 0 0) (3 1 1) (4 1 1) (5 7 7) (6 21 21) (7 23 23) (8 35 35)
(9 41 41) (10 50 50) (11 56 56) (12 56 56) (13 61 61) (14 65 65) (15 82 82)
(16 82 82) (17 86 86) (18 89 89) (19 100 100) (20 106 106) (21 110 110)
(22 121 121) (23 123 123) (24 127 127) (25 132 132) (26 127 127) (27 133 133)
(28 143 143) (29 146 146) (30 150 150) (31 151 151) (32 157 157) (33 162 162)
```

83 minutes, starting at noon, sampling every 2.0 seconds

```
Audio Relations: [Omitted]
Video Relations: [Omitted]
Inflections: ((PRESENT PRESENT) (PRESENT MOTION))
             ((PRESENT PRESENT) (MOTION PRESENT))
Predictions:
T=1000 A=[C=10.10 I=50.42 U=1004.0] V=[C=11.06 I=54.78 U=1311.0]
T=2000 A=[C=25.08 I=68.34 U=668.0] V=[C=7.20 I=72.66 U=711.0]
Number of Relations:
(1 0 0) (2 1 1) (3 1 1) (4 5 5) (5 18 18) (6 24 24) (7 29 29) (8 40 40)
(9 41 41) (10 53 53) (11 70 70) (12 79 79) (13 81 81) (14 92 92) (15 102 102)
(16 115 115) (17 121 121) (18 134 134) (19 134 134) (20 135 135) (21 146 146)
(22 149 149) (23 157 157) (24 172 172) (25 176 176)
```

83 minutes, starting at midnight, sampling every 0.5 seconds

```
Audio Relations: [Omitted]
Video Relations: [Omitted]
Inflections: ((PRESENT PRESENT) (PRESENT TYPE))
             ((PRESENT PRESENT) (TYPE PRESENT))
Predictions:
T=1000 A=[C=0.00 I=0.00 U=246.0] V=[C=0.00 I=0.00 U=132.0]
T=2000 A=[C=0.62 I=0.68 U=164.0] V=[C=0.38 I=2.20 U=112.0]
T=3000 A=[C=1.93 I=2.39 U=147.0] V=[C=1.16 I=0.71 U=70.0]
T=4000 A=[C=2.61 I=2.55 U=197.0] V=[C=1.85 I=1.25 U=76.0]
T=5000 A=[C=2.29 I=2.30 U=122.0] V=[C=2.16 I=3.53 U=38.0]
T=6000 A=[C=5.02 I=3.89 U=206.0] V=[C=6.25 I=2.19 U=106.0]
T=7000 A=[C=5.92 I=2.74 U=179.0] V=[C=2.84 I=3.22 U=99.0]
T=8000 A=[C=3.42 I=10.69 U=101.0] V=[C=1.55 I=3.10 U=131.0]
T=9000 A=[C=2.82 I=4.46 U=104.0] V=[C=3.18 I=1.99 U=37.0]
T=10000 A=[C=4.01 I=4.05 U=154.0] V=[C=2.37 I=2.61 U=41.0]
Number of Relations:
(1 0 0) (2 0 0) (3 0 0) (4 0 0) (5 0 0) (6 0 0) (7 0 0) (8 0 0) (9 0 0)
(10 0 0) (11 4 4) (12 11 11) (13 11 11) (14 11 11) (15 14 14) (16 12 12)
(17 15 15) (18 17 17) (19 17 17) (20 25 25) (21 29 29) (22 32 32) (23 32 32)
(24 32 32) (25 32 32) (26 40 40) (27 42 42) (28 43 43) (29 43 43) (30 52 52)
(31 54 54) (32 57 57) (33 57 57) (34 57 57) (35 60 60) (36 62 62) (37 64 64)
(38 64 64) (39 65 65) (40 71 71) (41 71 71) (42 72 72) (43 73 73) (44 73 73)
(45 73 73) (46 73 73) (47 74 74) (48 74 74) (49 74 74) (50 74 74) (51 78 78)
(52 78 78) (53 78 78) (54 78 78) (55 73 73) (56 72 72) (57 72 72) (58 72 72)
(59 73 73) (60 74 74) (61 77 77) (62 80 80) (63 80 80) (64 79 79) (65 82 82)
(66 82 82) (67 83 83) (68 87 87) (69 87 87) (70 87 87) (71 87 87) (72 87 87)
(73 87 87) (74 87 87) (75 87 87) (76 87 87) (77 87 87) (78 86 86) (79 87 87)
(80 87 87) (81 87 87) (82 87 87) (83 90 90) (84 90 90) (85 90 90) (86 90 90)
(87 94 94) (88 94 94) (89 94 94) (90 94 94) (91 93 93) (92 93 93) (93 94 94)
(94 95 95) (95 95 95) (96 95 95) (97 95 95) (98 96 96) (99 96 96) (100 97 97)
```

83 minutes, starting at 8am, sampling every 0.5 seconds

```
Audio Relations: [Omitted]
Video Relations: [Omitted]
Inflections: ((PRESENT PRESENT) (PRESENT MOTION))
             ((PRESENT PRESENT) (MOTION PRESENT))
Predictions:
T=1000 A=[C=0.00 I=0.00 U=313.0] V=[C=0.52 I=0.00 U=659.0]
T=2000 A=[C=0.68 I=1.10 U=415.0] V=[C=3.15 I=1.62 U=698.0]
T=3000 A=[C=1.21 I=5.70 U=178.0] V=[C=4.65 I=4.17 U=625.0]
T=4000 A=[C=0.51 I=0.46 U=151.0] V=[C=4.85 I=12.83 U=567.0]
T=5000 A=[C=1.15 I=7.33 U=148.0] V=[C=3.92 I=4.34 U=540.0]
T=6000 A=[C=1.53 I=15.13 U=150.0] V=[C=1.39 I=3.14 U=501.0]
T=7000 A=[C=0.39 I=14.67 U=311.0] V=[C=4.63 I=14.36 U=389.0]
T=8000 A=[C=1.78 I=33.02 U=426.0] V=[C=4.71 I=3.37 U=480.0]
T=9000 A=[C=2.94 I=29.24 U=490.0] V=[C=1.94 I=22.15 U=326.0]
T=10000 A=[C=2.44 I=15.72 U=281.0] V=[C=0.92 I=4.73 U=360.0]
Number of Relations:
(1 0 0) (2 0 0) (3 0 0) (4 0 0) (5 0 0) (6 0 0) (7 0 0) (8 1 1) (9 1 1)
(10 1 1) (11 1 1) (12 1 1) (13 1 1) (14 1 1) (15 1 1) (16 1 1) (17 3 3)
(18 2 2) (19 4 4) (20 6 6) (21 8 8) (22 7 7) (23 10 10) (24 9 9) (25 9 9)
(26 10 10) (27 16 16) (28 16 16) (29 15 15) (30 15 15) (31 16 16) (32 17 17)
(33 18 18) (34 27 27) (35 28 28) (36 27 27) (37 26 26) (38 26 26) (39 27 27)
(40 27 27) (41 32 32) (42 31 31) (43 31 31) (44 33 33) (45 34 34) (46 36 36)
(47 36 36) (48 37 37) (49 38 38) (50 38 38) (51 38 38) (52 46 46) (53 46 46)
(54 50 50) (55 50 50) (56 53 53) (57 51 51) (58 51 51) (59 59 59) (60 60 60)
(61 67 67) (62 70 70) (63 72 72) (64 79 79) (65 80 80) (66 78 78) (67 77 77)
(68 77 77) (69 79 79) (70 80 80) (71 84 84) (72 84 84) (73 84 84) (74 85 85)
(75 87 87) (76 87 87) (77 88 88) (78 93 93) (79 93 93) (80 93 93) (81 94 94)
(82 97 97) (83 96 96) (84 93 93) (85 91 91) (86 93 93) (87 97 97) (88 98 98)
(89 97 97) (90 95 95) (91 98 98) (92 99 99) (93 101 101) (94 107 107)
(95 109 109) (96 112 112) (97 113 113) (98 115 115) (99 116 116) (100 118 118)
```

83 minutes, starting at 3pm, sampling every 0.5 seconds

```
Audio Relations: [Identical except missing (70 VAN "d") (VOICE VAN "E")
  and (TALK VAN "E")]
Video Relations: [Omitted]
Inflections: ((PRESENT PRESENT) (PRESENT MOTION) (TYPE PRESENT) (TYPE MOTION))
             ((PRESENT PRESENT) (PRESENT TYPE) (MOTION PRESENT) (MOTION TYPE))
Predictions:
T=1000 A=[C=0.00 I=0.00 U=308.0] V=[C=0.09 I=0.00 U=733.0]
T=2000 A=[C=0.04 I=11.06 U=278.0] V=[C=0.79 I=1.07 U=589.0]
T=3000 A=[C=0.43 I=7.25 U=261.0] V=[C=0.29 I=7.71 U=626.0]
T=4000 A=[C=3.32 I=21.52 U=280.0] V=[C=1.71 I=4.41 U=593.0]
T=5000 A=[C=3.09 I=8.42 U=204.0] V=[C=2.38 I=9.19 U=612.0]
T=6000 A=[C=2.14 I=5.56 U=339.0] V=[C=1.27 I=8.35 U=676.0]
T=7000 A=[C=2.81 I=16.95 U=256.0] V=[C=1.10 I=9.44 U=610.0]
T=8000 A=[C=1.65 I=3.04 U=193.0] V=[C=3.05 I=13.55 U=545.0]
T=9000 A=[C=1.57 I=6.01 U=370.0] V=[C=3.72 I=9.52 U=432.0]
T=10000 A=[C=2.24 I=7.36 U=345.0] V=[C=1.39 I=4.63 U=301.0]
Number of Relations:
(1 0 0) (2 0 0) (3 0 0) (4 0 0) (5 0 0) (6 0 0) (7 1 1) (8 1 1) (9 1 1)
(10 1 1) (11 1 1) (12 1 1) (13 1 1) (14 5 5) (15 6 6) (16 5 5) (17 5 5)
(18 8 8) (19 8 8) (20 12 12) (21 12 12) (22 15 15) (23 16 16) (24 15 15)
(25 27 27) (26 30 30) (27 30 30) (28 28 28) (29 26 26) (30 26 26) (31 30 30)
(32 30 30) (33 31 31) (34 34 34) (35 35 35) (36 40 40) (37 43 43) (38 44 44)
(39 43 43) (40 43 43) (41 45 45) (42 49 49) (43 50 50) (44 50 50) (45 49 49)
(46 53 53) (47 54 54) (48 55 55) (49 53 53) (50 54 54) (51 54 54) (52 53 53)
(53 53 53) (54 53 53) (55 53 53) (56 55 55) (57 55 55) (58 56 56) (59 56 56)
(60 59 59) (61 59 59) (62 63 63) (63 66 66) (64 70 70) (65 70 70) (66 72 72)
(67 73 73) (68 72 72) (69 71 71) (70 71 71) (71 73 73) (72 74 74) (73 78 78)
(74 78 78) (75 85 85) (76 87 87) (77 87 87) (78 88 88) (79 95 95) (80 97 97)
(81 103 103) (82 101 101) (83 105 105) (84 104 104) (85 105 105) (86 110 110)
(87 110 110) (88 111 111) (89 111 111) (90 110 110) (91 116 116) (92 116 116)
(93 117 117) (94 120 120) (95 118 118) (96 124 124) (97 127 127) (98 127 130)
(99 125 128) (100 126 129)
```

5.6 hours, starting at noon, sampling every 0.5 seconds

```
Audio Results:
((CUCKOO WALKLITE "QUuEe") (CUCKOO DONTWALK "eD") (CUCKOO CAR "ue")
 (CUCKOO PERSON "ue") (CUCKOO ADULT "ue") (CUCKOO CHILD "e")
 (CUCKOO SEDAN "ue") (CUCKOO TRUCK "e") (CUCKOO VAN "e") (CUCKOO SUV "e")
 (CUCKOO CYAN "e") (CUCKOO BLUE "e") (CUCKOO MAGENTA "e") (CUCKOO UR "E")
 (CUCKOO U "E") (CUCKOO F "D") (CUCKOO B "E") (IDLE WALKLITE "UE")
 (IDLE CAR "ue") (IDLE SEDAN "u") (IDLE TRUCK "UE") (IDLE VAN "UE")
 (IDLE SUV "UE") (IDLE TOWTRUCK "E") (IDLE MAGENTA "E") (IDLE R "D")
 (IDLE UR "UE") (IDLE U "UE") (IDLE UL "UE") (IDLE DL "UE") (IDLE D "UE")
 (IDLE DR "UE") (DRIVE WALKLITE "UE") (DRIVE DONTWALK "u") (DRIVE CAR "QE")
 (DRIVE SEDAN "QUE") (DRIVE TRUCK "UE") (DRIVE VAN "UE") (DRIVE SUV "UE")
 (DRIVE TOWTRUCK "UE") (DRIVE CYAN "ED") (DRIVE BLUE "E") (DRIVE MAGENTA "E")
 (DRIVE R "UE") (DRIVE UR "UE") (DRIVE U "UE") (DRIVE UL "UE") (DRIVE DL "UE")
 (DRIVE D "UE") (DRIVE DR "UE") (DRIVE F "UED") (DRIVE B "UED")
 (HONK DONTWALK "e") (HONK CAR "ue") (HONK SEDAN "e") (HONK BLUE "E")
 (HONK MAGENTA "E") (HONK R "E") (HONK F "E") (TALK WALKLITE "E")
 (TALK PERSON "e") (TALK ADULT "e") (TALK UR "E") (TALK U "E") (TALK UL "E")
 (TALK DL "E") (TALK D "E") (TALK DR "UE") (TALK B "E") (ENGINE CAR "QUE")
 (ENGINE SEDAN "UE") (ENGINE VAN "UE") (ENGINE B "E") (VOICE WALKLITE "E")
 (VOICE PERSON "e") (VOICE ADULT "e") (VOICE UR "E") (VOICE U "E")
 (VOICE UL "E") (VOICE DL "E") (VOICE D "E") (VOICE DR "UE") (VOICE B "E")
 (F WALKLITE "UE") (F CAR "Que") (F SEDAN "Q") (F TRUCK "U") (F VAN "U")
 (F SUV "U") (F TOWTRUCK "E") (F CYAN "E") (F BLUE "E") (F MAGENTA "E")
 (F UR "UE") (F U "UE") (F D "UE") (F B "Q") (FL WALKLITE "UE") (FL CAR "Q")
 (FL CHILD "E") (FL SEDAN "Q") (FL TRUCK "U") (FL VAN "UE") (FL SUV "U")
 (FL BLUE "E") (FL R "QUE") (FL UR "UE") (FL U "UE") (FL DL "UE") (FL B "Q")
 (L ADULT "e") (L UR "E") (L U "E") (L DL "E") (L D "E") (L DR "E")
 (BL WALKLITE "UE") (BL DONTWALK "ue") (BL PERSON "e") (BL ADULT "e")
 (BL CHILD "e") (BL UR "UE") (BL U "E") (BL UL "E") (BL DL "E") (BL D "E")
 (BL DR "E") (BR WALKLITE "E") (BR DONTWALK "ue") (BR CAR "Qu")
 (BR PERSON "e") (BR ADULT "e") (BR CYAN "D") (BR UR "E") (BR U "E")
 (BR UL "E") (BR DL "E") (BR D "E") (BR DR "E") (R WALKLITE "E")
 (R DONTWALK "u") (R CAR "ue") (R PERSON "e") (R ADULT "e") (R CHILD "e")
 (R SEDAN "e") (R R "e") (R UR "E") (R U "E") (R UL "E") (R DL "E") (R D "E")
 (R DR "E") (R F "e") (FR WALKLITE "UE") (FR CAR "Q") (FR TRUCK "U")
 (FR VAN "UE") (FR SUV "UE") (FR TOWTRUCK "UE") (FR CYAN "E") (FR BLUE "E")
 (FR MAGENTA "E") (FR UR "E") (FR U "UE") (FR DL "UE") (FR D "UE") (FR B "E")
 (50 WALKLITE "UE") (50 VAN "UE") (50 SUV "UE") (50 TOWTRUCK "UE")
 (50 CYAN "E") (50 BLUE "E") (50 MAGENTA "UE") (50 R "E") (50 F "E")
 (60 WALKLITE "UE") (60 DONTWALK "ue") (60 CAR "QE") (60 SEDAN "UE")
 (60 TRUCK "UE") (60 VAN "UE") (60 SUV "UE") (60 TOWTRUCK "UE") (60 CYAN "ED")
 (60 BLUE "UE") (60 MAGENTA "E") (60 R "UE") (60 UR "UE") (60 U "UE")
 (60 UL "UE") (60 DL "UE") (60 D "UE") (60 DR "UE") (60 F "UE") (60 B "UE")
 (70 WALKLITE "UEe") (70 DONTWALK "ue") (70 CAR "Qud") (70 ADULT "e")
 (70 CHILD "e") (70 TRUCK "UEd") (70 VAN "U") (70 SUV "UEd") (70 UR "E")
 (70 U "E") (70 UL "E") (70 DL "E") (70 D "E") (70 DR "E") (70 F "e")
 (70 B "E") (90 DONTWALK "e") (90 CAR "ue") (90 SEDAN "ue") (90 BLUE "E")
 (90 MAGENTA "E") (100 DONTWALK "e") (100 CAR "ue") (100 SEDAN "e"))
Video Results:
((CUCKOO WALKLITE "QUuEe") (IDLE WALKLITE "ue") (DRIVE WALKLITE "ue")
 (TALK WALKLITE "e") (VOICE WALKLITE "e") (F WALKLITE "ue") (FL WALKLITE "ue")
 (BL WALKLITE "ue") (BR WALKLITE "e") (R WALKLITE "e") (FR WALKLITE "ue")
 (50 WALKLITE "ue") (60 WALKLITE "ue") (70 WALKLITE "uEe")
 (CUCKOO DONTWALK "Ed") (DRIVE DONTWALK "U") (HONK DONTWALK "E")
 (BL DONTWALK "UE") (BR DONTWALK "UE") (R DONTWALK "U") (60 DONTWALK "UE")
 (70 DONTWALK "UE") (90 DONTWALK "E") (100 DONTWALK "E") (CUCKOO CAR "UE")
 (IDLE CAR "UE") (DRIVE CAR "Qe") (HONK CAR "UE") (ENGINE CAR "Que")
 (F CAR "QUE") (FL CAR "Q") (BR CAR "QU") (R CAR "UE") (FR CAR "Q")
 (60 CAR "Qe") (70 CAR "QUD") (90 CAR "UE") (100 CAR "UE")
 (CUCKOO PERSON "UE") (TALK PERSON "E") (VOICE PERSON "E") (BL PERSON "E")
 (BR PERSON "E") (R PERSON "E") (CUCKOO ADULT "UE") (TALK ADULT "E")
 (VOICE ADULT "E") (L ADULT "E") (BL ADULT "E") (BR ADULT "E") (R ADULT "E")
 (70 ADULT "E") (CUCKOO CHILD "E") (FL CHILD "e") (BL CHILD "E") (R CHILD "E")
 (70 CHILD "E") (CUCKOO SEDAN "UE") (IDLE SEDAN "U") (DRIVE SEDAN "Que")
 (HONK SEDAN "E") (ENGINE SEDAN "ue") (F SEDAN "Q") (FL SEDAN "Q")
 (R SEDAN "E") (60 SEDAN "ue") (90 SEDAN "UE") (100 SEDAN "E")
 (CUCKOO TRUCK "E") (IDLE TRUCK "ue") (DRIVE TRUCK "ue") (F TRUCK "u")
 (FL TRUCK "u") (FR TRUCK "u") (60 TRUCK "ue") (70 TRUCK "ueD")
 (CUCKOO VAN "E") (IDLE VAN "ue") (DRIVE VAN "ue") (ENGINE VAN "ue")
 (F VAN "u") (FL VAN "ue") (FR VAN "ue") (50 VAN "ue") (60 VAN "ue")
 (70 VAN "u") (CUCKOO SUV "E") (IDLE SUV "ue") (DRIVE SUV "ue") (F SUV "u")
 (FL SUV "u") (FR SUV "ue") (50 SUV "ue") (60 SUV "ue") (70 SUV "ueD")
 (IDLE TOWTRUCK "e") (DRIVE TOWTRUCK "ue") (F TOWTRUCK "e") (FR TOWTRUCK "ue")
 (50 TOWTRUCK "ue") (60 TOWTRUCK "ue") (CUCKOO CYAN "E") (DRIVE CYAN "ed")
 (F CYAN "e") (BR CYAN "d") (FR CYAN "e") (50 CYAN "e") (60 CYAN "ed")
 (CUCKOO BLUE "E") (DRIVE BLUE "e") (HONK BLUE "e") (F BLUE "e") (FL BLUE "e")
 (FR BLUE "e") (50 BLUE "e") (60 BLUE "ue") (90 BLUE "e") (CUCKOO MAGENTA "E")
 (IDLE MAGENTA "e") (DRIVE MAGENTA "e") (HONK MAGENTA "e") (F MAGENTA "e")
 (FR MAGENTA "e") (50 MAGENTA "ue") (60 MAGENTA "e") (90 MAGENTA "e")
 (IDLE R "d") (DRIVE R "ue") (HONK R "e") (FL R "Que") (R R "E") (50 R "e")
 (60 R "ue") (CUCKOO UR "e") (IDLE UR "ue") (DRIVE UR "ue") (TALK UR "e")
 (VOICE UR "e") (F UR "ue") (FL UR "ue") (L UR "e") (BL UR "ue") (BR UR "e")
 (R UR "e") (FR UR "e") (60 UR "ue") (70 UR "e") (CUCKOO U "e") (IDLE U "ue")
 (DRIVE U "ue") (TALK U "e") (VOICE U "e") (F U "ue") (FL U "ue") (L U "e")
 (BL U "e") (BR U "e") (R U "e") (FR U "ue") (60 U "ue") (70 U "e")
 (IDLE UL "ue") (DRIVE UL "ue") (TALK UL "e") (VOICE UL "e") (BL UL "e")
 (BR UL "e") (R UL "e") (60 UL "ue") (70 UL "e") (IDLE L "d") (DRIVE L "ue")
 (HONK L "e") (FR L "e") (60 L "ue") (90 L "e") (IDLE DL "ue") (DRIVE DL "ue")
 (TALK DL "e") (VOICE DL "e") (FL DL "ue") (L DL "e") (BL DL "e") (BR DL "e")
 (R DL "e") (FR DL "ue") (60 DL "ue") (70 DL "e") (IDLE D "ue") (DRIVE D "ue")
 (TALK D "e") (VOICE D "e") (F D "ue") (L D "e") (BL D "e") (BR D "e")
 (R D "e") (FR D "ue") (60 D "ue") (70 D "e") (IDLE DR "ue") (DRIVE DR "ue")
 (TALK DR "ue") (VOICE DR "ue") (L DR "e") (BL DR "e") (BR DR "e") (R DR "e")
 (60 DR "ue") (70 DR "e") (CUCKOO F "d") (DRIVE F "ued") (HONK F "e")
 (R F "E") (50 F "e") (60 F "ue") (70 F "E") (CUCKOO B "e") (DRIVE B "ued")
 (TALK B "e") (ENGINE B "e") (VOICE B "e") (F B "Q") (FL B "Q") (FR B "e")
 (60 B "ue") (70 B "e"))
Inflections: ((0 0)) ((0 0))
Predictions:
T=1000 A=[C=0.00 I=0.00 U=405.0] V=[C=0.00 I=0.00 U=703.0]
```

```
T=2000 A=[C=0.04 I=0.00 U=294.0] V=[C=1.04 I=2.00 U=631.0]
T=3000 A=[C=0.51 I=0.52 U=292.0] V=[C=1.09 I=4.26 U=428.0]
T=4000 A=[C=0.48 I=4.16 U=328.0] V=[C=1.78 I=10.11 U=532.0]
T=5000 A=[C=1.17 I=6.91 U=180.0] V=[C=1.37 I=6.08 U=486.0]
T=6000 A=[C=3.31 I=13.76 U=288.0] V=[C=1.90 I=1.99 U=398.0]
T=7000 A=[C=0.87 I=11.64 U=141.0] V=[C=2.12 I=4.29 U=388.0]
T=8000 A=[C=0.94 I=10.97 U=119.0] V=[C=1.93 I=3.46 U=404.0]
T=9000 A=[C=0.00 I=0.25 U=306.0] V=[C=0.94 I=18.70 U=153.0]
T=10000 A=[C=0.00 I=0.00 U=116.0] V=[C=0.14 I=10.22 U=69.0]
T=11000 A=[C=0.17 I=14.15 U=369.0] V=[C=2.29 I=8.84 U=127.0]
T=12000 A=[C=0.56 I=0.41 U=598.0] V=[C=2.88 I=20.94 U=180.0]
T=13000 A=[C=3.84 I=29.41 U=336.0] V=[C=1.33 I=16.88 U=204.0]
T=14000 A=[C=2.86 I=68.85 U=193.0] V=[C=0.29 I=2.72 U=382.0]
T=15000 A=[C=10.60 I=28.07 U=307.0] V=[C=0.62 I=3.00 U=607.0]
T=16000 A=[C=1.70 I=28.58 U=204.0] V=[C=1.01 I=1.05 U=278.0]
T=17000 A=[C=1.15 I=12.22 U=195.0] V=[C=0.97 I=3.27 U=372.0]
T=18000 A=[C=0.35 I=24.81 U=139.0] V=[C=0.30 I=8.01 U=233.0]
T=19000 A=[C=0.75 I=21.10 U=138.0] V=[C=0.51 I=7.86 U=406.0]
T=20000 A=[C=2.52 I=16.41 U=203.0] V=[C=0.30 I=0.52 U=398.0]
T=21000 A=[C=0.44 I=7.84 U=71.0] V=[C=0.37 I=0.46 U=326.0]
T=22000 A=[C=0.54 I=12.92 U=65.0] V=[C=0.29 I=0.37 U=320.0]
T=23000 A=[C=1.21 I=10.15 U=116.0] V=[C=0.18 I=2.59 U=213.0]
T=24000 A=[C=0.28 I=8.53 U=148.0] V=[C=0.36 I=1.17 U=291.0]
T=25000 A=[C=0.31 I=11.26 U=100.0] V=[C=0.21 I=0.64 U=251.0]
T=26000 A=[C=0.70 I=6.47 U=174.0] V=[C=0.79 I=0.86 U=225.0]
T=27000 A=[C=0.72 I=42.47 U=111.0] V=[C=1.17 I=1.29 U=155.0]
T=28000 A=[C=0.90 I=12.30 U=195.0] V=[C=0.45 I=24.16 U=163.0]
T=29000 A=[C=0.09 I=2.95 U=60.0] V=[C=0.54 I=16.52 U=66.0]
T=30000 A=[C=0.50 I=1.50 U=64.0] V=[C=0.42 I=21.83 U=65.0]
T=31000 A=[C=0.02 I=0.42 U=81.0] V=[C=0.72 I=12.61 U=19.0]
T=32000 A=[C=0.00 I=0.10 U=87.0] V=[C=0.31 I=23.64 U=7.0]
T=33000 A=[C=1.16 I=0.33 U=150.0] V=[C=0.23 I=17.04 U=145.0]
T=34000 A=[C=0.65 I=6.91 U=112.0] V=[C=1.66 I=9.50 U=327.0]
T=35000 A=[C=0.00 I=3.00 U=100.0] V=[C=0.59 I=11.91 U=81.0]
T=36000 A=[C=0.00 I=3.00 U=74.0] V=[C=0.89 I=13.50 U=53.0]
T=37000 A=[C=0.00 I=2.50 U=46.0] V=[C=0.95 I=11.81 U=138.0]
T=38000 A=[C=0.01 I=0.00 U=73.0] V=[C=1.35 I=7.36 U=69.0]
T=39000 A=[C=0.00 I=2.50 U=115.0] V=[C=1.01 I=14.51 U=64.0]
T=40000 A=[C=0.00 I=0.00 U=139.0] V=[C=0.82 I=8.61 U=90.0]
Number of Relations:
 (1 0 0) (2 0 0) (3 0 0) (4 0 0) (5 0 0) (6 0 0) (7 0 0) (8 0 0) (9 0 0)
 (10 0 0) (11 0 0) (12 0 0) (13 0 0) (14 0 0) (15 0 0) (16 8 8) (17 9 10)
 (18 10 11) (19 10 11) (20 11 12) (21 15 19) (22 19 23) (23 19 23) (24 21 25)
 (25 19 24) (26 20 25) (27 21 28) (28 22 29) (29 23 30) (30 26 33) (31 22 28)
 (32 34 41) (33 39 46) (34 40 47) (35 41 48) (36 43 49) (37 46 52) (38 47 54)
 (39 54 61) (40 57 64) (41 65 71) (42 63 69) (43 63 69) (44 66 73) (45 68 75)
 (46 68 75) (47 69 76) (48 70 77) (49 69 76) (50 69 76) (51 80 87) (52 80 87)
 (53 80 87) (54 80 87) (55 80 88) (56 82 90) (57 83 91) (58 88 96) (59 88 96)
 (60 88 95) (61 90 98) (62 92 100) (63 92 100) (64 92 100) (65 92 100)
 (66 92 100) (67 96 104) (68 97 105) (69 100 108) (70 101 109) (71 102 110)
 (72 103 111) (73 104 112) (74 106 114) (75 106 114) (76 105 113) (77 102 110)
 (78 102 110) (79 102 110) (80 101 109) (81 99 106) (82 99 105) (83 99 105)
 (84 99 105) (85 98 104) (86 98 104) (87 98 104) (88 98 104) (89 99 105)
 (90 99 105) (91 99 105) (92 99 105) (93 99 105) (94 99 105) (95 99 105)
 (96 99 105) (97 99 105) (98 99 105) (99 99 105) (100 98 104) (101 98 104)
 (102 98 104) (103 98 104) (104 98 104) (105 99 105) (106 99 105) (107 100 106)
 (108 102 108) (109 102 108) (110 104 110) (111 103 109) (112 103 110)
 (113 108 115) (114 107 114) (115 108 115) (116 110 118) (117 112 120)
 (118 113 121) (119 118 126) (120 120 128) (121 119 127) (122 120 129)
 (123 125 134) (124 128 137) (125 128 137) (126 127 136) (127 132 142)
 (128 133 143) (129 134 144) (130 138 148) (131 139 150) (132 138 149)
 (133 137 148) (134 138 149) (135 140 151) (136 145 156) (137 145 156)
 (138 147 158) (139 147 157) (140 147 157) (141 145 155) (142 143 153)
 (143 138 148) (144 137 148) (145 138 148) (146 140 149) (147 140 149)
 (148 140 149) (149 141 150) (150 144 153) (151 146 155) (152 148 157)
 (153 146 155) (154 149 158) (155 149 159) (156 148 158) (157 148 158)
 (158 149 158) (159 150 159) (160 156 165) (161 155 164) (162 156 165)
 (163 157 166) (164 161 170) (165 161 170) (166 163 172) (167 167 176)
```

```
(168 172 181) (169 173 182) (170 175 184) (171 178 187) (172 179 188)
(173 180 189) (174 179 188) (175 181 190) (176 183 192) (177 181 190)
(178 184 193) (179 182 191) (180 184 194) (181 184 193) (182 184 193)
(183 184 193) (184 185 194) (185 190 199) (186 189 198) (187 186 195)
(188 186 195) (189 187 196) (190 187 196) (191 184 194) (192 186 196)
(193 192 203) (194 193 204) (195 196 207) (196 199 210) (197 199 210)
(198 202 213) (199 204 215) (200 205 216) (201 205 216) (202 205 216)
(203 205 216) (204 207 218) (205 207 218) (206 208 219) (207 206 216)
(208 205 215) (209 205 216) (210 204 215) (211 207 218) (212 211 222)
(213 211 222) (214 211 222) (215 211 222) (216 211 222) (217 211 222)
(218 216 227) (219 221 232) (220 232 243) (221 230 241) (222 232 243)
(223 233 244) (224 233 244) (225 234 245) (226 234 245) (227 245 256)
(228 245 256) (229 246 257) (230 248 259) (231 248 259) (232 248 259)
(233 249 259) (234 244 253) (235 244 253) (236 247 257) (237 252 261)
(238 252 261) (239 260 269) (240 260 269) (241 262 271) (242 265 274)
(243 265 274) (244 265 274) (245 269 279) (246 270 280) (247 272 281)
(248 271 280) (249 273 282) (250 271 280) (251 276 286) (252 286 296)
(253 287 297) (254 285 295) (255 286 296) (256 290 300) (257 291 301)
(258 292 302) (259 292 302) (260 292 302) (261 299 309) (262 300 310)
(263 296 306) (264 297 307) (265 296 306) (266 298 308) (267 298 308)
(268 302 312) (269 304 314) (270 307 317) (271 307 317) (272 306 316)
(273 304 314) (274 304 314) (275 307 317) (276 313 322) (277 315 324)
(278 317 325) (279 316 324) (280 316 324) (281 316 324) (282 315 323)
(283 315 323) (284 314 322) (285 314 322) (286 314 322) (287 314 322)
(288 313 321) (289 314 322) (290 314 322) (291 314 322) (292 314 322)
(293 314 322) (294 314 322) (295 314 322) (296 314 323) (297 314 323)
(298 314 323) (299 314 322) (300 314 322) (301 314 323) (302 316 325)
(303 316 325) (304 316 324) (305 316 324) (306 316 324) (307 316 324)
(308 316 324) (309 316 324) (310 316 324) (311 316 324) (312 317 325)
(313 317 325) (314 317 325) (315 317 325) (316 317 325) (317 316 324)
(318 315 323) (319 315 323) (320 315 323) (321 315 323) (322 315 323)
(323 315 323) (324 315 323) (325 315 323) (326 315 323) (327 315 323)
(328 317 325) (329 317 325) (330 315 323) (331 314 323) (332 314 323)
(333 315 324) (334 314 322) (335 317 325) (336 320 328) (337 321 329)
(338 322 330) (339 325 333) (340 327 336) (341 327 336) (342 327 336)
(343 327 336) (344 327 336) (345 331 340) (346 332 341) (347 332 341)
(348 332 341) (349 332 341) (350 332 341) (351 332 341) (352 332 341)
(353 332 341) (354 332 341) (355 332 341) (356 332 341) (357 332 341)
(358 331 340) (359 331 340) (360 331 340) (361 330 339) (362 330 339)
(363 330 339) (364 330 339) (365 331 340) (366 330 339) (367 330 339)
(368 330 339) (369 330 339) (370 331 340) (371 331 340) (372 331 340)
(373 329 337) (374 329 337) (375 329 337) (376 329 337) (377 329 337)
(378 329 337) (379 330 338) (380 330 338) (381 330 338) (382 329 337)
(383 331 339) (384 329 337) (385 329 337) (386 329 337) (387 329 337)
(388 330 338) (389 330 338) (390 330 338) (391 330 338) (392 330 338)
(393 330 338) (394 330 338) (395 330 338) (396 330 338) (397 331 339)
(398 332 340) (399 331 339) (400 330 338)
```

B.3 Focus Learning

These are the results of two specialists learning to communicate using reflex-driven focus of attention, used in Section 8.3.

83 minutes, starting at noon, sampling every 0.5 seconds, 4 foci, 1 request per sample

```
Audio Results:
((IDLE CAR "ue") (IDLE SEDAN "e") (IDLE RED "e") (IDLE GREEN "E")
 (IDLE BLUE "e") (IDLE DARK "ue") (IDLE OD "e") (IDLE 8D "ue")
 (DRIVE CAR "QUuEe") (DRIVE SEDAN "QUEe") (DRIVE TRUCK "UE") (DRIVE VAN "UEe")
 (DRIVE SUV "UE") (DRIVE RED "E") (DRIVE YELLOW "E") (DRIVE GREEN "E")
 (DRIVE CYAN "E") (DRIVE BLUE "UE") (DRIVE MAGENTA "E") (DRIVE DARK "QUuEe")
 (DRIVE OD "UEe") (DRIVE 8D "QUuEe") (DRIVE 16D "UE") (DRIVE 24D "UE")
 (DRIVE R "UEe") (DRIVE L "UEe") (DRIVE F "UEe") (DRIVE B "UEe")
 (STEPS PERSON "QUuEe") (STEPS ADULT "QUuEe") (STEPS CHILD "UEe")
 (STEPS RED "E") (STEPS YELLOW "E") (STEPS GREEN "E") (STEPS CYAN "E")
 (STEPS BLUE "E") (STEPS MAGENTA "Ee") (STEPS DARK "uEe") (STEPS OD "uEe")
 (STEPS 8D "uEe") (STEPS 16D "E") (STEPS R "E") (STEPS L "E") (STEPS F "E")
 (STEPS B "E") (ENGINE CAR "QUuEe") (ENGINE SEDAN "QUEe") (ENGINE TRUCK "UE")
 (ENGINE VAN "UE") (ENGINE SUV "UE") (ENGINE RED "E") (ENGINE YELLOW "E")
 (ENGINE GREEN "UE") (ENGINE CYAN "UE") (ENGINE BLUE "UE")
 (ENGINE MAGENTA "E") (ENGINE DARK "QUuEe") (ENGINE OD "QUEe")
 (ENGINE 8D "QUuEe") (ENGINE 16D "UE") (ENGINE 24D "UE") (ENGINE R "UEe")
 (ENGINE L "UEe") (ENGINE F "UE") (ENGINE B "UEe") (F CAR "uEe")
 (F PERSON "UEe") (F ADULT "UEe") (F CHILD "UE") (F SEDAN "Ee")
 (F TRUCK "UEe") (F VAN "E") (F SUV "UE") (F RED "UEe") (F YELLOW "UE")
 (F GREEN "UE") (F CYAN "UE") (F BLUE "E") (F MAGENTA "E") (F DARK "QuEe")
 (F OD "Ee") (F 8D "QuEe") (F 16D "UE") (F 24D "UE") (F R "UEe") (F L "Ee")
 (F F "E") (F B "Ee") (FL CAR "uEe") (FL PERSON "Ee") (FL ADULT "E")
 (FL SEDAN "Ee") (FL TRUCK "Ee") (FL VAN "Ee") (FL SUV "Ee") (FL RED "E")
 (FL YELLOW "Ee") (FL GREEN "E") (FL CYAN "E") (FL BLUE "E") (FL MAGENTA "E")
 (FL DARK "ue") (FL OD "Ee") (FL 8D "ue") (FL 16D "E") (FL 24D "E")
 (FL R "Ee") (FL L "e") (FL F "Ee") (FL B "Ee") (L CAR "e") (L SEDAN "e")
 (L DARK "e") (L 8D "eD") (L L "eD") (L F "eD") (BL B "E") (BR SEDAN "E")
 (BR R "E") (BR F "E") (R CAR "eD") (R SEDAN "e") (R DARK "ue") (R 16D "eD")
 (R 24D "eD") (R R "eD") (R F "eD") (FR CAR "uEe") (FR PERSON "Ee")
 (FR ADULT "Ee") (FR SEDAN "Ee") (FR TRUCK "Ee") (FR VAN "Ee") (FR SUV "Ee")
 (FR RED "E") (FR YELLOW "E") (FR GREEN "E") (FR CYAN "E") (FR BLUE "E")
 (FR MAGENTA "E") (FR DARK "uEe") (FR OD "Ee") (FR 8D "uEe") (FR 16D "E")
 (FR 24D "UE") (FR R "e") (FR L "Ee") (FR F "Ee") (FR B "Ee") (40 CAR "e")
 (40 PERSON "Ee") (40 ADULT "Ee") (40 CHILD "E") (40 SEDAN "e") (40 RED "Ee")
 (40 YELLOW "E") (40 GREEN "E") (40 CYAN "E") (40 BLUE "E") (40 MAGENTA "E")
 (40 DARK "ue") (40 OD "uEe") (40 8D "ue") (50 CAR "ue") (50 PERSON "UEe")
 (50 ADULT "UEe") (50 CHILD "UE") (50 SEDAN "Ee") (50 VAN "E") (50 SUV "E")
 (50 RED "E") (50 YELLOW "E") (50 GREEN "E") (50 CYAN "E") (50 BLUE "E")
 (50 MAGENTA "UE") (50 DARK "uEe") (50 OD "uEe") (50 8D "uEe") (50 16D "E")
 (50 24D "E") (50 R "Ee") (50 L "Ee") (50 F "E") (50 B "e") (60 CAR "QUuEe")
 (60 SEDAN "UEe") (60 TRUCK "UE") (60 VAN "UEe") (60 SUV "UE") (60 RED "E")
 (60 YELLOW "E") (60 GREEN "E") (60 CYAN "UE") (60 BLUE "UE") (60 MAGENTA "E")
 (60 DARK "QUuEe") (60 OD "UEe") (60 8D "QUuEe") (60 16D "UE") (60 24D "UE")
 (60 R "UEe") (60 L "UEe") (60 F "UEe") (60 B "UEe") (70 CAR "uEe")
 (70 SEDAN "Ee") (70 TRUCK "Ee") (70 VAN "Ee") (70 SUV "Ee") (70 RED "e")
 (70 YELLOW "e") (70 GREEN "e") (70 BLUE "E") (70 DARK "uEe") (70 OD "e")
 (70 8D "ue") (70 16D "QUEe") (70 24D "UE") (70 R "Ee") (70 L "Ee") (70 F "e")
 (70 B "Ee"))
Video Results:
((IDLE CAR "UE") (DRIVE CAR "QUuEe") (ENGINE CAR "QUuEe") (F CAR "UEe")
 (FL CAR "UEe") (L CAR "E") (R CAR "Ed") (FR CAR "UEe") (40 CAR "E")
 (50 CAR "UE") (60 CAR "QUuEe") (70 CAR "UEe") (STEPS PERSON "QUuEe")
 (F PERSON "uEe") (FL PERSON "Ee") (FR PERSON "Ee") (40 PERSON "Ee")
 (50 PERSON "uEe") (STEPS ADULT "QUuEe") (F ADULT "uEe") (FL ADULT "e")
 (FR ADULT "Ee") (40 ADULT "Ee") (50 ADULT "uEe") (STEPS CHILD "uEe")
 (F CHILD "ue") (40 CHILD "e") (50 CHILD "ue") (IDLE SEDAN "E")
 (DRIVE SEDAN "QuEe") (ENGINE SEDAN "QuEe") (F SEDAN "Ee") (FL SEDAN "Ee")
 (L SEDAN "E") (BR SEDAN "e") (R SEDAN "E") (FR SEDAN "Ee") (40 SEDAN "E")
 (50 SEDAN "Ee") (60 SEDAN "uEe") (70 SEDAN "Ee") (DRIVE TRUCK "ue")
 (ENGINE TRUCK "ue") (F TRUCK "uEe") (FL TRUCK "Ee") (FR TRUCK "Ee")
 (60 TRUCK "ue") (70 TRUCK "Ee") (DRIVE VAN "uEe") (ENGINE VAN "ue")
 (F VAN "e") (FL VAN "Ee") (FR VAN "Ee") (50 VAN "e") (60 VAN "uEe")
 (70 VAN "Ee") (DRIVE SUV "ue") (ENGINE SUV "ue") (F SUV "ue") (FL SUV "Ee")
 (FR SUV "Ee") (50 SUV "e") (60 SUV "ue") (70 SUV "Ee") (IDLE RED "E")
 (DRIVE RED "e") (STEPS RED "e") (ENGINE RED "e") (F RED "uEe") (FL RED "e")
 (FR RED "e") (40 RED "Ee") (50 RED "e") (60 RED "e") (70 RED "E")
 (DRIVE YELLOW "e") (STEPS YELLOW "e") (ENGINE YELLOW "e") (F YELLOW "ue")
 (FL YELLOW "Ee") (FR YELLOW "e") (40 YELLOW "e") (50 YELLOW "e")
 (60 YELLOW "e") (70 YELLOW "E") (IDLE GREEN "e") (DRIVE GREEN "e")
 (STEPS GREEN "e") (ENGINE GREEN "ue") (F GREEN "ue") (FL GREEN "e")
 (FR GREEN "e") (40 GREEN "e") (50 GREEN "e") (60 GREEN "e") (70 GREEN "E")
 (DRIVE CYAN "e") (STEPS CYAN "e") (ENGINE CYAN "ue") (F CYAN "ue")
 (FL CYAN "e") (FR CYAN "e") (40 CYAN "e") (50 CYAN "e") (60 CYAN "ue")
 (IDLE BLUE "E") (DRIVE BLUE "ue") (STEPS BLUE "e") (ENGINE BLUE "ue")
 (F BLUE "e") (FL BLUE "e") (FR BLUE "e") (40 BLUE "e") (50 BLUE "e")
 (60 BLUE "ue") (70 BLUE "e") (DRIVE MAGENTA "e") (STEPS MAGENTA "Ee")
 (ENGINE MAGENTA "e") (F MAGENTA "e") (FL MAGENTA "e") (FR MAGENTA "e")
 (40 MAGENTA "e") (50 MAGENTA "ue") (60 MAGENTA "e") (IDLE DARK "UE")
 (DRIVE DARK "QUuEe") (STEPS DARK "UEe") (ENGINE DARK "QUuEe") (F DARK "QUEe")
 (FL DARK "UE") (L DARK "E") (R DARK "UE") (FR DARK "UEe") (40 DARK "UE")
 (50 DARK "UEe") (60 DARK "QUuEe") (70 DARK "UEe") (IDLE 0D "E")
 (DRIVE 0D "uEe") (STEPS 0D "UEe") (ENGINE 0D "QuEe") (F 0D "Ee") (FL 0D "Ee")
 (FR 0D "Ee") (40 0D "UEe") (50 0D "UEe") (60 0D "uEe") (70 0D "E")
 (IDLE 8D "UE") (DRIVE 8D "QUuEe") (STEPS 8D "UEe") (ENGINE 8D "QUuEe")
 (F 8D "QUEe") (FL 8D "UE") (L 8D "Ed") (FR 8D "UEe") (40 8D "UE")
 (50 8D "UEe") (60 8D "QUuEe") (70 8D "UE") (DRIVE 16D "ue") (STEPS 16D "e")
 (ENGINE 16D "ue") (F 16D "ue") (FL 16D "e") (R 16D "Ed") (FR 16D "e")
 (50 16D "e") (60 16D "ue") (70 16D "QuEe") (DRIVE 24D "ue") (ENGINE 24D "ue")
 (F 24D "ue") (FL 24D "e") (R 24D "Ed") (FR 24D "ue") (50 24D "e")
 (60 24D "ue") (70 24D "ue") (DRIVE R "uEe") (STEPS R "e") (ENGINE R "uEe")
 (F R "uEe") (FL R "Ee") (BR R "e") (R R "Ed") (FR R "E") (50 R "Ee")
 (60 R "uEe") (70 R "Ee") (DRIVE L "uEe") (STEPS L "e") (ENGINE L "uEe")
 (F L "Ee") (FL L "E") (L L "Ed") (FR L "Ee") (50 L "Ee") (60 L "uEe")
 (70 L "Ee") (DRIVE F "uEe") (STEPS F "e") (ENGINE F "ue") (F F "e")
 (FL F "Ee") (L F "Ed") (BR F "e") (R F "Ed") (FR F "Ee") (50 F "e")
 (60 F "uEe") (70 F "E") (DRIVE B "uEe") (STEPS B "e") (ENGINE B "uEe")
 (F B "Ee") (FL B "Ee") (BL B "e") (FR B "Ee") (50 B "E") (60 B "uEe")
 (70 B "Ee"))
Inflections: NIL NIL
Number of Relations:
(1 0 0) (2 32 32) (3 88 88) (4 103 103) (5 133 133) (6 149 149) (7 175 175)
(8 205 205) (9 234 234) (10 234 234) (11 252 252) (12 264 264) (13 261 261)
(14 265 265) (15 280 280) (16 279 279) (17 287 287) (18 303 303) (19 311 311)
(20 320 320) (21 326 326) (22 332 332) (23 338 338) (24 337 337) (25 339 339)
(26 340 340) (27 348 348) (28 348 348) (29 358 358) (30 360 360) (31 367 367)
(32 368 368) (33 368 368) (34 374 374) (35 377 377) (36 373 373) (37 375 375)
(38 379 379) (39 383 383) (40 389 389) (41 386 386) (42 390 390) (43 390 390)
(44 390 390) (45 389 389) (46 392 392) (47 400 400) (48 402 402) (49 400 400)
(50 401 401) (51 403 403) (52 405 405) (53 407 407) (54 407 407) (55 408 408)
(56 412 412) (57 412 412) (58 414 414) (59 417 417) (60 417 417) (61 416 416)
(62 416 416) (63 417 417) (64 420 420) (65 426 426) (66 426 426) (67 429 429)
(68 425 425) (69 427 427) (70 437 437) (71 434 434) (72 434 434) (73 434 434)
(74 432 432) (75 431 431) (76 431 431) (77 433 433) (78 434 434) (79 435 435)
(80 436 436) (81 440 440) (82 438 438) (83 439 439) (84 439 439) (85 441 441)
(86 441 441) (87 440 440) (88 442 442) (89 445 445) (90 443 443) (91 441 441)
(92 444 444) (93 443 443) (94 443 443) (95 446 446) (96 441 441) (97 439 439)
(98 446 446) (99 448 448) (100 448 448)
```

Appendix C

Data Sheets

This appendix contains the data sheets for devices used in this dissertation. See Chapter 5 for an explanation of the specifications in these data sheets.

C.1 Distributed Map

A distributed map is a device for dynamically creating communication paths between elements of a set A and elements of another set B. These communication paths act as a one-to-one mapping between subsets of A and B, carrying signals in either direction.

C.1.1 Interface Specification

Conditions A distributed map connects two sets of identical elements, A and B. The two sets are not initially aligned with one another and may be of different sizes. Three types of operations can be carried out on a distributed map:

- **Connect:** Create a connection between an element $a \in A$ and another element $b \in B$, severing any previous connections to a or b.
- **Disconnect:** Disconnect a single element from its connection, if any.
- **Propagate:** Copy all the values held by elements in A to their corresponding elements in B (or vice versa).

Range of Behavior The range behavior exhibited by the three operations is:

- **Connect:** Each of the two elements being connected receives a boolean value indicating whether a connection has been made.
- **Disconnect:** No observable behavior results.

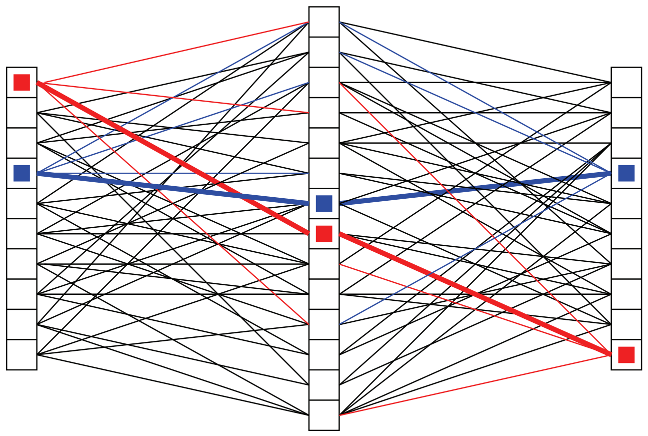


Figure C-1: A distributed map connects two sets through an intermediate set of rendezvous points. If each set element connects to a sparse random subset of rendezvous points, then a small oversupply of connections and rendezvous points will make it highly probable that the rendezvous points can be configured to represent any one-to-one function mapping from one set to the other.

- **Propagate:** each element in the target set either receives a **NULL** value or the value of an element in the source set.

Desirable Behavior A distributed map is behaving as desired when every connect operation succeeds and when the results of each propagate operation may be predicted from the preceding sequence of connect and disconnect operations.

C.1.2 Mechanism

A distributed map connects together two sets through an intermediate set of rendezvous points. The number of rendezvous points in the set R is equal to the size of the smaller set, multiplied by a small oversupply constant o_s. Each set connects to the rendezvous points with a random bipartite graph.[1] In order for any two elements to have an expected rendezvous size r_s of shared rendezvous points, we have each set element connect to $k = \sqrt{|R| \cdot r_s}$ rendezvous points.

The rendezvous points also participate in a competition, which will be used for determining which of several possible rendezvous points is used to make each connection.

Connect and Disconnect To create a communication path between two elements, we send a special creation signal from each element to its rendezvous points. The

[1]This design is similar to that used by butterfly graphs and other interconnects based on expander graphs[10].

rendezvous points that receive a signal from both sides compete (with preference given to rendezvous points that are not yet allocated). The winning rendezvous point then selects its paths that carried the creation signal: it will subsequently relay signals carried on these paths and ignore all signals on other paths besides creation signals. Finally, the rendezvous point sends an acknowledgement signal back along both paths to let the requesting elements know that creation succeeded.

As more paths are allocated, there is an increasing chance that all of the shared rendezvous points for a new path will already be allocated. In this case, one of the already allocated rendezvous points will be the winner of the competition and the path it previously carried will be unceremoniously terminated. Likewise, if a set element is connected to a new path, it disconnects from its previous path.

Communication paths can also be destroyed unilaterally: if an element sends a special deletion signal to its rendezvous points, then the rendezvous point for its communication path will reset, discarding its selections on both sides.

Propagate Propagation is initiated by clearing the values held in the target set. The source set transmits its values to the rendezvous points. The rendezvous points superimpose signals arriving on their designated paths to the source set, transmitting onward to the target set.

C.1.3 Configuration Parameters

A distributed map has two configuration parameters:

- o_s is a number greater than or equal to 1.
- r_s is a number greater than or equal to 1.

C.1.4 Dossier

Given this mechanism, misbehavior will come either from problems with connect operations or from external noise interfering with the signal being propagated. External noise is not affected by the configuration parameters, so we shall ignore it.

A connect operation can lead to misbehavior in two ways:

- There is no rendezvous point for the two elements, and the connection fails.
- There is no unallocated rendezvous point, so the new connection snaps a previous connection.

Since the likelihood of both of these is affected by distribution variance, we can make an educated guess than only a small oversupply factor o_s and moderate rendezvous size r_s will be sufficient to make such failures rare.

Saturation Survey

- **Conditions:** A and B both contain 1000 elements, initially unconnected.
- **Configurations:** o_s assumes five values, 1.0, 1.01, 1.1, 1.2 and 1.5, while r_s assumes four, 1, 3, 10, and 30.
- **Experiment:** For each of the twenty combinations of values, we run ten trials. A trial consists of a sequence of 1000 connect operations, randomly ordered to connect each element of A to an element of B in a random permutation. The success of the connection is tested with two propagate operations, one from A to B and one from B to A, where every element is given a unique value and we compare the contents of the target set to that of the source set to determining how many elements are correctly mapped and how many are incorrectly mapped.
- **Results:** No element is ever incorrectly mapped, and the number of correct mappings is always the same in both directions. Figure C-2 shows graphs of success rate plotted against the two parameters. Note that for even modest levels of o_s and r_s nearly every element connects with its pair in the other set. Finally, rendezvous size appears to have a much stronger impact on behavior than oversupply factor.

C.1.5 Usage Specification

- **Configuration Policy:** A rendezvous size of at least $r_s = 10$ and an oversupply of at least $o_s = 1.2$ should suffice to make misbehavior rare.
- **Limiting Conditions:** Misbehavior is most likely to occur when nearly all of the potential mappings are being used.
- **Failure Simplification:** With a reasonable rendezvous size, the likelihood of two elements being simply unable to connect can be made astronomically low. Much more likely are failures that occur when a new connection displaces a pre-existing connection. Assuming that connections are often being made

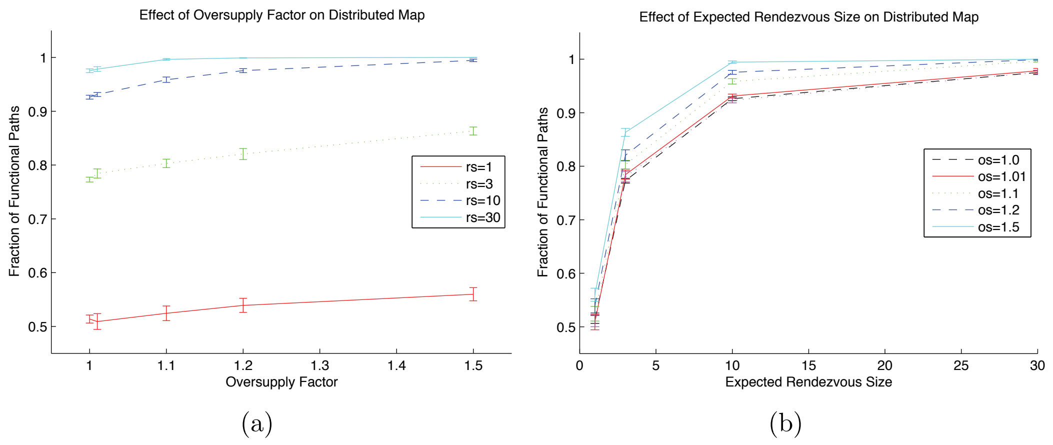


Figure C-2: A small oversupply factor and rendezvous size are all that is necessary to ensure that a distributed map can simultaneously maintain paths to nearly every element of the smaller set. These graphs show the success rate for a random permutation map between two sets of 1000 elements.

and reassigned, this can be modelled simply as a small cumulative chance of a connection randomly disappearing over time.

This misbehavior can be detected by echoing a propagation and comparing to look for missing values. If this is done not for all elements, but only for those currently being used, then we have a lazy detection mechanism that only looks for failure when they become relevant. An element that has become disconnected can always be reconnected (again with a small chance that it will destroy a currently active link), but over time the pairs most likely to be unexpectedly disconnected will be those least likely to be used, decreasing the impact.

Cost The hardware and development costs of the distributed map are set by the two random bipartite graphs composed to produce it. The execution time for disconnect and propagate operations is $O(1)$, while the execution time for connect operations is constrained by the time for competition between possible connections, to $O(1)$ amortized.

	Development	Mature
Time	$O(min(\|A\|,\|B\|))$	$O(1)$ amortized
Space	$O(1)$	$O(\sqrt{min(\|A\|,\|B\|)r_s o_s}\cdot(\|A\|+\|B\|))$
Imperfection	extra or missing paths and rendezvous points	noise, failed or dropped connections

C.2 Unidirectional Link

A unidirectional link transmits messages composed of symbols and inflections from a speaker to a listener, across a channel consisting of many communication paths connecting the speaker to the listener in an arbitrary pattern.

C.2.1 Interface Specification

Conditions There are three interfaces between a unidirectional link and the speaker: one is a boolean signal that tells the link whether it is free to engage in self-organization, another handles the allocation and deallocation of the link's vocabulary, and the third is for sending messages. On the listener side, there is only a mechanism for receiving messages.

A link has a potential vocabulary of s symbols and i inflections, where there are many more symbols than inflections ($s >> i$). Messages are specified as a set of symbol/inflection pairs, where there may be up to $m_s << s$ symbols and $m_i << i$ inflections per symbol. Any symbol may be combined with any inflection.

An allocation request asks for a symbol or inflection, but does not specify which particular symbol or inflection; a deallocation request specifies which particular symbol or inflection is to be deallocated.

Range of Behavior The listener side reports messages when they arrive; messages have the same structure when received as when sent.

An allocation request either returns a newly allocated symbol or inflection or else reports failure. A deallocation request reports only completion.

Desirable Behavior A unidirectional link is behaving desirably when there is:

- **Available vocabulary:** up to s symbols and i inflections can be allocated by the speaker for use in composing messages.

- **Clean allocation:** a newly allocated symbol or inflection is not entangled in any way with other currently allocated symbols or inflections or with those that have been deallocated in the past.

- **Persistent vocabulary:** once allocated, a symbol or inflection remains consistently usable until deallocated.

- **Consistent transmission:** there is a one-to-one mapping from sent to received symbols and inflections, and under this mapping the message received is always equivalent to the one sent.

C.2.2 Mechanism

The architecture for a link is shown in Figure C-3: in both the speaker and the listener, a crossbar connects the inflection coders to the symbol coders. The channel between the speaker and listener is an arbitrarily twisted bundle of communication paths connecting a set of c cable-head devices in the speaker to a set of c cable-head devices in the listener. The communication paths are directional: only a small amount of information is allows to flow from listener to speaker. Each symbol coder is connected by a random sparse map to k out of the c cable-heads on its side of the channel.

Besides the structure visible in Figure C-3, each set of coders includes a competition (see Section 7.1.1), and the inflections in the speaker are connected in an all-to-all network.

Inflections are encoded as a sparse patterns of p pulses out of b time slots in a burst, symbols as sparse subsets of the communication paths, and a message as the set of active junctions in the crossbar. To transmit a message, the speaker's inflection coders activate, sending their patterns across the active crossbar junctions to superimpose and activate symbol coders. The active symbol coders relay their patterns to a chosen subset of at least d_s communication paths encoding the symbol. The patterns once again superimpose at the cable-heads, then propagate along the communication paths from speaker to listener.

On the other side, each listener symbol coder monitors a chosen subset of communication paths, and if at least d_s are active, it detects the symbol and activates. Active symbol coders then calculate a consensus pattern (those portions on at least d_c communication paths) and relay it into the crossbar. Finally, the listener's inflection coders fire their patterns into the crossbar and each junction compares the two incoming patterns, setting if at least d_i pulses coincide. The contents of the message can now be read out of the pattern of active crossbar junctions.

Self-Organization Initially, the encodings for symbols and inflections in the speaker do not match those in the listener. Self-organization produces an aligned set of symbols and inflections, which may then be used for communication.

Symbol and inflection coders thus have four states:

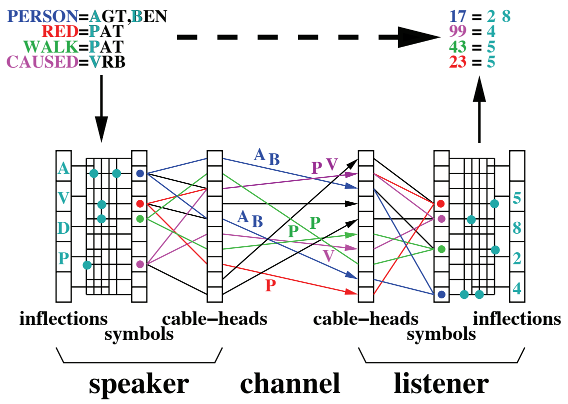


Figure C-3: A unidirectional link is implemented using sparse coding on top of a random wiring pattern. Symbols are encoded as sets of active communication paths and relations as the pattern transmitted on a symbol's communication paths.

- **disabled:** coder is not yet in use
- **immature:** coder is seeking alignment with a partner coder on the other side of the channel.
- **mature:** coder has become aligned with a partner coder and is ready for use in messages
- **allocated:** coder is aligned and has been allocated for use in building messages. This state only exists in the speaker.

The self-organization process is designed to co-exist with mature use: as soon as the first symbols mature, they can be allocated and used; once mature, a symbol is rarely interfered with by the ongoing self-organization (and critical symbols could be protected entirely). Eventually, self-organization reaches a stable point, in which no changes occur except in response to external perturbation.

Self-organization proceeds one coder at a time, beginning with all symbol and inflection coders disabled in both the speaker and the listener. Whenever a new coder is needed, the disabled coders compete to be the next enabled. In the speaker, allocation proceeds in an orderly manner, with a learning target of one symbol coder and one inflection coder. When the current coder matures or fails, a new one is allocated, until there are no disabled coders left. In the listener, allocation is in response to unrecognized activity on the communication paths (as reported by the cable-heads) or in a pattern, and there may be many immature coders.

The self-organization process is driven by babble generated by the speaker: the babble includes the current target, but is otherwise random. As the collection of mature elements grows, the size of the average babble message grows as well, ultimately reaching maximum-size messages. Sending multiple symbols makes it possible to identify problematic interference between symbols and reallocate to try to work around it.

Because the speaker chooses which coders are used, there is no way for the listener to distinguish between an unmatched coder and one that is merely rarely used. Accordingly, mature coders will occasionally join the competition when the supply of disabled coders is low, and the listener needs an oversupply of coders compared to the speaker in order to avoid thrashing when almost all coders are allocated. We will thus give the speaker an oversupply of o_s times as many of each set of coders.

The speaker inflection coders adapt stochastically. Each one initializes to a random pattern. Active inflections send patterns to one another over their all-to-all network, and if two patterns overlap by at least d_i/m_i pulses, one of the inflections is randomly chosen to reinitialize itself with a new random pattern. With sparsely encoded patterns, this converges rapidly to a set of encodings with high noise margins.

All other coders align by comparing subsequent patterns of activation, using a codetector to decide whether each possible component is part of the pattern (the parameters are set at $accept = 5$, $reject = 0$, $rail- = -50$, $rail+ = 20$, $miss = -2$). When enough of its components are accepted (d_s for symbols, d_i for inflections), a coder matures.

The inflection task is easy, since the incoming pattern is filtered by passing through symbol consensus. Symbol coders, on the other hand, are attempting to rendezvous with their complement on a small subset of their communication paths; listener symbol coders push a single bit of feedback up their chosen communication paths to the speaker to enable this rendezvous. The rendezvous is generally very small compared to the number of cable-heads a symbol coder connects to, since it must be possible to connect any arbitrary pair of symbol coders.

As a result, we must take careful steps to avoid coders becoming entangled with one another. When feedback arrives at a speaker cable-head, it relays the pattern is transmitted so that symbols can ignore contaminated feedback information. When a successful rendezvous contains too many communication paths, they must be pruned away one by one down to a maximum size s_+, in case a coder has connected to multiple coders on the other side. Pruning needs to be slow to assure that one coder remains connected.

Finally, any coder connected to a number of communication paths below activation threshold (d_s or d_i) deallocates itself and resets. The oversupply in the listener means that there are many opportunities for any speaker coder to find a complement, so if one attempt at alignment fails, the next is likely to succeed.

Allocation requests are drawn from the pool of mature symbol and inflection coders. If this pool is empty, then the request fails. Deallocation requests reset the element, breaking its alignment and returning it to the disabled state for fresh alignment, likely to a different element.

C.2.3 Configuration Parameters

There are a total of nine configuration parameters: p, b, k, c, s_+, d_c, d_s, d_i, and o_s.

C.2.4 Dossier

Rather than explore all the combinations of configuration parameters in detail, we will start by reducing the number of parameters.

First, any pair of symbol coders is expected to rendezvous at $r = k^2/c$ communication paths that they share. Let us take the symbol parameters as small steps away from this rendezvous size: $d_c + 1 = d_s = s_+ - 1 = r - 2$. We may likewise take the inflection detection threshold from its encoding $d_i = p - 1$.

Sparse coding capacity is discussed in [4]: the upshot is that there are many reasonable values for p, b, r and c that will allow many symbols and inflections to be encoded with little interference.

We thus are assured reasonable values of all configuration parameters except for o_s, which needs exploration.

Regarding the four criteria for desirable behavior: clean allocation is assured by the self-organization process. The other three we will need to determine experimentally.

Learning Rate First, we will test whether self-organization is fast enough to make vocabulary available for use.

- **Conditions:** Vocabulary size is $s = 1000$ symbols and $i = 20$ inflections. Message bounds are $m_s = 5$ symbols and $m_i = 2$ inflections per symbol.

- **Configurations:** The oversupply multiplier is $o_s = 1.1$ for the listener. Inflections are encoded with $p = 11$ pulses in $b = 60$ time-slots in a burst, with a detection threshold $d_i = 10$; symbols are encoded on $c = 1000$ communication paths, each symbol connecting to a subset of $k = 100$, a maximum rendezvous size of $s_+ = 9$, detection threshold of $d_s = 8$, and consensus threshold of $d_c = 7$.

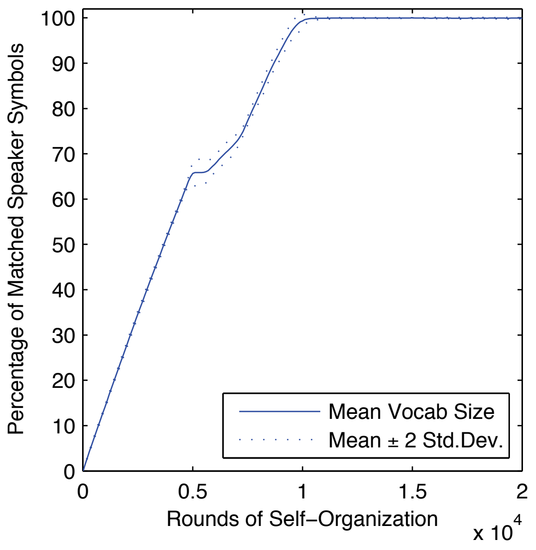


Figure C-4: When vocabulary size is far from capacity, the mature vocabulary grows rapidly. After an initial period of fast linear growth, symbols that failed to rendezvous in their first attempt find complements at a slower rate.

- **Experiment:** In each of 10 trials, the link is allowed to self-organize for 20,000 rounds. The number of mature symbols and inflections is measured once every 100 rounds.
- **Results:** The results for symbols are plotted in Figure C-4; inflections mature almost immediately. When the system is far from capacity, self-organization is fast. Vocabulary growth is approximately linear, slowing as symbols that failed to connect on their first try begin to retry.

Excess Capacity The next question we will address is how to set the oversupply multiplier o_s.

- **Conditions:** Vocabulary size is $s = 1000$ symbols and $i = 20$ inflections. Message bounds are $m_s = 5$ symbols and $m_i = 2$ inflections per symbol.
- **Configurations:** The oversupply multiplier o_s assumes nine values: 1.0, 1.01, 1.02, 1.05, 1.1, 1.2, 1.5, 2, and 5. All other parameters are set as before.
- **Experiment:** In each of 10 trials, the link is allowed to self-organize for 20,000 rounds. During the last 2,000 rounds of self organization, once every 100 rounds a set of 100 random max-size messages is sent, and the rate of perfect transmission recorded.

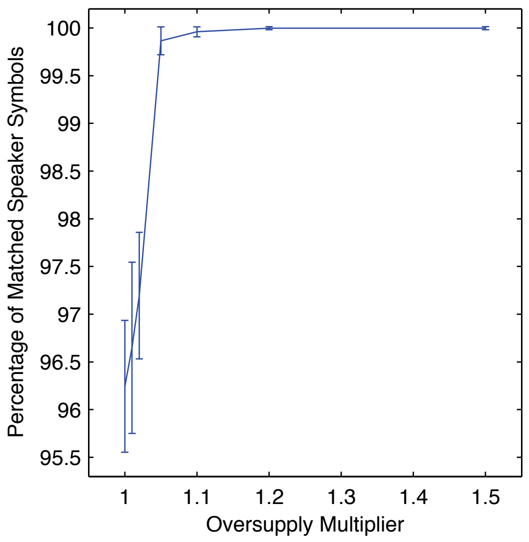


Figure C-5: A small constant factor of coder oversupply is sufficient for efficient self-organization. This graph shows the percentage of 1000 symbols that have found matches after 20,000 rounds of self-organization at various levels of oversupply (vertical bars show twice standard deviation). Inflections are always fully matched.

- **Results:** Figure C-5 shows terminal symbol vocabulary size; inflections are always fully matched. As can be seen, a small constant factor of coder oversupply in the listener is sufficient for efficient self-organization:

Graceful Degradation The last pair of experiments demonstrate the desirable properties of persistent vocabulary and consistent transmission by showing that transmission quality decays gracefully when the messages are assailed by extrinsic noise or symbol-to-symbol interference.

- **Conditions:** Vocabulary size is $s = 1000$ symbols and $i = 20$ inflections. Message bounds are $m_s = 5$ symbols and $m_i = 2$ inflections per symbol. Noise is added to every communication path with probability n of disrupting each individual time slot (adding a pulse where there is none or removing an existing pulse). Nine levels of noise n are considered: 0.0001, 0.0003, 0.001, 0.003, 0.01, 0.015, 0.02, 0.025, and 0.03.

- **Configurations:** All parameters are set as for the learning rate test.

- **Experiment:** In each of 10 trials, the link is allowed to self-organize for 20,000 rounds. During the last 2,000 rounds of self organization, once every 100 rounds

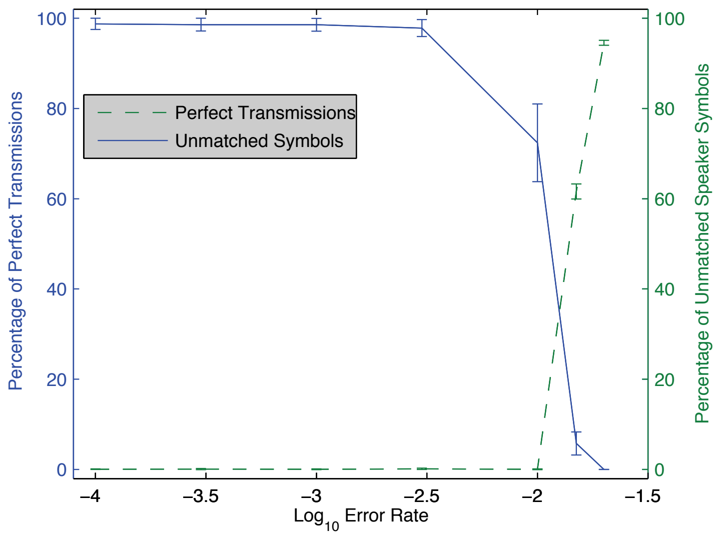


Figure C-6: Noise has minimal impact until around 1% noise, when noise is high enough to disrupt rendezvous during self-organization. This graph shows the final behavior for 20,000 rounds of self-organization.

a set of 100 random max-size messages is sent, and the rate of perfect transmission recorded. At the end of the trial the number of mature symbols and inflections is recorded.

- **Results:** A small amount of noise on the communication channel does not affect either self-organization or normal message transmission. Figure C-6 shows that noise has minimal impact at low levels, then causes a dramatic collapse in performance at around 1% noise, where the noise bits begin to inhibit symbol rendezvous during self-organization.

- **Conditions:** Vocabulary size is $s = 1000$ symbols and $i = 20$ inflections. Message bounds are $m_s = 5$ symbols and $m_i = 2$ inflections per symbol.

- **Configurations:** The number of communication paths c assumes six values: 100, 200, 300, 400, 500, and 1000. The number of paths connected to each encoder is adjusted to $k = \sqrt{10c}$, keeping the expected rendezvous size constant. All other parameters are set as for the learning rate test.

- **Experiment:** In each of 10 trials, the link is allowed to self-organize for 20,000 rounds. During the last 2,000 rounds of self organization, once every 100 rounds a set of 100 random max-size messages is sent, and the rate of perfect trans-

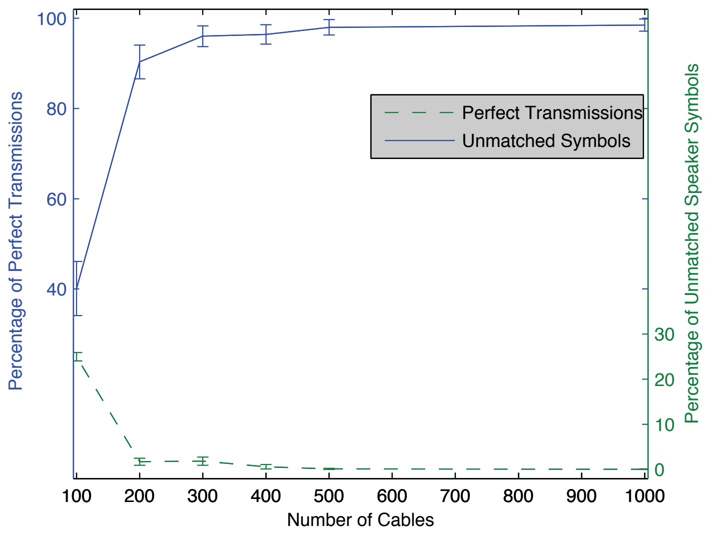


Figure C-7: As the number of communication paths in the channel decreases, self-organization gradually declines in effectiveness, eventually degrading badly. This graph shows the final behavior for 20,000 rounds of self-organization.

mission recorded. At the end of the trial the number of mature symbols and inflections is recorded.

- **Results:** Figure C-7 shows self-organization success for different sizes of channel. When the channel is smaller, symbols interfere with one another more: past a critical threshold, self-organization runs more slowly and gradually collapses

Although this dossier is only a rough survey, it gives us reason to expect desirable behavior from the unidirectional link over a broad range of conditions and configurations.

C.2.5 Usage Specification

- Configuration Policy: Parameters chosen according to the method in the dossier section will function well. A broader policy is beyond the scope of this investigation.
- Limiting Conditions: High noise will prevent self-organization, and if not enough time is allocated for self-organization, symbols will not be available when needed. Even under ideal conditions, however, it is expected that coders will occasionally spontaneously deallocate due to problems during self-organization.

- Failure Simplification: Assuming that noise is low enough to allow self-organization to succeed, most difficulties can be handled by the simple expedient of deallocating the elements involved.

  When the elements are next organized, they will likely be paired up differently and use different encoding patterns, making the chance of a recurrent failure low unless one of the coders is itself bad. If the link is set up to provide more symbols and inflections than strictly necessary, then the next allocation of a coder is unlikely to provide the same bad coder, and the bad coders will eventually end up unused.

Cost Development time is dominated by the time to build the random bipartite graph connecting the symbols to the channel, which is never worse than the number of symbols. The set-based architecture means that encoding the hardware is constant cost.

The time to send a message is dominated by the length of the pulse sequences; the time of all other operations is no worse than amortized constant.

There are three significant sources of hardware complexity: storage of the pulse sequences, which is $O(ib)$, the crossbar connecting symbols to inflections, which is $O(si)$, and the random bipartite graph connecting symbols to communication paths, which is $O(sk)$. All others are dominated by these three, but it is not guaranteed that any one of these three will dominate, so we add them to get the hardware complexity.

Finally, variation during development will results in small variations in set size or the bipartite graph. The only dangerous defects are in the crossbar: any non-functional crossbar junction means that either the symbol coder or inflection coder connected to it will not be able to be used effectively.

	Development	Mature
Time	$O(s)$	$O(b)$ amortized
Space	$O(1)$	$O(ib + si + sk)$
Imperfection	extra or missing links, coders, DOA coders	noise, lost or extra message elements

C.3 Bidirectional Link

A bidirectional link connects two specialists and allows messages to be simultaneously sent and received using the same set of symbols and inflections.

C.3.1 Interface Specification

Conditions There are three interfaces between a bidirectional link and each of the two specialists it connects. One is a boolean signal that tells the link whether it is free to engage in self-organization, another handles the allocation and deallocation of the link's vocabulary, and the remaining one is for sending and receiving messages.

A link has a potential vocabulary of s symbols and i inflections, where there are many more symbols than inflections ($s >> i$). Messages are specified as a set of symbol/inflection pairs, where there may be up to $m_s << s$ symbols and $m_i << i$ inflections per symbol. Any symbol may be combined with any inflection.

An allocation request asks for a symbol or inflection, but does not specify which particular symbol or inflection; a deallocation request specifies which particular symbol or inflection is to be deallocated.

Range of Behavior Messages are reported to a device when they arrive.

An allocation request either returns a newly allocated symbol or inflection or else reports failure. A deallocation request reports only completion.

Desirable Behavior A bidirectional link is behaving desirably when there is:

- **Available vocabulary:** up to s symbols and i inflections can be allocated for use in composing messages.
- **Clean allocation:** a newly allocated symbol or inflection is not entangled in any way with other currently allocated symbols or inflections or with those that have been deallocated in the past.
- **Persistent vocabulary:** once allocated, a symbol or inflection remains consistently usable until deallocated.
- **Consistent transmission:** there is a one-to-one mapping from sent to received symbols and inflections, and under this mapping the message received is always equivalent to the one sent.

C.3.2 Mechanism

A bidirectional link is built around two unidirectional links, one in each direction. The mechanism is explained fully in the main text in Section 7.1.3, so I will not reproduce the description here.

C.3.3 Configuration Parameters

There are eight configuration parameters for a bidirectional link: five for the codetectors it uses to make decisions, max_{rtt}, p_{gen}, and r_{ub}.

C.3.4 Dossier

We begin by eliminating parameters. The rails of the codetectors should simply be placed far enough away that they are unlikely to matter. Our data sheet for the codetector also tells us that the *accept* and *reject* thresholds should not matter much, so long as they are moderate in size.

We want max_{rtt} to be as low as possible, and if the two sides run at approximately the same rate, then we need never set max_{rtt} above 3—two for a round trip, plus a third to compensate for any small differences in timing.

That leaves us three values needing experimental investigation: *miss*, p_{gen}, and r_{ub}. We will determine these using two experiments.

Generation and Miss First we will determine reasonable values of *miss* and p_{gen} using idealized unidirectional links. The reason for using the idealized links is to make the computational cost of the survey affordable.

- **Conditions:** The number of potential symbols s and inflections i is assumed to be infinite.

- **Configurations:** *rail*+ and *rail*− are set out at infinity, $max_{rtt} = 3$, $accept = 5$, and $reject = -5$. p_{gen} ranges from 0.01 to 0.99 in steps of 0.02 and *miss* ranges from -5 to -1/5 in steps of 1/5.

- **Experiment:** Ten trials were run; in each trial, the link was allowed to self-organize for 10,000 rounds. Every 1000 rounds, 100 test messages of five symbol/inflection pairs were looped from each side, recording the number of messages that successfully returned to their sender without error. At the end, the vocabulary sizes were recorded.

- **Results:** The results are summarized by the chart shown in Figure C-8, where p_{gen} rises from top to bottom and *miss* drops from left to right.

  The color of each pixel indicates the link's behavior with the conditions and configuration for that location in the chart, using two components of its color:

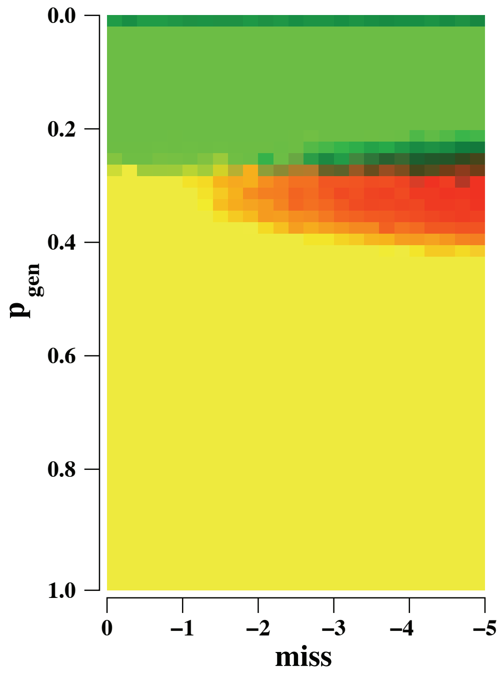


Figure C-8: Simulation of a bidirectional link with idealized unidirectional links shows that a low p_{gen} leads to successful self-organization, and that the $miss$ penalty is largely irrelevant. Red indicates significant transmission errors, green indicates fast acquisition of vocabulary, yellow indicates fast acquisition of faulty vocabulary, and black indicates failure to communicate.

- Red indicates significant transmission errors. Intensity is calculated from the average percentage of imperfect messages, calibrated so that zero imperfections is zero intensity and 25% imperfections is full intensity.
- Green indicates fast vocabulary acquisition. Intensity is calculated from the average final number of elements (symbols and inflections) in each side's vocabularies, calibrated with zero intensity at zero entries and full intensity at 100 entries.

As can be seen, a p_{gen} of approximately 0.2 or lower leads to successful self-organization and $miss$ is largely irrelevant.

Self-Organization Ratio With these values in hand, we can investigate our parameter value, r_{ub}, using a system built with real unidirectional links.

- **Conditions:** Vocabulary size is $s = 1000$ symbols and $i = 20$ inflections. Message bounds are $m_s = 5$ symbols and $m_i = 2$ inflections per symbol.

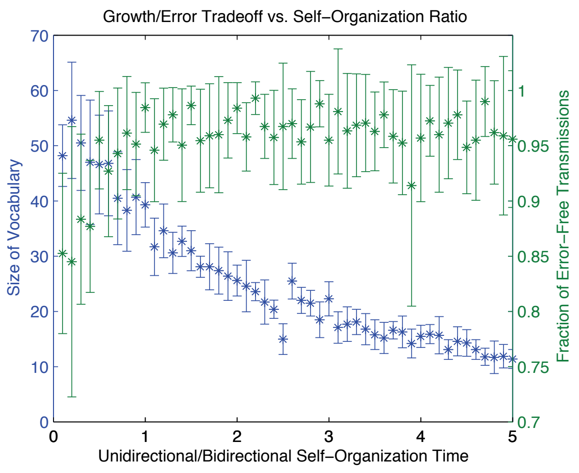


Figure C-9: More self-organization time needs to be devoted to the unidirectional links than to the bidirectional link: the lower the ratio, the faster the bidirectional link vocabulary grows, but below around 2:1 unidirectional to bidirectional ratio, the error begins to increase.

- **Configurations:** The parameters are set as before, except that $miss$ is -2, $p_{gen} = 1/10$, and r_{ub} ranges from 1/10 to 5 in steps of 1/10. The unidirectional link parameters are set as for the learning rate experiment in that dossier.

- **Experiment:** The link was allowed to self-organize for 5000 rounds. Self-organization swapped between the unidirectional and bidirectional links no faster than once every 100 rounds, with the ratio between block sizes equal to r_{ub}. Once every 500 rounds, 100 test messages were looped from each side, as before, and the final vocabulary sizes recorded.

- **Results:** Figure C-9 compares speed and success of self-organization for various values of r_{ub}. We see that the lower the ratio, the faster the bidirectional link vocabulary grows, but below around 2:1 unidirectional to bidirectional ratio, the error begins to increase.

All told, we have seen that a bidirectional link is able to behave desirably over a broad range of parameters.

C.3.5 Usage Specification

- Configuration Policy: The ratio of unidirectional to bidirectional self-organization r_{ub} needs to be at least 2:1, and the generation of call-and-response messages should be no faster than about $p_{gen} = 0.2$ and should be significantly lower to be safe.

- Limiting Conditions: Misbehavior will occur, as for a unidirectional link, if self-organization is disrupted by noise or not allowed enough time before symbols and inflections are needed.
- Failure Simplification: As for a unidirectional link, failure simplification of the bidirectional link is shaped by the likelihood that a bad pairing, once discarded, will not be reformed. The panacea is thus to deallocate any problematic symbols or inflections, with the assumption that the next allocated to fill the need is unlikely to also be flawed.

Cost The cost of the bidirectional link is largely dominated by the two unidirectional links it contains. The one exception is in the hardware cost, where the distributed map connecting the two sets of symbols may dominate if s is large.

	Development	Mature
Time	O(s)	O(b) amortized
Space	O(1)	O($ib + si + sk + s^{3/2}$)
Imperfection	extra or missing links, coders, DOA coders	noise, lost or extra message elements

C.4 Distributed Focus

Distributed focus allows a group of devices to balance the goals of maintaining consensus on a set of focus topics and allowing each device an equal chance to quickly steer the consensus.

C.4.1 Interface Specification

Conditions Distributed focus operates on a network of n devices, each containing a set of f foci (Figure C-10). Each of the foci contains a topic that is currently the subject of attention. There is no constraint on the structure of the network. Each device has a stream of local requests for attention arriving, which it may service or discard. Finally, devices update periodically at roughly the same frequency as one another, and transmit the contents of their foci to their neighbors after each update.

Range of Behavior The observable behavior of distributed focus is contents of the foci on the various devices.

Desirable Behavior Distributed focus is behaving desirably when it exhibits:

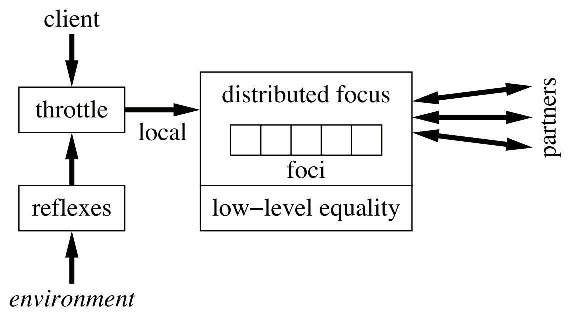


Figure C-10: Distributed focus operates on a network of n devices, each of which contains a set of f foci. Each device decides the contents of its foci based on local requests for attention (dotted arrow) and the contents of its neighbors foci. The goal is for the devices to reach consensus on a set of topics, yet allow any device to quickly change that consensus.

- **Dominance:** There are always a few topics that are dominant, occupying foci throughout the network.
- **Fairness:** Any local request is likely to become a dominant topic
- **Agility:** Dominant topics can shift quickly
- **Stability:** Topics are dominant for long enough to allow devices to work together.

C.4.2 Mechanism

The mechanism is explained fully in the main text in Section 8.2.1, so I will not reproduce the description here. I will, however, add two notes not included in the main text.

p_{ext} There is another mechanism besides privilege that might aid convergence. The other option is to give each topic request originating with a neighbor only a probability p_{ext} of being serviced, thus limiting the likelihood of such a request continuing to propagate. We will see, however, that this is not worth using and end up just setting $p_{ext} = 1$.

Replacement Policies We will consider three replacement policies: **round-robin** (foci are replaced sequentially), **random** replacement, and **follow**. Under the follow policy, if a request originated with a neighbor, then if it is possible, the topic will replace the same topic that the neighbor replaced; if not, the choice is random.

C.4.3 Configuration Parameters

There are five configuration parameters: p_{wait}, t_{priv}, k_p, p_{ext}, and *policy*.

C.4.4 Dossier

As we begin building the dossier, our first concern needs to be bringing the range of options under control. Since four of the five options are about resolving competition between topics, we will start by looking at convergence from a state of maximum competition.

I will not pursue this dossier to completion, but only far enough to have a good guess at how to set each of the five parameters. Further work is necessary to resolve questions about how dominance and request patterns relate to one another.

We start with a (very) little analysis. Based on Rauch's group selection work, we can predict that t_{priv} should be set to a value higher than the network diameter, in order to ensure propagation throughout the network. We do not want it to be much higher than diameter, though, since privilege fixes a topic unshakably in place while it runs.

We further expect at least two phases of behavior: the desirable phase, *fast convergence*, is characterized by a rapid settling into a state where a few topics are dominant throughout the foci. The undesirable phase, *thrashing*, is characterized values "chasing each other" through the network, resulting in the system rapidly cycling from state to state.

Privilege Survey We will now do a restricted survey for t_{priv}, intended to confirm our analysis and allow us to eliminate a set of options.

- **Conditions:** The number of devices n takes five values: 2, 5, 10, 20 and 50, while the number of foci per node f takes on four values: 1, 2, 4, and 8. The devices are connected with six types of graph: a four-connected grid and random graphs with connection probability 0.2 to 1.0 in steps of 0.2. Finally, updates either take place synchronously or in a random sequence; in the case of the random sequence, the results of each device update are available to the next device that updates. There are a total of 240 combinations.
- **Configurations:** t_{priv} ranges from 1 to 30 in steps of 1. The other parameters are fixed at plausible values: $p_{ext} = 1$, $k_p = 2$, $p_{wait} = 0.5$, and the follow policy.
- **Experiment:** For each of the 7200 combinations, we run ten trials. At the beginning of each trial, every focus in every device is changed to a different

topic, meaning that there are initially nk competing topics. Update cycles (in which every device updates once) are then run until no device changes or every device has updated 1000 times, whichever comes first.[2] Convergence time is then taken to be the number of updates before trial terminates.

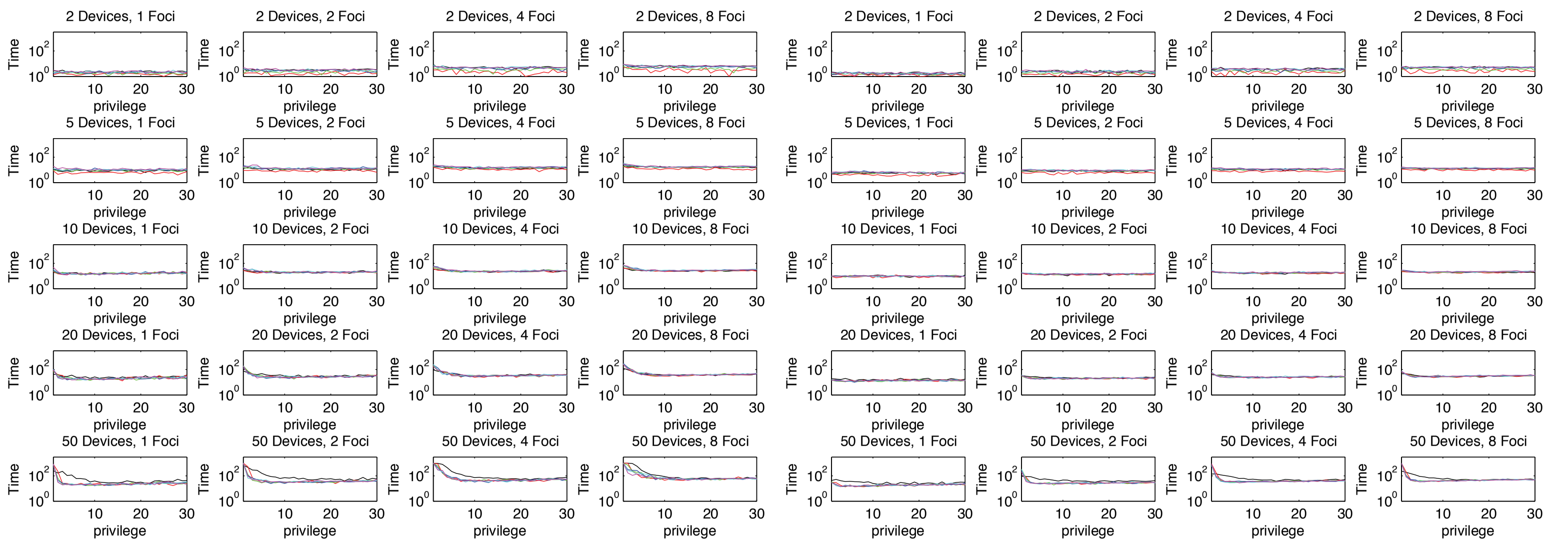
- **Results:** To results of the survey are summarized in the tableau shown in Figure C-11. Convergence time starts high and drops rapidly as t_{priv} increases, levelling off at to consistent low level at a threshold proportional to the diameter of the network. The strength of the effect depends on the number of nodes in the network. Random updates are faster to converge than synchronous updates.

We can now effectively eliminate t_{priv}, since we know how to set the value safely. Hereafter, we will always use a value of $t_{priv} = 20$, since the largest diameter networks we consider are expected to have diameter 15.

p_{ext} **Survey** Our next target for elimination is p_{ext}. Although I had originally believed this would be an important part of letting the system converge, this is a dangerous parameter because it stands directly opposed to the goal of spreading topics throughout the network. The next survey will be to see how much p_{ext} actually changes the convergence time. Even if it helps, if the benefit is small, then we will fix it at $p_{ext} = 1$, effectively eliminating the parameter, since that puts the least impediment to the spread of topics.

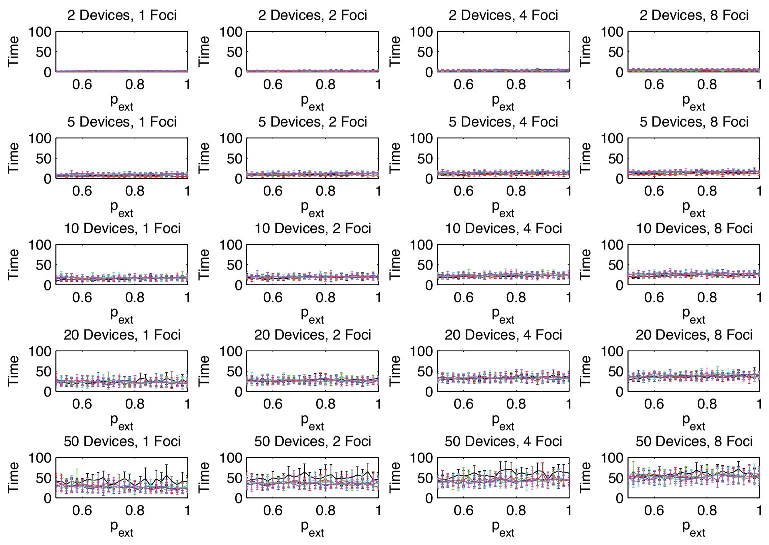
- **Conditions:** Conditions range over the same 240 combinations as in the privilege survey.
- **Configurations:** The value of p_{ext} ranges from 0.5 to 1.0 in steps of 0.02. The other parameters are fixed at plausible values: $t_{priv} = 20$, $k_p = 2$, $p_{wait} = 0.5$, and the follow policy.
- **Experiment:** For each of the 6240 combinations, we run ten trials. The trials measure convergence time, the same as in the privilege survey.
- **Results:** The tableau in Figure C-12 summarizes the results of the survey. The value of p_{ext} has no significant effect on the convergence time, so we may safely eliminate it by setting it always to a value of 1.

[2]No device changing does not mean no device will change in the future (because of the action of p_{wait}), but it is usually close.

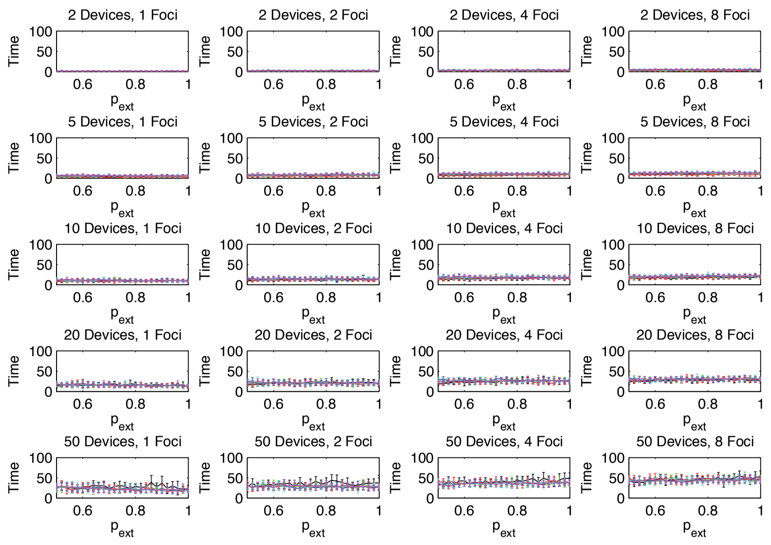


(a) Synchronous Update (b) Random Update

Figure C-11: Tableau showing the results of a survey of distributed focus privilege duration against convergence time. Convergence time starts high and drops rapidly as t_{priv} increases, levelling off at to consistent low level at a threshold proportional to the diameter of the network. The strength of the effect depends on the number of nodes in the network. Random updates are faster to converge than synchronous updates.



(a) Synchronous Update



(b) Random Update

Figure C-12: Tableau showing the results of a survey of distributed focus p_{ext} against convergence time. The value of p_{ext} has no significant effect on convergence time, so we may feel safe in eliminating the parameter.

Convergence Survey With t_{priv} and p_{ext} out of the way, we can take a closer look at the relationship between the other three parameters and convergence time.

- **Conditions:** Conditions range over the same 240 combinations as in the privilege survey.
- **Configurations:** p_{wait} ranges from 0.05 to 0.85 in steps of 0.05, k_p ranges from 0.25 to 4.0 in steps of 0.25, and we try all three replacement policies, for a total of 816 combinations.
- **Experiment:** For each of the combinations, we run ten trials. The trials measure convergence time, the same as in the previous surveys.
- **Results:** Figure C-13 shows a tableau of charts summarizing the results of the survey. Within each chart, p_{wait} ranges from 0.05 to 0.85 top to bottom and k_p ranges from 0.25 to 4 left to right.

  I expect the follow policy to give the best consensus later, so I will give it primacy in the visualization, comparing to see where either of the other policies are significantly better. Thus, the colors of the individual charts are as follows:

  – The red component shows the mean convergence time for the follow policy, scaling linearly from full intensity at 0 cycles to zero intensity at 100 cycles.
  – The green component shows when the random policy is better than the follow policy. The comparison is scaled by the sum of standard deviations, from zero intensity at 1/2 sum of deviations to full intensity at 3/2 sum of deviations.
  – The blue component shows when the round-robin policy is better than the follow policy. The comparison is scaled by the sum of standard deviations, just as for the green component.

Inspecting the results, we see that random updates are generally slightly faster than synchronous updates, but that there is no qualitative change in behavior between the two.

At low numbers of devices, the interaction between pairs of devices is the limiting factor, and so higher levels of p_{wait} produce better convergence times (though the different is small in absolute terms). Since the numbers are small, even moderate variation shows up on the chart as a sparse dusting of yellow and magenta pixels.

In the upper left hand corner of the synchronous charts, there is often a dark "thrashing hole"—this is the result of having both p_{wait} and k_p low, effectively disabling both short and long-range symmetry breaking.

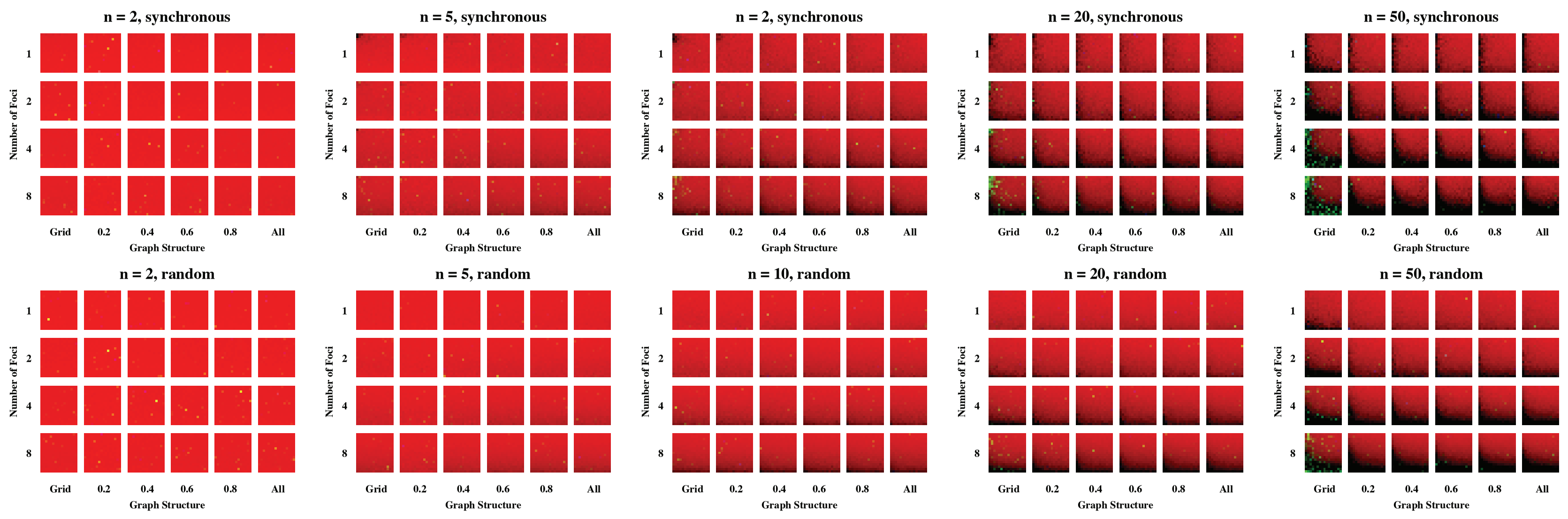


Figure C-13: Tableau showing convergence time for distributed focus. Within each chart, p_{wait} ranges from 0.05 to 0.85 top to bottom and k_p ranges from 0.25 to 4 left to right. Red intensity shows the speed of convergence for the **follow**, policy, green shows significantly better performance by **random** policy, and blue shows significantly better performance by **round-robin** policy.

For low numbers, having either symmetry breaking strategy suffices. As the numbers of devices and foci rise, however, we see that it convergence begins to fail when either p_{wait} is high or k_p is very low. What this is telling us is that the two symmetry-breaking strategies are actually interfering with one another. The reason is that as we increase p_{wait}, we are decreasing the effective diameter of privilege, since it takes longer on average to make each hop. We do need p_{wait} for short-range symmetry breaking, so what this is telling us is that we want to keep it low, since short-range symmetry breaking can resolve much faster than long-range.

Considering the three strategies, we can see that overall the round-robin strategy is never clearly better than the follow strategy—there are a few scattered instances, but not enough to form a pattern. The random strategy does perform better than the follow strategy, but only when numbers of devices and foci are high and privilege is largely disabled.

Interestingly, a detailed inspection of a few surveys suggests that almost all of the competing topics are eliminated very quickly. The bulk of the convergence time is spent eliminating the last few competing topics, when they are widespread and "chasing each others' tails" around the network. This is exactly what Rauch's work would lead us to expect, and is interesting because it suggests that the impact of competition on behavior is largely unaffected by the number of competing topics.

C.4.5 Usage Specification

- Configuration Policy: p_{ext} should be disabled (e.g. set to 1), t_{priv} should be approximately the diameter of the network, any of the three replacement strategies is reasonable, and k_p should be relatively high, while p_{wait} is low but non-zero.

- Limiting Conditions: When requests arrive faster than the system can converge, there is no hope of stabilizing to a set of dominant topics. Determining the threshold where this becomes true requires more detailed surveying.

- Failure Simplification: The most problematic failure is a lack of dominant topics, since that failure is difficult for any given device to detect. This failure mode can be avoided by throttling the rate of requests, such that convergence is always likely. This effectively sacrifices a small amount of fairness, but need not sacrifice agility, as we shall see in the design of the request throttle.

Cost Since the number of foci is expected to be low, no great pains have been taken to ensure that the distributed focus is efficient. In particular, since the set of foci is a precise set, each must be constructed separately to ensure there is no variance in

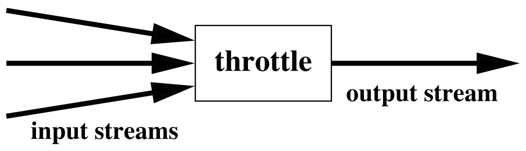


Figure C-14: A throttle takes several streams of events and filters them to produce a fairly mixed output stream with a limited overall rate, while allowing short-term bursts of activity from particular sources.

the size of the set, thus costing encoding and development time linear in the number of foci.

During operation, a comparison can be done in parallel on nf^2 hardware per node, but the actual changes are applied serially to the foci.

This fragility also means that variation during development, if it does occur, is likely to disrupt the operation of a focus entirely. At run time, however, all that will happen is a perturbation of the contents of a particular node's foci.

	Development	Mature
Time	$O(f)$	$O(f)$
Space	$O(f)$	$O(n^2f^2)$
Imperfection	DOA foci	perturbation of contents

C.5 Throttle

A throttle filters several streams of events fairly down to a single, rate-limited stream of events.

C.5.1 Interface Specification

Conditions Events arrive at the throttle in streams from s different sources. The rate and distribution of the incoming streams is unconstrained. The throttle then emits a filtered stream of events, to be constrained to a sustained rate of no more than r_s events per time unit and a burst rate of no more than b_{max} events per source.

Range of Behavior The throttle's behavior is the stream of events it emits.

Desirable Behavior A throttle is behaving desirably when it is:

- **Responsive:** when the overall rate is low, every event input enters the output stream quickly.

- **Lock-less:** a previously quiet source can always invade a stream dominated by other, more active sources.
- **Limited:** neither bursts nor the sustained rate are too large
- **Non-stuttering:** sustained activity on the inputs ends up producing sustained (rather than bursty) activity in the output.
- **Fair:** no source is denied an equal share of the total permitted activity. If a source does not use its share, however, it can be reapportioned among others.

C.5.2 Mechanism

The throttle is implemented with a simple mechanism closely related to a Token Bucket Filter[11]. Each source stream has an associated activity level. When an event arrives, the activity level is checked: if it is more than $b_{max} - 1$, the event is discarded, otherwise the event is serviced—sent to the output stream.

When an event is serviced, the activity level raised by one plus a random number in the range $[-c_{var}/2, c_{var}/2]$, where c_{var} is a constant range of cost variation. This variant cost works to prevent stuttering by keeping event servicing from falling into a regular rhythm.

Finally, activity slowly leaks away. Every activity above zero decreases at a rate of r_s/n_{active}, where n_{active} is the number of sources with positive activity. Thus, the total activity of the system decreases at rate r_s, no matter how many sources are active.

Thus, when the a source is idle its activity settles to zero, banking away the ability to transmit a burst when it next activates. When a source is active, the rate that activity leaks away dictates how often its events can be serviced, and that rate scales inversely proportional to the number of active sources, effectively controlling the overall rate.

C.5.3 Configuration Parameters

There is only one configuration option, the value of c_{var}.

C.5.4 Dossier

The token-bucket mechanism guarantees that transactions are lock-less, and a moderate c_{var} (say, above 0.2) is likely to disrupt any rhythmic patterns of activity. Our survey will thus attempt to determine whether it is also responsive, limited, and fair.

- **Conditions:** the maximum rate r_s assumes four values, 1/100, 1/30, 1/10, and 1/3; the number of sources s is 2, 4, or 8 and b_{max} is fixed to 4. Sources generate events at an average rate r ranging from 1/100 to 1/2 in steps of 1/100. Half of the sources are high rate and the other half low rate, the ratio between rates set by a parameter *skew* ranging from 1 to 10 in steps of 1/5.

- **Configurations:** c_{var} assumes four values, 0, 0.2, 0.5 and 1.

- **Experiment:** For each of the combinations of conditions and configuration, I run a single trial in which the throttle runs for 10,000 rounds. In each round, a fast source has a probability $\frac{2r \cdot skew}{skew+1}$ of generating an event, and a slow source has a probability $\frac{2r}{skew+1}$. I collect data on how many high and low rate events are generated (h_{in}, l_{in}) and how many are actually serviced (h_{out}, l_{out}).

- **Results:** First, the size of the output stream is well controlled to the goal for the throttle (the target rate, plus a single transient of size b_{max} in each channel). Of all the trials, 95% produce a number of output events at or below the goal; the remaining 4% are mostly at high values of c_{var} and at worst are only 8% above the desired rate. Not a single $c_{var} = 0$ trial is above the desired rate. All told, we can be confident that the throttle produces an appropriately limited stream of events.

  The rest of the results are summarized by the tableau shown in Figure C-15. Each of the four square blocks shows the results for one value of c_{var}; within a block, the number of sources s rises from right to left and the target rate r_s rises from top to bottom. Finally, in each chart, r rises from left to right and *skew* rises from top to bottom.

  The color of each pixel indicates the behavior of the throttle with the conditions and configuration for that location in the tableau, using the three components of its color:

  - Red indicates saturation of the output. Intensity is the ratio of actual rate to target rate ($\frac{h_{out}+l_{out}}{10000 r_s + b_{max} s}$), calibrated so 4/5 is zero intensity and 1 is full intensity.
  - Green indicates unfairness. Intensity is calculated as $1 - max(l_{in}/l_{out}, l_{out}/h_{out})$, calibrated so 1/2 is full intensity and zero is zero intensity.
  - Blue indicates full service of input events. Intensity is the average service for low and high, $\frac{l_{in}/l_{out}+h_{in}/h_{out}}{2}$, calibrated to 4/5 is zero intensity and 1 is full intensity.

This survey shows that the throttle makes a smooth transition between responsive service, in which all events are served when the rate is low (solid blue regions), and fair service, in which each source is apportioned an equal share of the limited output stream of events (solid red regions). Only on the boundary between the two regions is neither goal fully served, though even here the service is reasonable. Finally, the only significant unfairness that occurs is in the region where the low-rate sources are in transition from sparse to saturated service, an equivalent boundary and therefore not worrisome.

Finally, we can see that the larger the c_{var}, the rougher the transition between sparse and saturated usage. This will motivate us to choose moderate values of c_{var}, to disrupt rhythms without spreading the transition too much.

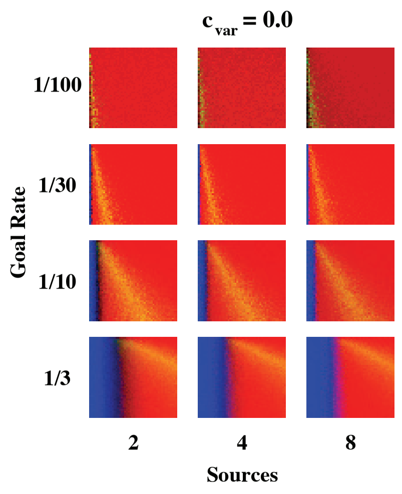
C.5.5 Usage Specification

- Configuration Policy: c_{var} should be set to a moderate value such as 0.2, large enough that patterns are likely to be disrupted but small enough to avoid spreading the transition between sparse and saturated behavior much.
- Limiting Conditions: Performance is only significantly impaired when a source is generating events at approximately the same rate that they can be serviced.
- Failure Simplification: The misbehavior of this device is mild and easy to compensate for: no further failure simplification is necessary.

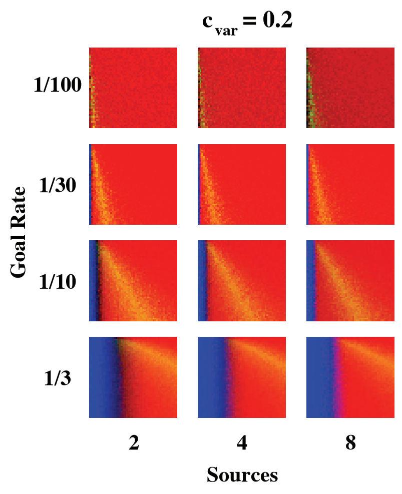
Cost Each source can be handled in parallel, except for the one central activity measure, which can be calculated in logarithmic time. Execution time is thus the minimum of this calculation and the number of elements in a burst. The hardware required is proportional to the number of sources. Errors in the mature system will lead to events occasionally slipping through or being blocked when they should not have.

While it can no doubt be implemented more concisely, a naive implementation of the central activity still takes only development time and encoding linear in the number of sources. With this naive encoding, however, the only development failure that may occur is an entire missing source.

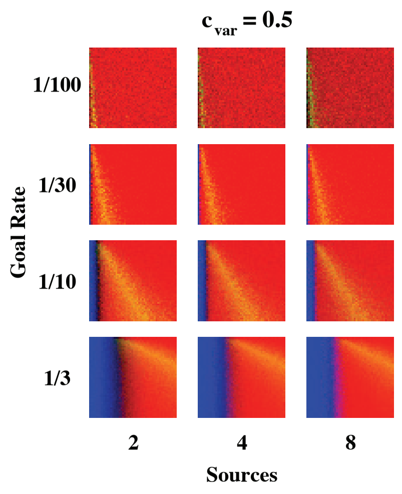
	Development	Mature
Time	$O(s)$	$O(\log s + b_{max})$
Space	$O(s)$	$O(s)$
Imperfection	missing source	rate perturbations



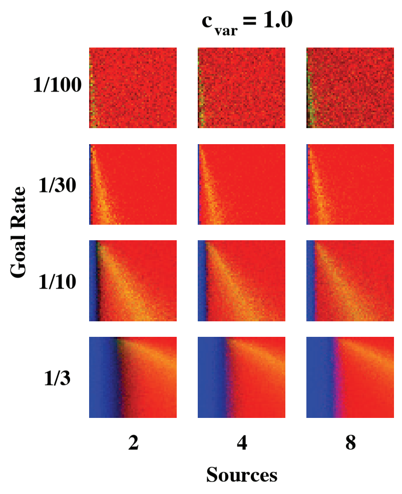
(a) $c_{var} = 0.0$



(b) $c_{var} = 0.2$



(c) $c_{var} = 0.5$



(d) $c_{var} = 1.0$

Figure C-15: Tableau of throttle behavior survey: within each chart, arrival rate increases left to right and skew between slow and fast sources increases top to bottom. For each pixel, the blue component shows full service of events, red shows saturation of the output capacity, and green shows unfairness in service. Except at the transition points between low and high utilization (for either the slow or fast population) the behavior is as desired, and the misbehavior in the transition regions is not severe.

Bibliography

[1] James F. Allen. Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26(11):832–843, 1983.

[2] John Batali. The negotiation and acquisition of recursive grammars as a result of competition among exemplars. In Ted Briscoe, editor, *Linguistic Evolution through Language Acquisition: Formal and Computational Models*, chapter 5. Cambridge University Press, 2002.

[3] Jacob Beal. An algorithm for bootstrapping communications. In *International Conference on Complex Systems (ICCS 2002)*, 2002.

[4] Jacob Beal. Generating communications systems through shared context. Master's thesis, MIT, 2002.

[5] Jacob Beal. Sidestepping impossibility: Combat consensus in the assassins' guild. In *MIT CSAIL Student Workshop 2006*, September 2006.

[6] F.P. Brooks. No silver bullet: essence and accident in software engineering. In *Proceedings of the IFIP Tenth World Computing Conference*, pages 1069–1076, 1986.

[7] William Butera. *Programming a Paintable Computer*. PhD thesis, MIT, 2002.

[8] Susan Carey. Bootstrapping and the origin of concepts. *Daedalus*, pages 59–68, Winter 2004.

[9] Sean B. Carroll. *Endless Forms Most Beautiful*. W.W. Norton, 2005.

[10] F. Chong, E. Brewer, F.T. Leighton, and Jr. T. Knight. Building a better butterfly: The multiplexed metabutterfly. In *International Symposium on Parallel Architectures, Algorithms, and Netw orks*, pages 65–72, 1994.

[11] David D. Clark, Scott Shenker, and Lixia Zhang. Supporting real-time applications in an integrated services packet network: architecture and mechanism. *SIGCOMM Comput. Commun. Rev.*, 22(4):14–26, 1992.

[12] M. Coen. *Multimodal Dynamics: Self-Supervised Learning in Perceptual and Motor Systems*. PhD thesis, MIT, 2006.

[13] Gregory R. Ganger, John D. Strunk, and Andrew J. Klosterman. Self-* storage: Brick-based storage with automated administration. Technical Report CMU-CS-03-178, Carnegie Mellon University, 2003.

[14] Seth Gilbert and Nancy Lynch. Brewer's conjecture and the feasability of consistent, available, partition-tolerant web services. *Sigact News*, 2002.

[15] R.V. Guha and D.B. Lenat. Cyc: A midterm report. *AI Magazine*, 1990.

[16] Daniel Hein. Simloid - evolution of biped walking using physical simulation. Master's thesis, Institut fur Informatik, Humboldt Universitat of Berlin, 2007.

[17] Linda Hermer and Elizabeth Spelke. Modularity and development: the case of spatial reorientation. *Cognition*, 61:195–232, 1996.

[18] Linda Hermer-Vasquez, Anne Moffett, and Paul Munkholm. Language, space and the development of cognitive flexibility in humans. *Cognition*, 79:263–299, 2001.

[19] Linda Hermer-Vasquez, Elizabeth Spelke, and Alla Katznelson. Sources of flexibility in human cognition: Dual-task studies of space and language. *Cognitive Psychology*, 39:3–36, 1999.

[20] J. Kephart and D. Chess. The vision of autonomic computing. *IEEE Computer Magazine*, 2003.

[21] Simon Kirby. Language evolution without natural selection: From vocabulary to syntax in a population of learners. Technical report, Language Evolution and Computation Research Unit, University of Edinburgh, 1998.

[22] Simon Kirby. Learning, bottlenecks and the evolution of recursive syntax. In Ted Briscoe, editor, *Linguistic Evolution through Language Acquisition: Formal and Computational Models*, chapter 6. Cambridge University Press, 2002.

[23] Simon Kirby. Natural language from artificial life. *Artificial Life*, 8:185–215, 2002.

[24] Thomas F. Knight and Gerald Jay Sussman. Cellular gate technology. In *First International Conference on Unconventional Models of Computation (UMC98)*, 1998.

[25] N Koenig and A Howard. Design and use paradigms for gazebo, an open-source multi-robot simulator. In *Intelligent Robots and Systems, 2004. (IROS 2004)*, pages 2149–2154, 2004.

[26] Teuvo Kohonen. *Self-Organization and Associative Memory.* Springer-Verlag, Berlin, 3rd edition, 1989.

[27] C. Lebiere and J.R. Anderson. A connectionist implementation of the act-r production system. In *Fifteenth Annual Conference of the Cognitive Science Society*, pages 635–640, 1993.

[28] Marvin Minsky. K-lines: A theory of memory. Technical Report AI Lab Memo 516, MIT, June 1979.

[29] Calvin Mooers. Putting probability to work in coding punched cards: Zatocoding. Technical Report Technical Bulletin No. 10, Zator, 1947.

[30] Erik M. Rauch, Hiroki Sayama, and Yaneer Bar-Yam. Dynamics and genealogy of strains in spatially extended host-pathogen models. *Journal of Theoretical Biology*, 221:655–664, 2003.

[31] Registry of standard biological parts. `http://parts.mit.edu` (visited 5/31/2007).

[32] Martin Rinard, Cristian Cadar, Daniel Dumitran, Daniel M. Roy, Tudor Leu, and Jr. William S. Beebee. Enhancing server availability and security through failure-oblivious computing. In *6th Symposium on Operating Systems Design and Implementation*, 2004.

[33] Deb Roy. *Learning from Sights and Sounds: A Computational Model.* PhD thesis, MIT, 1999.

[34] Yossi Sheffi. *The Resilient Enterprise: Overcoming Vulnerability for Competitive Advantage.* MIT Press, 2005.

[35] Push Singh, Thomas Lin, Erik T Mueller, Grace Lim, Travell Perkins, and Wan Li Zhu. Open mind common sense: Knowledge acquisition from the general public. In Robert Meersman and Zahir Tari, editors, *On the Move to Meaningful Internet Systems 2002: DOA/CoopIS/ODBASE 2002*, volume 2519 of *Lecture Notes in Computer Science*, pages 1223–1237. Springer-Verlag, Heidelberg, 2002.

[36] Russel Smith, Geoff Carlton, Frank Condello, Norman Lin, Martin C. Martin, Tim Schmidt, Konstantin Slipchenko, Jeffrey Smith, Vadim Macagon, Adam D. Moss, Erwin de Vries, Nate Waddoups, and David Whittaker. Open dynamics engine (version 0.7). `http://www.ode.org`, 2001 to 2006.

[37] Elizabeth Spelke. What makes humans smart? In D. Gentner and S. Goldin-Meadow, editors, *Advances in the Investigation of Language and Thought.* MIT Press, 2003.

[38] L. Steels. Emergent adaptive lexicons. In P. Maes, editor, *SAB96*, Cambridge, MA, 1996. MIT Press.

[39] Shimon Ullman. *High-Level Vision*, chapter "Sequence Seeking and Counter-Streams". MIT Press, 1996.

[40] D.L. Waltz. Understanding line drawings of scenes with shadows. In Patrick .H. Winston, editor, *The Psychology of Computer Vision*, pages 19–92. McGraw-Hill, 1975.

[41] Y. Wang and J.E. Laird. Integrating semantic memory into a cognitive architecture. Technical Report CCA-TR-2006-02, University of Michigan, 2006.

[42] R. Weiss, S. Basu, S. Hooshangi, A. Kalmbach, D. Karig, R. Mehreja, and I. Netravali. Genetic circuit building blocks for cellular computation, communications, and signal processing. *Natural Computing*, 2(1):47–84, 2003.

[43] R. Weiss and T. Knight. Engineered communications for microbial robotics. In *Sixth International Meeting on DNA Based Computers (DNA6)*, 2000.

[44] K Wolff and P Nordin. Learning biped locomotion from first principles on a simulated humanoid robot using linear genetic programming. In *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2003*, 2003.

[45] Holly Yanco. Robot communication: issues and implementations. Master's thesis, MIT, 1994.