ABSTRACT

Title of dissertation: A COGNITIVE ROBOTIC IMITATION LEARNING SYSTEM BASED ON CAUSE-EFFECT REASONING

Garrett Ethan Katz
Doctor of Philosophy, 2017

Dissertation directed by: Professor James A. Reggia
Department of Computer Science

As autonomous systems become more intelligent and ubiquitous, it is increasingly important that their behavior can be easily controlled and understood by human end users. Robotic imitation learning has emerged as a useful paradigm for meeting this challenge. However, much of the research in this area focuses on mimicking the precise low-level motor control of a demonstrator, rather than interpreting the intentions of a demonstrator at a cognitive level, which limits the ability of these systems to generalize. In particular, cause-effect reasoning is an important component of human cognition that is under-represented in these systems.

This dissertation contributes a novel framework for cognitive-level imitation learning that uses parsimonious cause-effect reasoning to generalize demonstrated skills, and to justify its own actions to end users. The contributions include new causal inference algorithms, which are shown formally to be correct and have reasonable computational complexity characteristics. Additionally, empirical validations both in simulation and on board a physical robot show that this approach can ef-

ficiently and often successfully infer a demonstrator's intentions on the basis of a single demonstration, and can generalize learned skills to a variety of new situations. Lastly, computer experiments are used to compare several formal criteria of parsimony in the context of causal intention inference, and a new criterion proposed in this work is shown to compare favorably with more traditional ones.

In addition, this dissertation takes strides towards a purely neurocomputational implementation of this causally-driven imitation learning framework. In particular, it contributes a novel method for systematically locating fixed points in recurrent neural networks. Fixed points are relevant to recent work on neural networks that can be "programmed" to exhibit cognitive-level behaviors, like those involved in the imitation learning system developed here. As such, the fixed point solver developed in this work is a tool that can be used to improve our engineering and understanding of neurocomputational cognitive control in the next generation of autonomous systems, ultimately resulting in systems that are more pliable and transparent.

A COGNITIVE ROBOTIC IMITATION LEARNING SYSTEM BASED ON CAUSE-EFFECT REASONING

by

Garrett Ethan Katz

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2017

Advisory Committee:
Professor James A. Reggia, Chair
Professor Yiannis Aloimonos
Professor Rodolphe J. Gentili
Professor Jeffrey W. Herrmann
Professor Dana S. Nau

© Copyright by
Garrett Ethan Katz
2017

Acknowledgments

This work was supported financially by ONR award N000141310597 and a seed grant from Lockheed Martin Corporation. Previous versions of some content in Chapters 2, 4, 5, and 7 were published in [67–71], all papers on which I was first author and to which I made the primary contributions.

To my classmates - Konstantinos Zampogiannis, Jason Filippou, Mahfuza Sharmin, Michael Maynord, and Tommy Pensyl: Thank you for providing the spice of life, getting me out of the house, and keeping me balanced. To my former roommates, Raegan and Kevin Conlin: thank you for so many years of warmth and friendship. You and your family are an inspiration.

To my fellow research assistants over the years - Charmi Patel, Dale Dullnig, Baram Sosis, Josh Langsfeld, Hyuk Oh, Isabelle Shuggi, Theresa Hauge, Gregory Davis, and Di-Wei Huang: thank you for your fruitful collaborations and comradery. Thanks especially to you, Di-Wei, for showing me the ropes and always bringing me back gifts from your trips to Taiwan. It was a small thing but it said a lot.

To all of the professors at University of Maryland with whom I was a student or a teaching assistant - Aravind Srinivasan, Howard Elman, Héctor Corrada Bravo, Jandelyn Plane, James Reggia, David Jacobs, Yiannis Aloimonos, Michael Hicks, Dave Levin, Dana Nau, Giovanni Forni, and David Van Horn: Thank you for sharing your time and knowledge, and making me a better learner and teacher. Thanks especially to Jandelyn Plane, for giving me the opportunity to be a summer course instructor. Special thanks also to Howard Elman and Dana Nau for generously

agreeing to be on my preliminary exam committee, and to Giovanni Forni, for the insightful feedback on the content of Chapter 7. And thanks to Dana Nau, Yiannis Aloimonos, Rodolphe Gentili, and Jeffrey Herrmann for taking an interest in my work and agreeing to be on my dissertation committee.

Rodolphe, thank you for being so generous and hospitable, letting me use your lab day in and day out. Thank you also for your guidance, and for always valuing my opinion and treating me as if I were your peer - so much so that I sometimes forgot how much farther I have to go before that point. You always suffered through my less self-aware moments with grace.

Jim, thank you for dutifully performing the demanding and at certain times thankless job of being my advisor. You have always been patient, encouraging, and remarkably devoted to my well-being and success, as you are with all of your students. You gave form to my vague notions and taught me what research was, always steering me towards the right path. It has made all the difference.

Lastly, Al and Paul, thank you for giving me the opportunity to first do research back at City College, and for supporting me in my move to Maryland to pursue a doctoral degree. And to Mom, Dad, Adria, and the rest of my family: Thank you for your unconditional love and support. Everyone should be so lucky.

Table of Contents

List of Tables

List of Figures

List of Abbreviations

AI	Artificial Intelligence
CPG	Causal Plan Graph
FSN	Minimum Forest Size
FSX	Maximum Forest Size
GTN	Goal-Task Network
GUI	Graphical User Interface
HGN	Hierarchical Goal Network
HTN	Hierarchical Task Network
IFT	Inverse Function Theorem
IL	Imitation learning
IR	Irredundancy
MC	Minimum Cardinality
MD	Minimum depth
MP	Minimum Parameters
MPC	Monroe Plan Corpus
PCT	Parsimonious Covering Theory
P&T	Pliability and Transparency
RNN	Recurrent Neural Network
XAI	Explainable Artificial Intelligence
XD	Minimax depth

Chapter 1: Introduction

1.1 Problem Statement

Recent years have seen an explosion of interest in Artificial Intelligence (AI) and autonomous systems. Some of the largest corporations, federal agencies, and research institutions in the world are devoting extensive resources to these technologies. Self-driving cars, drones, humanoid robots, and other autonomous systems are slowly pervading our streets, skies, factories, hospitals, battlefields, and homes.

This momentum notwithstanding, AI researchers and engineers know first-hand just how over-sensationalized, fragile, and stubbornly unintelligent today's autonomous systems can be. Manually programming robots to exhibit any form of autonomous behavior is a painstaking and time-consuming process, even for highly trained roboticists. This makes it extremely difficult for end users to control or guide the behavior of an autonomous system, at any substantial level of intelligence beyond rote mimicry of desired movements. The issue is confounded by the increasing use of deep neural networks for autonomous control (e.g., [77]), which are often very opaque even to their architects - let alone end users - and designed and tuned largely through trial and error. This can severely limit both the architect's and the

end user's ability to control, understand, predict, or trust an autonomous system's behavior.

The current AI focus on deep learning performance, at the expense of end user control and understanding, has prompted vocal warnings from some prominent technologists and scientists [54,86]. These warnings sometimes fall flat, as AI historians know that the lead time to human-level AI has been sorely underestimated in the past [31]. Our aspiration is a world filled with human-level autonomy and beyond, but we may not expect to be around when that world comes. This expectation makes it easy to forget the associated risks and luxuriate in our scientific enterprise. But we read the news, and watch the movies - and for all their hype, a small doubt begins to tug on our assuredness that AI is so far off. If we were to come face to face with that object of our aspiration, and lock eyes with our human-level AI: what would we feel then?

No one can foretell with certainty the impacts of widespread intelligent autonomy. It could eliminate jobs, but could also create new jobs or fill labor shortages. It could reduce accidents from human error, but could also cause new accidents stemming from imperfect software. AI world domination or mass extinction events lie at the more dubious and sensationalized end of the spectrum. Nevertheless, even without a catastrophic event, there is still the profound existential risk that human civilization might gradually evolve into one with completely inorganic constituents, upending the distinction between natural and artificial.

True human-level AI is necessarily unpredictable, to the extent that humans are unpredictable. Unpredictable AI is endlessly fascinating, but given the high

stakes, it does not belong in the upcoming generation of autonomous systems. This dissertation is based on the premise that tomorrow's autonomous systems should meet at least the following criteria:

- *Pliability*: It should be easy for end users with no robotics training to control and guide an autonomous system's behavior.
- *Transparency*: An autonomous system's behavior should be well understood by the architects at a technical level, and by end users at a practical level.

These two criteria are henceforth abbreviated as P&T.

A relevant paradigm for P&T is *robotic imitation learning*, wherein a robot imitates a human demonstrator and thereby acquires a new behavior. Since a human is in the loop, guiding robotic learning, the system is highly pliable and subsequent behavior will be relatively predictable and understandable. This can improve user safety, as well as trust in and adoption of the robotic system. However, much of the existing work on imitation learning focuses on sensorimotor behavior, with little attention to cognition. This limits the ability of these systems to generalize to new situations. *Cause-effect reasoning* is a particularly important cognitive faculty that is under-represented in these systems. As such a quintessential and familiar cognitive faculty in humans, it stands to reason that causal inference mechanisms could significantly improve generalization ability in imitation learning systems and simultaneously promote P&T.

Based on this hypothesis, this dissertation presents a novel cognitive robotics system that uses **C**ause-**E**ffect **R**easoning for **I**mitation **L**earning, called CERIL.

Initialized with some prespecified causal background knowledge, CERIL can use causal inference to interpret human demonstrations at a cognitive level and generalize to new situations, much as human learners do. CERIL works by hypothesizing plausible intentions on the part of the demonstrator, which serve as parsimonious explanations for the actions that were observed.

As an imitation learning system, CERIL is highly pliable. It can generalize a learned skill on the basis of just a single end user demonstration. Moreover, the generalization can extend to situations requiring significantly different actions from what was demonstrated, such as using different arms to reach different objects in different locations, handing off objects between grippers, and so on. Consequently, end users can rapidly shape a robot's behavior at a cognitive level.

CERIL is also highly transparent, at both the technical level and the end user level. At the technical level, CERIL's symbolic algorithms for causal intention inference, which were developed as part of this work, come with formally verified correctness and complexity characteristics, so they are well understood. At the end user level, CERIL can leverage its causal knowledge to provide intuitive human-readable justifications for what it has planned to do at imitation time. This can potentially make the system more understandable for non-expert human users.

CERIL's theoretical analysis is backed by several experimental results. An empirical validation both in simulation and on board a physical robot demonstrates CERIL's high success rate and capacity for generalization. Additional computer experiments are used to compare several formal criteria of parsimonious explanation in the context of intention inference, revealing their relative strengths and weaknesses.

In particular, it is shown that a novel parsimony criterion proposed in this work is often favorable as compared with more traditional criteria. Lastly, computer experiments are used to quantify CERIL's ability to produce intuitive human-readable justifications for its planned actions. In particular, it is shown that CERIL's causal knowledge may admit combinatorially many valid justifications for any particular planned action, but a parsimony criterion can reduce this set of justifications to a small, intelligible subset.

CERIL's cognitive-level reasoning is currently implemented using symbolic computation, which is amenable to formal analysis and promotes transparency. However, a neural reimplementation could potentially improve CERIL's performance, adaptability, and capacity for learning new causal knowledge from experience that was not provided during initialization. In particular, future versions of CERIL could incorporate recent approaches to "programmable" neural networks, that can encode symbolic cognitive-level processing using neural attractor dynamics [122]. However, these attractor dynamics remain poorly understood from a theoretical perspective, rendering the resulting systems fairly opaque and unpredictable. To maintain transparency in a neural reimplementation of CERIL, it is important that neural attractors be better understood at a technical level. To that end, this dissertation also contributes a novel mathematical method for systematically locating neural attractors. This method constitutes an important new tool that can be used to more effectively study neural dynamics, both in CERIL and at large.

1.2 Objectives

The overall goal of this work was to design an effective imitation learning system based on cause-effect reasoning (CERIL). The guiding hypothesis was that causal reasoning would facilitate cognitive-level generalization as well as P&T. In that context, the following specific objectives guided this research:

1. **Design causal reasoning mechanisms for robotic imitation learning.** Specifically, design a knowledge representation and set of parsimonious cause-effect reasoning algorithms for inferring a demonstrator's intentions that support real-valued spatial information processing, causal chaining, and temporal ordering constraints. Conduct a formal analysis to establish their soundness, completeness, and computational complexity guarantees.

2. **Implement and empirically validate CERIL.** The implementation should incorporate the causal reasoning mechanisms from objective (1) on board a physical robot. Empirical validation should be conducted using a suite of assembly and maintenance tasks in a physical tabletop workspace, as well as a battery of synthetic testing data. In addition to validating performance and reliability, these empirical studies should also be used to inform the parsimony criteria employed by the causal reasoning mechanisms from objective (1).

3. **Extend CERIL so that it can explain its actions to an end user.** In particular, devise a procedure through which CERIL can query its underlying causal representation to explain why it has planned any particular action. The

procedure should provide a small, intelligible set of possible justifications that are compelling and intuitive to a human end user. This procedure should be exposed through a human-friendly interface that provides 3D visualizations of the actions, human-readable justifications for them, and a navigable graphical representation of the planning process.

4. **Develop a mathematical tool for locating attractors in recurrent neural networks.** The purpose of this tool is to aid the study of attractor dynamics in programmable neural networks. These attractor dynamics are relevant to a neural reimplementation of CERIL, so a deeper understanding of them is important for transparency. The tool should have a solid theoretical basis, and be verified in practice through empirical computer experiments.

1.3 Overview

The rest of this dissertation is organized as follows.

Chapter 2 surveys background work relevant to this dissertation. First, Section 2.1 covers work on sensorimotor- and cognitive-level imitation learning. Next, Section 2.2 covers work on causal inference that forms the basis for CERIL's reasoning algorithms, including abductive inference of potential causes (e.g., intentions) from observed effects (e.g., demonstrated actions), and automated planning based on causal relationships between robotic actions and the environment. Finally, Section 2.3 briefly reviews some work on neurocomputational autonomous control and neural attractor dynamics.

Chapter 3 presents the overall architecture of CERIL at a conceptual level, and introduces some concrete running examples. This chapter describes in concrete terms every stage of CERIL's imitation learning process as well as the causal background knowledge required by CERIL to operate. This chapter does not include any results but explains in detail how CERIL works and sets the stage for Chapters 4-7.

Chapter 4 details the results of objective 1. It presents a formal model of causal knowledge that is sufficient for intention inference in the context of robotic imitation learning, including temporal ordering constraints, causal chaining, and real-valued spatial information. It formally defines the intention inference problem and presents algorithms for solving it. Lastly, it proves theorems that guarantee the soundness and completeness of these algorithms, as well as characterizing their computational complexity.

Chapter 5 details the results of objective 2. It describes empirical results confirming that the causal inference algorithms are correct and efficient in practice. It also validates the end-to-end imitation learning pipeline on board a physical robot, showing that CERIL can generalize a variety of skills to new situations from single demonstrations, with a high success rate. Lastly, it compares several possible parsimony criteria for assessing the plausibility of inferred intentions. Which criterion is used can affect which intentions CERIL hypothesizes and tries to imitate. The empirical comparison shows that some traditional criteria are less appropriate in the context of intention inference, whereas a new criterion proposed in this dissertation is often more appropriate.

Chapter 6 details the results of objective 3. It presents a procedure with which CERIL can query its causal knowledge in order to justify its planned actions to an end user. Some preliminary experimental results show that this procedure can effectively reduce a combinatorially large set of possible justifications down to a small, intelligible subset that is suitable for human end users.

Chapter 7 describes the results of objective 4. A new method for systematically locating neural attractors is presented. Some theoretical properties of the method are proven. Empirical computer experiments show in practice that the method is competitive and complementary to existing approaches. This method constitutes a new tool that can be used in studies of programmable neural attractor dynamics that are relevant to neural implementations of systems like CERIL.

Finally, Chapter 8 concludes with a discussion of the limitations and contributions of this dissertation, as well as directions for future work.

Chapter 2: Background

This chapter reviews some past research that is relevant to the work in this dissertation. Section 2.1 surveys past approaches to imitation learning. Section 2.2 focuses on three important topics in AI that are all associated with cause-effect reasoning: abductive inference, automated planning, and plan recognition. Section 2.3 briefly summarizes some relevant past work on sensorimotor and cognitive control with neural networks and highlights common sources of limited reliability and transparency.

2.1 Imitation Learning

Over the past several decades, the commercial use of robots has become increasingly wide-spread [80]. But in most industrial applications, robots are manually programmed by human experts to execute highly specialized, repetitive tasks. Manually programming contemporary robotic systems is time consuming, difficult, and expensive, requiring a trained roboticist to make changes for even slightly altered tasks. A potential alternative is to replace manual programming with *imitation learning* (IL), in which a robotic system learns a task by watching a human perform the task, and then attempts to imitate what was observed [17]. IL holds the promise

of highly pliable intelligent agents that can be taught by human domain experts and other end users, who have little or no expertise in robotics and computer programming. IL also has the potential to produce artificial collaborators that are safer, more trust-worthy, and more usable, aiding and protecting human operators in difficult or dangerous situations, but at the same time keeping a responsible human in the loop to guide behavior.

Empirical work from cognitive science and neuroscience conducted on humans and monkeys reveals that imitation learning is an important component of our own motor repertoire [83]. In particular, humans have neural mechanisms (the mirror neuron system) that likely facilitate decoding of intentions and understanding of actions previously observed in order to infer a forthcoming new goal [65]. In other words, human imitation learning involves an understanding of the intentions of a demonstrator, in addition to their observed actions [11, 43, 53]. We refer to the understanding of a demonstrator's intentions as "high-level" or "cognitive-level" imitation learning.

Much past work on robotic IL has focused on low-level sensorimotor learning in robots, where the objective is to closely mimic the precise motor trajectories and dynamics relevant to a given skill [1, 10, 12, 16, 34, 41, 139]. For example, Fitzgerald et al. use "kinesthetic teaching" for two-dimensional block rearrangement tasks, in which humans demonstrate by physically guiding the robot arm to a target location [41]. They compare two different representations for new skills - one that uses a full trajectory, and one that uses a finite set of points along the trajectory most relevant to the skill. Wu et al. place fiducial markers on objects and on the hand of a human

demonstrator, and use standard visual processing and motion tracking methods to record the relevant trajectories [139]. Their one-shot learning framework can repeat these actions when the manipulated objects are in new locations, by using thin plate spline warping to map the demonstrated trajectory onto the new scene. Barros et al. train a teleoperated humanoid robot to walk, using joysticks designed for human feet that can transmit the forces experienced on the robot's feet in real time [12].

Methods in this category are very important and have led to impressive results, but are typically limited to a single class of relatively simple low-level tasks not requiring interactions of sensorimotor processes with high-level planning, reasoning and control. As such, they are not human-competitive on complex tasks that require a sequence of structured actions. Further, procedures learned via robot imitation are often quite brittle and do not generalize well to situations that are not in the training set, in part because most past imitation learning systems have focused on copying demonstrated actions verbatim rather than trying to "understand" the goals and intentions of the human demonstrator [24].

There is some past work on imitation learning at a higher cognitive level, although it is often restricted to simple, simulated, and/or highly constrained task scenarios, or requires many demonstrations for learning. There have been two prominent branches of research in this area. One branch works within the framework of reinforcement learning, using environments with simple state representations such as cells in discrete two-dimensional grids, or low-dimensional real-valued vectors (e.g., a ⟨position, velocity⟩ pair describing motion along a single coordinate axis) [28, 45, 79, 133]. For example, Verma and Rao approached goal imitation from

the perspective of reinforcement learning and graphical models [133]. Rather than exploring the entire search space to learn optimal action policies, exploration is focused on the regions of the search space corresponding to high-reward action sequences that are demonstrated by a teacher. The technique requires a discrete, finite set of states, and represents goals as elements in that set - for example, in the experimental evaluation, states were locations in a discretized two-dimensional grid, and goals were target locations. This approach was later extended to include hierarchical actions [45], and recently a similar technique using graphical models was trained by leveraging crowd-sourcing to generate a large number of demonstrations [27]. However all experimental evaluations were confined to discretized two-dimensional worlds. There has also been some work on neural models of goal-directed imitation learning, but focused on low-level goals such as the target position of a reaching movement [94].

The other branch uses symbolic representations of goals with internal structure [25,35,66,142–144]. For example, Chella et al. devised a framework for converting raw sensor data into structured symbolic representations, which was tested on a physical robot [24]. However, their methods still assume a highly constrained task scenario, involving the rearrangement of convex blocks in a two-dimensional plane. Another example is the work of Jansen and Belpaeme, which includes a symbolic approach to inferring a demonstrator's intentions, although they also restrict their attention to simple two-dimensional environments [66]. Their inference mechanism focuses on the final changes to symbolic object properties and relationships in the last step of a demonstration to form plausible hypotheses about the teacher's intent.

However, their system operates solely on discrete symbolic representations, not addressing the problem of real-world continuous valued sensorimotor data, and requires on the order of hundreds of demonstrations for convergence (synthetic demonstration data was generated by a computer program). More recently, Mohseni-Kabir et al. developed a framework in which humans can use natural language to explain high-level tasks to a robot, such as removing tires from a car [85]. Specifically, the human explains a high-level task (e.g., "remove the tire") as a sequence of lower-level tasks that have already been taught (e.g., "grab tire, pull off tire, put down tire"). Their system represents tasks with the "Hierarchical Task Network" formalism, which is described in more detail in Section 2.2.2. However, the hierarchy is designed manually by a user through a voice recognition interface, not inferred from a demonstration; and the question of generalization to new situations was not addressed.

In contrast, Yang et al. [142] also devised a cognitive representation based on hierarchical structure, but it was able to automatically interpret human demonstrations, rather than relying on manual construction of the hierarchy by the user. Their system was applied in more realistic and unconstrained scenarios such as household kitchen work. Subsequently it was extended to include a symbolic model of spatial object relationships [144] and to learn from unconstrained cooking videos on the web [143]. The underlying hierarchical structure of actions was based on an analogy with natural language. Demonstrations were treated as "sentences" that could be generated by a context-free grammar, and the demonstrations were parsed according to this grammar in order to form a representation with tree structure. Using that

tree structure, combined with some semantic background knowledge, the system was able to infer "hidden" consequences of some actions that were not explicitly listed in the machine representation of the demonstration.

CERIL builds on ideas in this past work, especially the use of hierarchical representations as in [85,143]. However, CERIL is differentiated from these past approaches in its emphasis on cause-effect reasoning. As explained later, this enables generalization to new situations that require significantly different low-level plans, on the basis of a single demonstration, in contrast with this past work. Moreover, CERIL comes with strong formal guarantees, and is shown empirically to be capable of handling less constrained task scenarios than most past work, involving bimanual manipulation of non-convex, composite objects in three dimensions, not only in simulation but also the physical world. This was verified with systematic experiments that quantified the success rate of the end-to-end IL process on a physical robot, which was not always done in past work. Lastly, the CERIL architecture supports a mechanism with which it can justify the actions that it plans before imitating, a feature not available in most past IL work. This leads to improved transparency for end users.

2.2 Causal Inference

There are three prominent topics in AI that involve cause-effect reasoning and are relevant to this dissertation. The first, described further in Sect. 2.2.1, is abductive inference. In CERIL, abductive inference is used to interpret demonstrations

and infer the higher-level intentions of a demonstrator. The second, described further in Sect. 2.2.2, is automated planning. In CERIL, automated planning is used by the robot to plan new actions for new situations that carry out the demonstrator's intentions. The third, described further in Sect. 2.2.3, is plan recognition. CERIL's use of abductive inference to interpret demonstrations is related to plan recognition, but is different from past work on that topic in several ways, as explained below.

2.2.1 Abductive Inference

Abductive inference, commonly known as "inference to the best explanation," is the process by which one uses cause-effect knowledge to form plausible hypotheses that explain the available evidence [95]. Familiar examples include a detective finding clues and solving a mystery, or a medical doctor examining a patient and making a diagnosis.

As compared with deductive inference, abduction is much less developed in AI. However, in the past several decades, some computational models of abduction have been proposed. Most of these models rely on symbolic representations and often employ concepts from deductive logic and automated theorem proving. The core idea behind these models is as follows. An observation is modeled by a logical proposition p. To explain the observation, the abductive inference system must provide a logical proof whose conclusion is p. The system has background knowledge in the form of several available axioms that can be used to construct the explanatory proof. Various criteria are used to evaluate an explanatory proof as "good" or

"bad." For example, Poole used the criteria that good explanations should not be presumptive (should not imply other explanations) and should be minimal (should not contain redundant antecedents) [100]. In "cost-based abduction", each axiom has an associated cost when used in a proof, and finding a good explanation is construed as finding a low-cost proof [23]. Mitchell et al. used similar proof-based representations in their work on "explanation-based generalization" [84]. In this context, the objective is to explain why (i.e., prove that) a particular instance (e.g., a yellow mug on the table) is an example of some concept (e.g., the concept of a cup), based on some predefined background knowledge, and use that explanation to classify other instances of the same concept.

Another prominent approach to abductive inference is parsimonious covering theory (PCT) [98]. In the most basic formalism, the objective is to explain one or more observed "manifestations" in terms of the "disorders" which cause them. Manifestations and disorders are modeled as vertices of a bipartite graph. Links in the graph capture causal relationships about which disorders can be used to explain which manifestations. Using M to denote the set of manifestations and D to denote the set of disorders, PCT defines a function $\textsc{causes}(m)$ which returns a subset of D, containing every disorder d with a causal link to the manifestation $m \in M$. In a particular inference problem instance, one is given a subset of M, representing the *observed* manifestations, and a valid explanation is a subset of D, called a *cover*. A cover is defined by the property that every observed manifestation is linked to at least one disorder in the cover by the edges of the graph. In other words, for every observed manifestation m, a valid cover must contain at least one

d in CAUSES(m). "Good" explanations are covers that satisfy some criterion of parsimony: for example, having small cardinality, or being irredundant (containing no proper subset that also covers the observations). The best criterion may be application dependent, and various criteria have been compared empirically in the context of diagnosing brain damage [130].

There are many extensions to the basic form of parsimonious covering theory. Maximum likelihood can be used as a parsimony criterion, by assigning edges weights that measure the "causal probability" that a disorder actually causes a manifestation (which is distinct from the more familiar notion of conditional probability) [97]. Causal graphs with more than two layers allow for causal chaining, in which the manifestations (effects) at one layer are also the disorders (causes) for the next layer [96]. Temporal information can be incorporated as well, by allowing individual disorders to explain entire sequences of manifestations subject to partial ordering constraints [135].

In the context of cognitive robotics, inferences involving spatial information are essential, but there is no unified theory concerning the incorporation of spatial information into abduction. There are several approaches specialized for certain applications, but often at geospatial scales that have limited relevance to robots. Shakarian and Subrahmanian have devised methods for "geospatial abduction problems": a class of problems involving discretized two-dimensional space, in which observed events at one set of locations must be explained in terms of hypothesized agents based at another set of locations [112]. Couclelis studied how we might explain man-made structures in urban or natural landscapes in terms of their intended

purpose [30]. An exception is the work of Shanahan, who considered the case of a mobile vacuuming robot, which needs to explain noisy sensor data in terms of a hypothesized model of the environment [113]. He devised a collection of deductive axioms which use two-dimensional geometric primitives to describe spatial relations between the robot and its environment, which can be supplied to a logic-based abduction system that uses the automated theorem prover approach.

Hierarchical causal structure has also seen somewhat piecemeal treatment in the literature. Citro et al. developed a system for neurological diagnosis which combines logic-based abduction with hierarchical spatial structure, in which a manifestation is explained by satisfying a logical clause involving brain regions, and a database of nested cubes relates volumes in space to corresponding brain regions [29]. Mozetic presents a logic-based method for handling general hierarchies which can but need not be spatial [87]. Each level of the hierarchy defines a logical model which maps hypotheses to observations they can explain. Logical predicates are also used to define how low-level hypotheses and observations can be abstracted to higher-level counterparts. The procedure operates by climbing from detailed observations to abstracted observations, reasoning backwards at the higher levels to abstracted hypotheses, and finally descending the hierarchy to obtain concrete hypotheses. However, this approach assumes that all information is encoded in logical form, and lacks a notion of an embodied agent that can interact with the world to bring about its goals.

2.2.2 Automated Planning

The field of automated planning, which centers on computational models of goal-directed reasoning, has been active for several decades. A thorough review is given by Ghallab, Nau, and Traverso [49]. As with abduction, there have been a variety of models, but the majority share a common basis which relies on top-down symbolic processing and is described here. In any particular application, a full symbolic description of the planning problem is typically provided in a knowledge base referred to as a *domain*, and the human who populates that knowledge base is referred to as a *domain author*. A classical planning domain specifies a finite set of possible *states* of the environment, which are symbolic descriptions of the world. Typically each state is represented as a list of logical propositions asserting object properties and relationships, such as `(on, block-A, block-B)`. In the following, s will be used to denote an arbitrary state. The domain also specifies a collection of *operators*, which represent choices the planner can make to transition from one state to another. Each operator accepts a list of *parameters* which determine exactly how the state will change. For example, an operator `pick-up` might accept a single parameter `block` specifying which block is picked up. Formally, each operator o is a function

$$o : S \times X^* \to S \cup \{\texttt{Failure}\}$$

where X^* is the set of all finite lists of parameter values. The output value represents the new state after the operator is applied, except in the case of `Failure`, which indicates that a given operator and parameter binding is not a valid option in the

current state. For example, picking up a wall, or picking up a block that has another block on top of it, may be invalid options that return `Failure`.

Any grounded operator, in which each parameter is bound to a specific value, is referred to as an *action*. In other words, each possible parameter binding for one operator represents a different action. Formally, each action is a function $a : S \rightarrow S \cup \{\texttt{Failure}\}$. The *postconditions* of an action refer to the resulting propositions that become true in state after the action is performed. The *preconditions* of an action refer to propositions in the current state which must obtain before the action can be used. Alternatively, an action's preconditions can be equated with the subset of states in which the action is valid.

Given a planning domain, the classical planning problem consists of an *initial state*, s_0, and a set of *goal* states, g. A solution (i.e., a plan) is any sequence of actions that ultimately transitions s_0 to some state in g, without any violated preconditions along the way.

Historically, action pre- and postconditions have been represented with logical formulae, similarly to states, and automated theorem proving techniques are used to compute new states when actions are performed. A seminal example in this vein was the STRIPS planner [39]. An alternative representation is used by PyHop[1], a more recent planner written in the Python programming language. In PyHop, states are represented with arbitrary Python data structures, and operators are represented with arbitrary Python functions. Each operator function accepts the

[1]https://bitbucket.org/dananau/pyhop

current state as input in addition to other parameters, and returns the new state as output (or an indication of failure when the operator is not a valid option in the current state).[2] PyHop is agnostic to the inner workings of state data structures and operator functions and treats them as black boxes: domain authors are free to encode states, actions, preconditions, and so on with generic programming language constructs available in Python, and need not use logic-based representations or automated theorem proving techniques.

The computational complexity is known for several variants of the classical planning problem and is generally worse than polynomial time [37]. To make automated planning more practical, one strategy is to let the human domain expert give additional guidance to the planning process, based on their knowledge of the domain. A popular formalism for this is the "Hierarchical Task Network" (HTN) [88]. HTNs introduce *tasks*, which generalize the notion of actions. A domain author defines a number of tasks and arranges them in a hierarchy, where tasks lower in the hierarchy can be combined into useful recipes for accomplishing tasks higher in the hierarchy. For example, the task of "traveling to the Bellagio hotel" might be accomplished through a sequence of three sub-tasks: "riding the metro to Dulles airport", "flying to Las Vegas", and "riding a cab to the Bellagio". The domain author might also include an alternative recipe: "renting an RV", and "driving to the Bellagio".

Formally, an HTN domain includes a number of *tasks* of the form $t\langle x_1, x_2, ...\rangle$,

[2]Unless it is a *non-primitive* operator, as described below.

where t is a unique name for the task, and $x_1, x_2...$ are parameters such as hotels or manipulable objects. Each task is associated with several alternative *recipes*, or *methods*, which can be applied to accomplish the task. Some methods are *primitive operators*: like operators in classical planning, they are transition functions that map the current state (and zero or more parameters) onto a new state, or an indication of failure. However, other non-primitive methods higher in the hierarchy are different: instead of producing a new state, they produce a sequence of sub-tasks, each with their own associated methods. This induces a recursive decomposition of tasks into sub-tasks into sub-sub-tasks, until ultimately primitive operators are reached. Tasks with more than one method allow for branching and backtracking during the decomposition process. If a decomposition branch produces a successful sequence of primitive operators, that sequence is considered to be a solution to the planning problem.

The HTN planning process is illustrated in Figure 2.1. For the sake of example the figure assumes a total ordering on the tasks at each level, although that is not required in the most general HTN variants. The planner is given a list of high-level tasks that must be accomplished, which it processes sequentially. If the current task is primitive, it is *applied* to update the current state. Otherwise, the planning algorithm *branches* to search each alternative method available for the current task. In each branch, the current method is *decomposed* into its constituent sub-tasks, and the algorithm is called on the sub-tasks recursively. If at any point the algorithm attempts to apply an action or method whose preconditions are not satisfied, it receives an indication of failure, and back-tracks to try a different branch. When

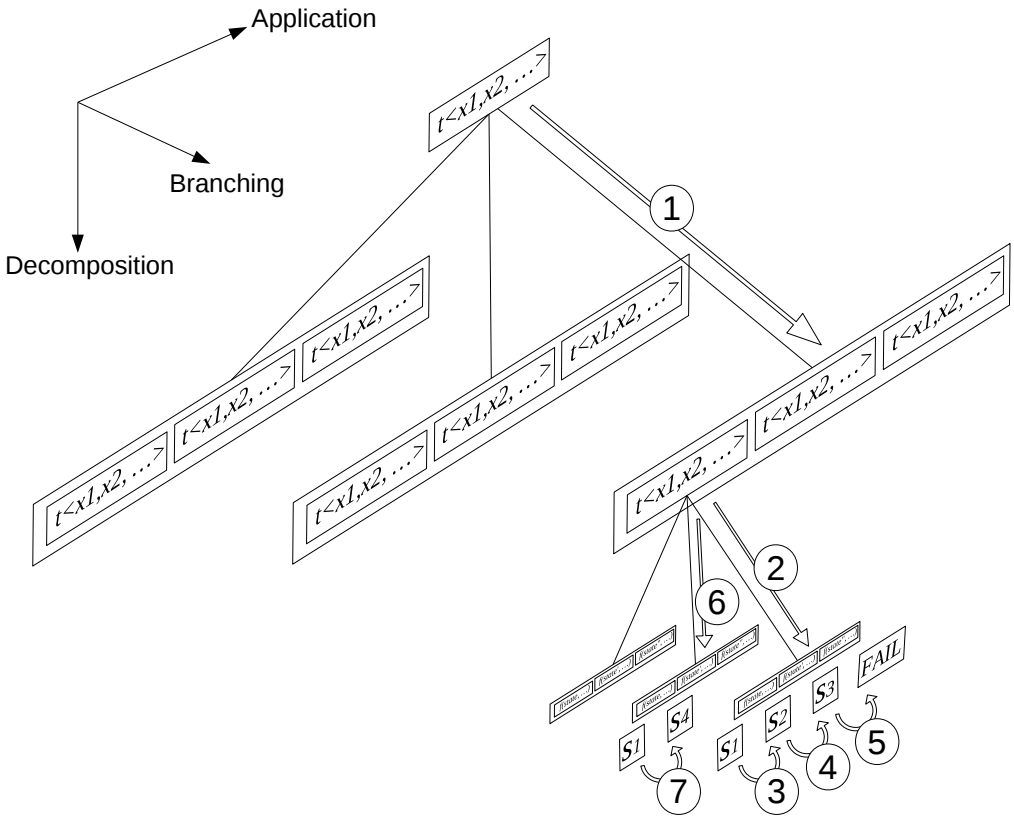

Figure 2.1: Hierarchical Task Network Decomposition. Parameterized tasks are indicated by the expression $t\langle x_1, x_2, ...\rangle$. This notation is only schematic; multiple occurrences of this expression in the figure are not necessarily the same task. The three labeled axes correspond to the main dimensions of the search algorithm: Tasks are recursively *decomposed* until primitive operators are reached, those operators are *applied* to update the state, and if failure is reached, the algorithm backtracks and tries a new *branch*. Each branch corresponds to a different method relevant to the parent task. Numbered circles indicate the sequence of steps performed by the algorithm. States are indicated by "S1"..."S4" in the boxes at bottom - note that the algorithm reverts to initial state S1 when trying the second branch, but the new branch may lead to new states.

all top-level tasks have been successfully decomposed, the algorithm terminates, returning as its solution the sequence of successful actions obtained at the bottom of the decomposition. If all available branches ultimately fail, the algorithm terminates with no solution.

HTNs provide a convenient framework for domain experts to guide the planning process. This is a significant advantage when dealing with robots, which generally lack the large body of background knowledge that humans acquire over years of development and often take for granted. On the other hand, the HTN framework places a significant burden on the domain author. Defining a suitable task hierarchy is a manual process which can require significant time, effort, and ingenuity. Since the domain author is responsible for constructing the task hierarchy, most of the reasoning takes place in the domain author rather than the automated planning process. This is particularly true when using representations like PyHop's in which states and operators are treated as black boxes whose inner logic is not interpretable to the planning algorithm. Moreover, the notion of a *task* is not included in the classical planning formalism, and as such, task representations are generally unintelligible to classical planners. So classical planning cannot be invoked as a fallback option in situations where HTN planning failed due to human errors or omissions mid-way up the task hierarchy.

To mitigate these issues, Shivashankar et al. recently introduced an analog of HTNs called Hierarchical Goal Networks (HGNs) [114–116]. The main differences are that while HTN methods return *sub-tasks*, HGN methods return *sub-goals*, and while HTN methods accept the current state as input, HGN methods accept both

the current state and the next sub-goal as input. Each sub-goal is a goal in the classical planning sense, i.e., a set of target states, represented by explicit logical propositions describing desired object properties and relationships. As a result, HGNs can leverage useful recipes when they are provided by a human, but fall back on classical planning techniques when no recipes are available. HGN methods induce a recursive decomposition of goals into sub-goals, until there are no more methods relevant to pairs of consecutive sub-goals at the bottom of the hierarchy. At this point, classical planning is used to bridge the consecutive sub-goals with primitive action sequences. The concatenation of these action sequences is the solution returned by the HGN algorithm.

Even more recently, Alford et al. introduced "Goal-Task Networks" (GTNs), which allow for a heterogeneous mix of formal tasks and goals within a single domain [5]. This can potentially enable the best of both worlds, allowing a domain author to mix task and goal methods depending on which is a more natural representation for any particular piece of background knowledge. The GTN problem has been formally defined and analyzed, but a concrete algorithm for solving it has not yet been proposed or validated.

Most work in automated planning, hierarchical or classical, focuses on an offline reasoning process and treats actions as atomic concepts, thereby failing to capture some of the complexities involved in real-world robotics. Recently Nau et al. have highlighted and elaborated on this issue [89]. The authors distinguish between *descriptive* action models, which capture *what* an action does (i.e., pre- and post-conditions in STRIPS-like classical planning), and *operational* action models,

which capture *how* an action is done (i.e., arbitrary computer programs in PyHop-like hierarchical planning, for operations like motor planning and feedback control on a robot). They present a preliminary framework, termed "Refinement Acting Engine" (RAE), which seeks to unify these two action models. RAEs bear some similarity to hierarchical planners, in that methods authored by domain experts are used to refine high-level, abstract activities into low-level, concrete operations. However, RAE methods permit complex code bodies, in which each line can either generate sub-tasks as part of the planning process or issue commands as part of the acting process. The two processes are blended using a multi-threaded, stack-based architecture.

2.2.3 Plan Recognition

Plan recognition is the problem of inferring an agent's goals and intentions, after observing the agent carry out a plan of action [72]. Plan recognition can be viewed as a form of cause-effect reasoning, and as such a number of abductive inference models have been proposed for it, including methods based on logic, parsing, set-covering, and probabilistic reasoning, as well as hybrids and heuristics [13, 22, 46, 67, 82, 118, 134]. Plan recognition has been applied in areas such as story understanding and natural language human-computer discourse [21]. On the other hand, the utility of plan recognition for human-robot interaction, and in particular robotic imitation learning, has been largely unexplored [15, 17]. As intelligent robots become a reality of everyday life, it is increasingly important that robots can

interpret the intentions and desires of the humans with whom they interact, so that they can behave accordingly.

Outside the realm of IL, there are many methods for plan recognition. Singla and Mooney augmented Markov Logic Networks (MLNs) to perform abductive inference, and used their abductive MLN technique to infer high level plans from observed action sequences [118]. A major benefit of their approach is that MLNs can combine probabilistic inference with first order logic. However, it must be trained through supervised learning on a large data set of plans, and inference is computationally expensive. Meadows et al. used a logic programming approach to perform plan *understanding*, which they defined as inferring intermediate layers of a hierarchical plan in addition to the top-level task [82]. Their method constructs explanations in a bottom-up manner from an observed action sequence, and degrades gracefully when some actions in the sequence are hidden. However, their approach has no formal guarantees, and sometimes fails to recover the correct explanation even when the full action sequence is observed. Both of these approaches were applied to plan recognition in simulated domains and it is not clear whether they would be sufficient for real-world imitation learning and execution on board a physical robot.

Saffar et al. approached the problem with a brain-inspired strategy, by mapping an HTN onto an activation-spreading network [110]. Network nodes were in one-to-one correspondence with HTN operators, and synaptic connections corresponded with parent-child task relationships. Additional connections were used to encode ordering constraints between the sub-tasks for a particular parent task, and additional nodes were used to supply contextual information. Observed low-level

actions would stimulate the corresponding network nodes, whose activity would spread to their parents, ultimately activating top-level tasks that were plausible explanations. This process could rapidly activate plausible high-level parent tasks, even before the full sub-task sequence was observed, and in real time. This method was successfully applied in conjunction with a visual processing system that worked on real-world image data as input. However, the inferred top-level tasks were never decomposed to form motor directives for a robot, and the method only applies to restricted HTN domains where a given method always returns the same sub-tasks regardless of current state (i.e., no branching and backtracking). Moreover, even though the approach is brain-inspired, the representation is still local and idealized since nodes are in one-to-one correspondence with planning operators. It is unclear how well this methodology would apply to real-world imitation learning where the inferred intentions must be ultimately decomposed into motor directives suitable for a physical robot, especially in a way that generalizes to new situations.

Li et al. reinterpreted HTNs as context free grammars, and applied grammar induction to interpret a training set of observed low-level actions [78]. Unlike the foregoing methods, their procedure does not require a predefined knowledge base of tasks, but instead constructs this knowledge base autonomously. In other words, it is an *inductive* rather than *abductive* inference mechanism. On one hand, this approach greatly relieves the burden on the domain author. On the other hand, the autonomously constructed tasks may not have intuitive meanings to human domain experts. Moreover, this approach was tested using computer experiments and not applied to robotics, so it is unclear how well this approach would translate to

real-world imitation learning, in which explicitly providing a certain level of detailed human domain knowledge goes a long way in making the many challenges of robotics more tractable.

In contrast with this past work, the CERIL architecture proposed here approaches plan recognition from the perspective of real-world imitation learning on board a physical robot. It extends PCT for plan recognition, rather than using automated theorem proving approaches, which are often more computationally expensive and in some ways more cumbersome to implement and maintain. Put differently, it uses an operational rather than descriptive encoding, which is more appropriate for the complexities of robot control. Lastly, it provides formal guarantees that all and only the valid explanations for an observed action sequence are actually inferred.

2.3 Neural Network Transparency and Control

Although CERIL is currently implemented with symbolic computation, neural networks are emerging as indispensable tools for maximally performant autonomous control. However, neural networks generally suffer from limited transparency both to experts and end users. To reconcile these conflicting aims of performance and transparency, it is important to improve our understanding of how neural networks operate - not only for a neural reimplementation of CERIL's causal inference mechanisms, but also for any other neurocomputational autonomous system. This dissertation contributes to this effort in Chapter 7. Accordingly, this section provides a brief summary of past work in neurocomputational control of sensorimotor and

cognitive processes, and how it relates to the P&T criteria.

Neural networks are simplified models of the brain that are simulated on a computer and harnessed to perform useful computations. By carefully optimizing the parameters of a neural network model, it can be "trained" to exhibit desired patterns of activity. For example, if an activity pattern encodes an array of pixel values, a neural network can learn to generate images. If an activity pattern encodes an array of joint velocities, it can be used for robotic motor control.

In recent years, large neural networks (so-called "deep learning") have come to the forefront of sensorimotor processing in artificial systems. For example, Levine et al. presented a striking use of neural computation [76]. Their end-to-end neural architecture maps raw input pixels directly to raw output torques at each joint. The network has been trained to deftly execute action policies related to tool use, such as placing a clothes hanger on a rod, screwing caps on bottles, or fitting the claw of a hammer under a nail. Similar approaches have used intensive training with physical robots to produced highly performant neural models specialized for various purposes, such as grasping a wide variety of objects [77], or predicting the expected visual field after a planned object interaction [40]. Gentili et al. developed effective strategies for general and robust motor control of upper extremities using more biologically plausible architectures, that explicitly model specific brain regions and draw on theories of motor control in humans [47,48]. This work was extended to model brain regions implicated in spatial transformations and low-level sensorimotor imitation learning [92,93]. Similar theories of human motor control were also used in a neural model based on limit cycles in self-organizing maps [63].

Historically, neural networks have been much more effective for sensorimotor control than cognitive control. Computational models of cognitive processing have most commonly been implemented using traditional, top-down symbolic programming paradigms, in which systems are explicitly programmed to manage working memory, bind variables, perform logical goal-directed reasoning, and make executive decisions. Well-known examples of general-purpose cognitive systems include the ACT-R and SOAR architectures [8, 75]. The cognitive control mechanisms in these systems are largely based around "production rules," a collection of logical *if-then* statements which define how the system should respond in any given situation. Examples of more specific cognitive functions come from AI planning and abduction, as already described in Sections 2.2.1 and 2.2.2, as well as AI theorem provers, natural language processing systems, and so on. Building neural systems capable of similar cognitive functions has proven to be very challenging. Existing approaches tend to require manual hard-wiring of the network to mimic a specific symbolic functionality, or to solve a highly specialized problem (e.g., [32, 117, 127, 128]). One exception is Neto et al.'s high-level language and "compiler" that, given any particular program, can automatically construct a network that is hard-wired to implement that program [91]. However, any such network can only perform the program from which it was compiled - a new program requires an entirely new problem-specific network. Better understanding the relationships between neural substrate, cognitive processing, and symbolic computation remains an active and important research direction [38, 99, 108].

In the past few years, deep learning researchers have proposed a number of

"programmable" neural network models for cognitive-level computation. Common themes in this work are: (1) a coupling of neural components with non-neural external resources, (2) program induction from large training sets of input-output examples, and (3) learning via gradient-based optimization. One of the first such models was the "Neural Turing Machine" (NTM) which has since been expanded to the "Differentiable Neural Computer" [51, 52]. The basic model couples a non-neural, persistent memory store with a neural Long-Short-Term-Memory (LSTM) controller [55]. An output layer of the controller modulates a soft attention mechanism that accesses memory according to differentiable formulas, making the entire system amenable to gradient-based learning. The model can successfully learn procedural skills such as recalling/sorting an input sequence. In particular, it can generalize to new sequences with lengths greater than what was presented during training. Subsequent work expanded on the NTM by combining neural controllers with other non-neural resources, such as arithmetic and logic modules [90], topic-knowledge databases [3], binary search trees [9], and program stacks [105].

While impressive, this past work suffers from limited P&T. Most deep architectures are designed iteratively through a combination of intuition and trial and error. They are treated essentially like black boxes during training, using gradient-based optimization that must be carefully monitored, and using large data sets that must be prepared beforehand, and therefore have limited pliability for end users. Once training is complete, it can be difficult to understand what has been learned and how the network produces accurate output, hindering transparency. It can also be difficult to guarantee desired behavior on new data outside of the training set,

hindering trustworthiness and user adoption.

Another recent and more biologically plausible approach to neurocomputational cognitive control is Sylvester's GALIS framework [121]. A central theme of this approach is that neural-based working memory can learn and store "programs" in the form of itinerant attractor dynamics [60]. This is a form of dynamics in which the neural state trajectory moves along an "itinerary" of nearly fixed points, settling at one before proceeding to the next. Each waypoint represents a single instruction within a sequential program. The programs are not hard-wired into the architecture, but are learned by adjusting the synaptic weights and can be "overwritten" when a new task is at hand. The GALIS framework has been applied successfully to several cognitive psychology tasks, including the n-back task, and a well known card-matching game [122, 123]. The system was evaluated in simulated environments using idealized sensory input and motor output.

An important quality of GALIS that distinguishes it from deep architectures is its emphasis on neural attractor dynamics. Neural attractors have been recognized as highly relevant to many neurocomputational phenomena, ranging from low-level motor control and tool use (e.g., [2, 136]) to high-level cognitive functions such as problem solving and decision making (e.g., [101, 103, 132]). A critical advantage of this dynamical systems perspective is that there is a large body of foundational mathematics available which can be used to improve our understanding of neural networks, although that understanding is still far from complete.

On the other hand, reliability can be an issue for the dynamical systems approach. Hopfield showed that through Hebbian learning, networks can emerge that

possess attractor dynamics relevant to a desired neural computation [58]. However, the resulting networks often possess many additional "spurious" attractors that were not present during training, and whose locations are not known a priori [20]. Limited memory capacity can also result in the networks failing to learn attractors that *were* presented during training. In relation to GALIS, this means that program encodings may be partially corrupted, resulting in behavior that deviates from the expectation of the programmer. These issues compromise the predictability and trustworthiness of GALIS and other neural systems.

Our limited ability to guarantee desired neural attractor dynamics stems from their limited transparency. In general, some of the most basic questions that characterize any dynamical system - such as how to compute the location of every fixed point - remain poorly understood and very difficult to answer in the case of neural networks. This is notwithstanding many impressive mathematical analyses in the literature. Numerous empirical and theoretical studies have provided a solid understanding of the local and global stability of fixed points (surveyed in [146]), as well as their arrangement in neural phase space, given certain conditions on the connection weights such as symmetry (e.g., [4,7]), but not a method for ascertaining the precise locations of every fixed point for arbitrary connection weights. Zeng and Wang derived remarkably fine-grained theoretical results: Given an arbitrary weight matrix, their analysis partitions the phase space into exponentially many regions, and for each region, provides sufficient conditions under which a unique locally or globally stable attractor is present [145]. However, short of a brute force approach that checks each region, which is infeasible on large networks, it is not obvious how

to efficiently and precisely locate the attractors that are actually present, avoiding regions where they are absent.

In practice, fixed point attractors are often found by repeated local optimization from random initial points (e.g., [120]). This method can find many fixed points, but is not guaranteed to find them all. To our knowledge, there is no efficient procedure that precisely locates every fixed point of recurrent neural networks with arbitrary connectivity. The closest works we have seen are generic global solvers for arbitrary dynamical systems that use bisection-based branch-and-bound search ([73, 138]), which may not scale to large recurrent neural networks.

The fixed point locations in any dynamical system are one of the most fundamental pieces of information and often a necessary first step towards understanding the system's dynamics. Therefore, developing new methods for neural fixed point location is an important research endeavor for improving transparency in neural network controllers for autonomous systems. In turn, an improved understanding of system dynamics can lead to better methods for verification, resulting in more reliable and predictable systems. This dissertation contributes a new fixed point location method in Chapter 7. This will facilitate future work on CERIL to incorporate neural networks in the causal inference mechanisms while maintaining acceptable P&T.

Chapter 3: The CERIL architecture

The main contributions of this dissertation include formal algorithms for cause-effect reasoning during imitation learning, as well as theoretical and empirical results that characterize those algorithms' properties and performance. CERIL is made possible by these algorithms and validated by these results, which are presented in subsequent chapters. However, in order to properly present these contributions, it will be helpful to first provide a conceptual overview of how CERIL works during imitation learning. The intent is to first show in an intuitive fashion what CERIL does to motivate and introduce the basic concepts before covering the technical details. To that end, this chapter presents the CERIL architecture at a conceptual level and introduces a running example that will be referenced throughout the remainder of the dissertation. Subsequent chapters provide more detail on the core components of CERIL as well as the theoretical and empirical results.

The goal of CERIL is to support cognitive-level imitation learning, capable of generalizing on the basis of a single demonstration, much as people do. The guiding hypothesis is that cause-effect reasoning is an effective vehicle to accomplish this goal. Based on this idea, I developed the framework pictured in Fig. 3.1, and described in the following. I developed new algorithms to support this framework,

established formal correctness and complexity characteristics, and conducted an empirical validation using a battery of experiments [67–69]. This chapter gives an overview of the causal knowledge incorporated in CERIL and of how CERIL functions. The algorithms are detailed in Chapter 4. The experiments use our own robotic testing environment as well as a 3^{rd} party dataset known as the Monroe County Corpus, covered in more detail in Chapter 5.

A central innovation of the CERIL architecture is the integration of two causal inference paradigms: abductive inference and automated planning. Abductive reasoning is used to infer the intentions of a demonstrator (Fig. 3.1, A). These same intentions can then be imitated in new situations (Fig. 3.1, B). Automated planning is used in those new situations to carry out the same intentions, but with potentially different actions as needed (Fig. 3.1, C). In this way CERIL is able to generalize learned skills to situations that are different from what was observed in a demonstration. A notable aspect of CERIL is the *hierarchical* arrangement of intentions and actions: any given intention can cause CERIL to carry out a sequence of sub-intentions, each of which can cause its own sequence of sub-sub-intentions, and so on until observable actions are performed. Throughout the following we refer to actions as the "lowest level" of the hierarchy, with intentions at the "higher levels" of the hierarchy. Actions are indicated schematically in Fig. 3.1 with nodes labeled "Act", and intentions with nodes labeled "Intent." This hierarchy is elucidated in concrete terms below.

The CERIL framework requires substantial background knowledge about the environment, actions, and intentions to be provided up front. The upshot is that it

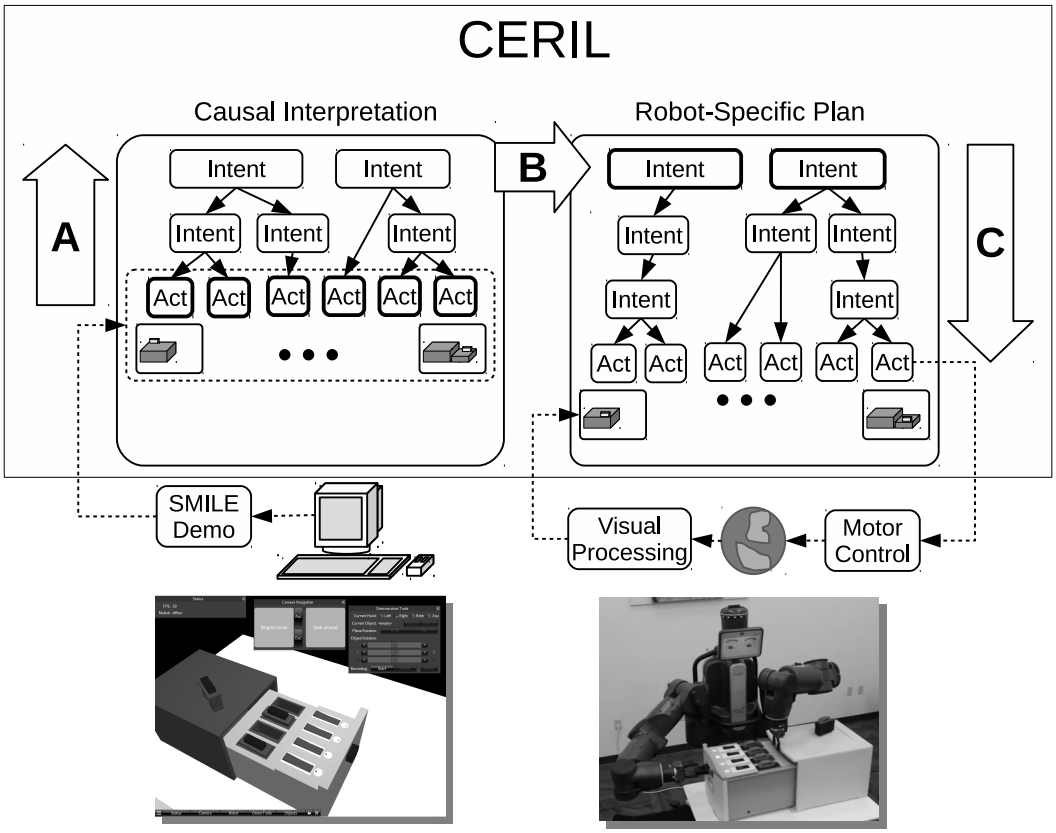

Figure 3.1: The CERIL architecture. Abductive inference is used to infer the intentions of a demonstrator (A). The same intentions can then be imitated in new situations (B). Automated planning is used to carry out those intentions in the new situation, using potentially different actions if needed (C). Solid black arrows represent causal relationships. More detail is provided in the text.

can generalize on the basis of just a single demonstration, resulting in a highly pliable system. Moreover, CERIL can leverage the rich causal background knowledge to justify planned actions to an end user, thereby promoting transparency, as detailed in Chapter 6.

CERIL also relies on peripheral components that are not part of the architecture proper. First, CERIL presupposes some external apparatus for recording human demonstrations and converting them to a machine-readable representation. For the purposes of this dissertation, demonstrations were recorded with SMILE,[1] a virtual environment developed at the University of Maryland in which a human user can click and drag objects and export a text-based event record of their actions [61,62]. A screenshot from SMILE can be seen in Fig. 3.1 (lower left). Several screenshots from a sample demonstration in SMILE are also shown in Fig. 3.2.

Second, CERIL presupposes the existence of external low-level sensorimotor controllers for the target autonomous platform. These controllers must be capable of converting raw sensory data into a machine-readable representation of the environment, and of successfully executing motor directives (e.g., computing and moving to the joint angles needed to pose a physical gripper at a position requested by CERIL). For the purposes of this dissertation, the physical platform was Baxter from Rethink Robotics™, an upper-torso humanoid robot with two 7-degree-of-freedom arms and a head-mounted Microsoft Kinect™. A photograph of Baxter can be seen in Fig. 3.1 (lower right). Several snapshots of Baxter imitating a demonstration are also

[1]Thanks to Di-Wei Huang for developing SMILE. SMILE is available for download at `https://github.com/dwhuang/SMILE`.

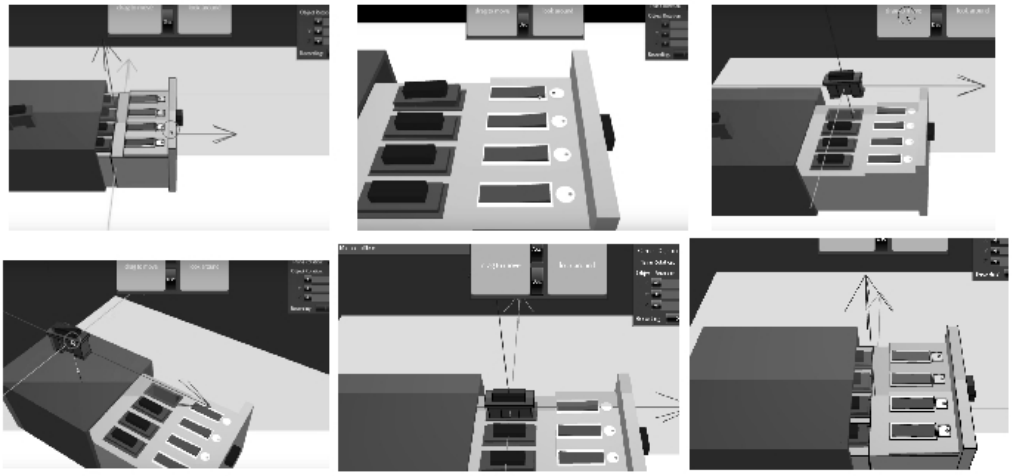

Figure 3.2: Screenshots from a SMILE demonstration. The dock is opened (top left), a red toggle is pressed (top middle), and the adjacent faulty drive is removed (top right). Then, a spare drive on top of the dock is picked up (bottom left), it is inserted in the empty slot (bottom middle), and the dock is closed (bottom right).

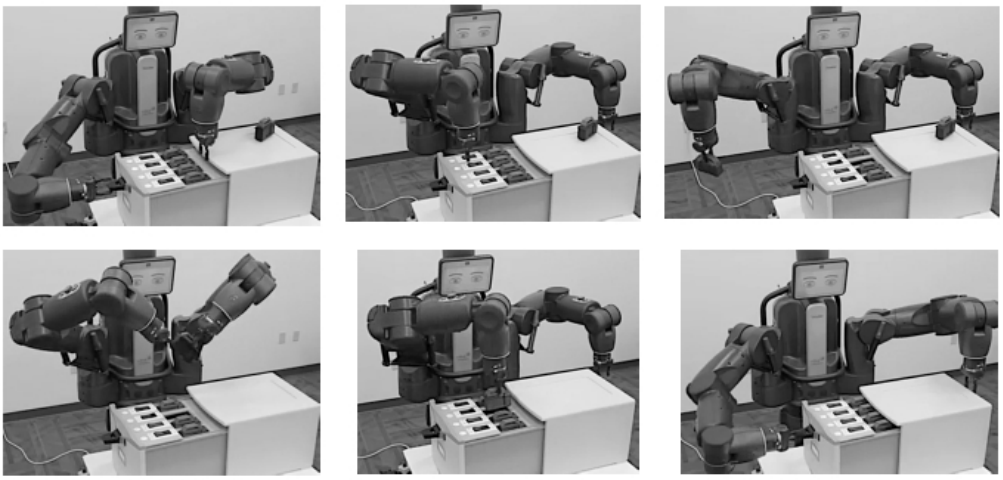

Figure 3.3: Baxter imitates the demonstration pictured in Fig. 3.2. The same intentions are carried out, but with some notable examples of generalization. A different LED is red (top middle), so a different drive is removed (top right). In addition, due to the physical constraints of Baxter's embodiment, the spare drive is picked up by the left gripper but inserted by the right gripper, using a hand-off that was not included in the demonstration (bottom left).

shown in Fig. 3.3. Motion planning used an extension[2] of the bio-inspired DIRECT model for inverse kinematics [47]. Visual sensory processing used home-grown techniques that are not as sophisticated as the state of the art but proved sufficient for the purposes of this work.

[2]Thanks to Gregory Davis for his work implementing this extension.

3.1 A Running Example: Hard Drive Maintenance

To understand how CERIL works in a more detailed and concrete fashion, consider the following learning scenario called the "hard drive docking station" as a running example. A robot must learn procedures for maintaining a docking station for several hard drives that are subject to hardware faults (pictured in Fig. 3.1, bottom left and right, and Figs. 3.2 and 3.3). Each drive slot is linked to an LED fault indicator and a switch that must be toggled before inserting or removing drives. The goal is to replicate a teacher's *intentions*, based on just one demonstration, in new situations that require different motor plans. For example, if the teacher discards a faulty drive and replaces it with a spare, so must the robot, even when a different slot is faulty and the spare is somewhere else. Due to the robot's physical constraints, it may need to use different motor actions than those used by the demonstrator, such as using a different arm for certain steps, handing off objects between grippers, or temporarily putting down one object to perform another manipulation that only one arm can reach. Physical experiments were conducted using a mock-up docking station that was constructed for our lab (Fig. 3.1, lower right, and Fig. 3.3).[3] The dock has faux 3D-printed "hard drives" and an Arduino controller for the LEDs and toggle switches. In our full set of experiments we also worked with two additional scenarios. First, we used a toy block scenario where the teacher stacks blocks in various patterns such as letters, and the robot must replicate those patterns even when extraneous blocks are present and the important blocks are in completely different

[3]Thanks to Ethan Reggia for building the physical docking station.

initial positions. Second, we used a pipe and valve scenario where an LED pressure gauge indicates the status of a pipe and the robot has to manipulate or maintain pressure valves. Applying the CERIL framework in these settings involves the steps described in the following sections. Sections 3.2-3.5 describe the typical operation of CERIL whenever a demonstration is imitated. CERIL also relies on a set of background knowledge that must be provided once up front before any demonstrations can be imitated. Section 3.6 describes the background knowledge required.

3.2 Recording Demonstrations

The first step in the imitation learning process is to record a demonstration. To capture human demonstrations, we use SMILE, a virtual environment shown in Fig. 3.1 (bottom left) and Fig. 3.2. SMILE is a graphical computer program with intuitive GUI controls in which users can manipulate 3D objects located on a tabletop and record their actions. Objects are grasped and moved by clicking and dragging. Radio buttons are used to indicate which hand should perform the manipulation. Push buttons are used to start or stop a recording, and to undo recent actions.

The recording is output in both video format and a machine-readable (text-based) event transcript, describing which objects were grasped, with which hands, and real-time changes in object positions and orientations. SMILE supports several built-in shapes, an XML schema for composing shapes to define more complex objects, and also raw STL files for describing arbitrary face-vertex geometry. Aside

from the existence of a left and right hand, SMILE contains no model of the user's embodiment (e.g., kinematic chains or joint angles), and no indication of the user's overarching intentions that lead them to grasp, move, and release various objects in various positions. Instead, objects are manipulated through mouse clicks, drag-and-drops, and keystrokes. As a result, SMILE bypasses the substantial image processing challenges of human motion capture, as well as the challenges of viewpoint transformation (including orientation, distance, and anthropometry) [92, 93].

SMILE's event record includes entries such as `grasp` and `release`, which might be viewed as primitive from a demonstrator's perspective. The former specifies which object was grasped, and both specify which hand performed the action. However, this level of detail is not primitive from a robotics perspective. In particular, a geometric transformation may be needed to compute the ideal placement of the gripper relative to the object that is being grasped, or relative to the destination where the object is being released. In turn, an inverse kinematics solver may be needed to compute the ideal trajectory of joint angles that results in the correct position for the gripper, while avoiding obstacles in the environment. In general, the lack of embodiment in SMILE means that some events viewed as primitive from SMILE's perspective will most naturally map onto intentions in CERIL's knowledge base that are low but not lowest in the hierarchy, and therefore not actions, strictly speaking. This is indicated in Fig. 3.1 by showing a slightly deeper hierarchy on the top right than on the top left. In other words, some of the bottom-most nodes on the top-left will be low-level intentions recorded in SMILE, not lowest-level robotic actions, but we abuse notation and label them all "Act" in the figure for sake of

simplicity in presentation.

The resulting demonstration is presented to CERIL as a sequence whose entries are either actions or low-level intentions, including labels for what was done (e.g., `grasp`) and also parameters such as which hand was used and which object was manipulated (e.g. `right-gripper`, `drive-1`). We refer to the number of actions/intentions in this sequence as the *length* of the demonstration.

3.3 Learning New Skills by Inferring Intentions

Once CERIL receives a new demonstration from SMILE, it uses abductive inference to infer the high-level intentions of the demonstrator (Fig. 3.1, A). The inference process draws on detailed background knowledge about the various intentions available and their causal relationships (described further in Section 3.6). The result of the inference is one or more *explanations* for the demonstration. Each explanation is a sequence of high-level intentions that could account for the actions that were observed in the demonstration. These explanations serve as CERIL's internal representation of learned behaviors.

For example, one of the simplest skills taught to CERIL was how to discard a drive next to a red LED. The demonstration was the following length-10 sequence of events:

```
move-arm-and-grasp(right-hand, dock-drawer)
move-grasped-object(right-hand, dock-drawer, table)
release(right-hand)
press-dock-switch(left-hand, dock-switch-1, off)
move-arm-and-grasp(left-hand, drive-1)
move-grasped-object(left-hand, drive-1, discard-bin)
```

```
release(left-hand)
move-arm-and-grasp(right-hand, dock-drawer)
move-grasped-object-right-hand, dock-drawer, table)
release(right-hand)
```

Note that although SMILE contains no model of a robot's kinematic chains, it does have radio buttons to indicate which hand is used, from which the parameters above are populated. The low-level intention `move-arm-and-grasp` is primitive from SMILE's perspective but not from the robot's. It is added to SMILE's event record whenever an object is clicked, whereas during robot planning (Section 3.5) it decomposes into lower-level operators with the necessary gripper pose relative to the object and ultimately the necessary joint trajectories. For `move-grasped-object`, the second parameter is the object being moved and the third is a destination to which it is moved. There are also real-valued matrix parameters indicating the relative geometric transformations between target object positions and the destinations where they are placed, omitted above for simplicity.

Based on this demonstration, CERIL inferred two possible top-level intention sequences for the skill. First:

```
open(dock-drawer)
set-dock-switch(dock-switch-1, off)
discard-object(drive-1)
close(dock-drawer)
```

and second:

```
open(dock-drawer)
set-dock-switch(dock-switch-1, off)
```

```
move-to-free-spot(drive-1, discard-bin)
close(dock-drawer)
```

In this case, both explanations amount to the same behavior, but are technically distinct since `discard-object` and `move-to-free-spot` were encoded as distinct top-level intentions in CERIL's knowledge base. The `discard-object` intention always targets the discard bin, whereas `move-to-free-spot` can also target other surfaces such as the top of the dock case or the tabletop. Each top-level intention in each sequence can explain a sub-sequence of the demonstration, through a chain of causal relationships present in the knowledge base.

The abductive inference mechanism used here is this dissertation's novel extension of PCT, detailed in Chapter 4. The main entrypoint to this mechanism is an algorithm called EXPLAIN, which takes a demonstration sequence as input and returns zero or more high-level intention sequences as output. Each sequence in the output represents another viable explanation for the input. An important advantage of using PCT is that various parsimony criteria can be used to filter the full set of explanations down to a plausible subset. As detailed in Chapter 5, there were many cases where the full set of valid explanations was combinatorially large, with cardinalities in the thousands or more. Filtering by the right parsimony criterion resulted in cardinalities on the order of 1 or 2, as in the example above.

If the user provides a label for a skill that they demonstrate (e.g., `discard-bad-drive`), then the skill can be added to the knowledge base as a new intention that was not initially provided by the domain author. This is relatively

straightforward as long as one settles for a descriptive rather than operational representation of the new intention (the previously existing knowledge can still be operational). Specifically, the new label serves as the name of a new HTN task, and each inferred intention sequence serves as another HTN method for the new task. The parameters for the new task can be taken as the concatenation of parameters from each intention in the inferred sequence, and the values for those parameters can be propagated during planning or abduction with simple copies. Only one demonstration is needed to learn the new task, but if subsequent demonstrations for the same task label are recorded later, the new sequences inferred by EXPLAIN can be added as new HTN methods for the existing task.

Consequently, every time a new demonstration is interpreted, the inferred intention sequences are added to a growing library of learned skills, and the robot can be asked to imitate any of them at any time. So the system can be (re)programmed to perform different imitation tasks after others have already been learned, without losing any learned knowledge that has been acquired previously. These learned skills also become available as new intentions that were not originally codified by the domain author. This means that in future demonstrations, the new intentions can be used as components of an explanation. In other words, intentions that were originally top-level can become mid-level as even higher-level intentions are learned from demonstration and the causal knowledge grows over time.

As a concrete example and proof of concept for this process, CERIL was first presented the following demonstration (states and some parameters omitted for simplicity):

```
press-dock-switch(right, dock-switch-3, off)
move-arm-and-grasp(right, drive-3)
move-grasped-object(right, discard-bin)
release(right)
```

and inferred the following top-level intention sequence:

```
set-dock-switch(dock-switch-3, off)
discard-object(drive-3)
```

This "skill" was labeled `discard-bad-drive` and added to the knowledge base as a new task, with the intention sequence above as an associated task method. Then, CERIL was presented with the following new demonstration:

```
move-arm-and-grasp(right, dock-drawer)
move-grasped-object(right, table)
release(right)
press-dock-switch(right, dock-switch-2, off)
move-arm-and-grasp(right, drive-2)
move-grasped-object(right, discard-bin)
release(right)
press-dock-switch(right, dock-switch-1, off)
move-arm-and-grasp(right, drive-1)
move-grasped-object(right, discard-bin)
release(right)
move-arm-and-grasp(right, dock-drawer)
move-grasped-object(right, dock-case)
release(right)
```

and inferred the following intention sequence:

```
open(dock-drawer)
discard-bad-drive(drive-2)
discard-bad-drive(drive-1)
close(dock-drawer)
```

As seen in this example, the newly added task was recruited (twice) as part of CERIL's interpretation of the second demonstration. Again, some parameters are

omitted for simplicity: in particular, the full parameter list for `discard-bad-drive` was the concatenation of all parameters from CERIL's interpretation of the first demonstration.

It is important to note that each high-level intention originally provided by the domain author, such as `set-dock-switch` or `discard-object`, is rather general, but not general enough that a single high-level intention can explain an entire demonstration. The typical demonstration can only be explained by a novel *sequence* of high-level intentions, which is not pre-defined in the knowledge base, and must be constructed through causal reasoning. As such, despite the wealth of background knowledge available to the system, there is still non-trivial cause-effect reasoning that must take place in the autonomous system which was not already hand-coded by the domain author. This non-trivial reasoning problem can be also be understood with an analogy to parsing ambiguous grammars. The domain knowledge is analogous to an ambiguous grammar and vocabulary, demonstrations are analogous to previously unseen sentences, and EXPLAIN is analogous to a parser, which still must perform the non-trivial work of parsing a new ambiguous sentence that was not built in to the grammar. Certainly, one could author a domain in which single pre-defined root intentions explain entire demonstrations. We elected not to do so in this work, in order to show that learning and generalization can be achieved by *constructing* new explanations through automated causal inference rather than *recognizing* existing explanations that were hand-coded in an exhaustive knowledge base.

3.4 Transferring Learned Skills to New Situations

After CERIL has learned a new skill from a demonstration, in the form of one or more high-level intention sequences, it is ready to imitate in a new situation (Fig. 3.1, B). However, there is often a non-trivial correspondence problem to be solved between the state that was observed in the demonstration, and the state that is observed at imitation time. For example, suppose that there is a single drive present in the demonstration called `drive-1`. At imitation time, CERIL observes a scene in the physical world with two unlabeled drives. Which one should play the role of `drive-1`, and which is extraneous? We could depend on the end user to annotate objects in every new scene, but that would be tedious and error-prone, reducing the human-friendliness of the system. Instead, it would be preferable for CERIL to automatically find correspondences between the objects it observes at imitation time and demonstration time. For example, if `drive-1` was next to a red LED in the demonstration, and a new drive (let us call it `drive-A`) is next to a red LED at imitation time, perhaps `drive-A` should correspond to `drive-1`. Once this correspondence is found, it can be put into effect by substituting every occurrence of `drive-1` with `drive-A` in any given high-level intention sequence for the learned skill. Then the right intentions will be applied to the right objects in the new situation.

In sum, CERIL needs to solve a correspondence problem and then use the correspondences to perform variable binding and ground the intention parameters with objects in the new situation. We can allow that different numbers of objects

may be present, as long as each object manipulated in the demonstration has a counterpart in the new situation. For example, the robot should not be expected to imitate a demonstration of replacing a faulty drive with a spare drive in a situation where no spare drives are present.

The correspondence matching procedure devised in this work is based on the premise that object colors, shapes, and in/on/part-whole relationships are more important than spatial positions (this could, of course, be changed). The idea is that the robot should be able to generalize to new situations where the initial object positions are completely different than those observed in the demonstration. For example, the fact that `drive-1` is in a closer or farther slot is probably less important than the fact that `drive-1` is in `slot-2`, `slot-2` and `led-2` are both part of `dock-module-2`, and `led-2` is red. These colors, shapes, and in/on/part-whole relationships are available from the state representation that was designed for the dock domain. In particular, the in/on/part-whole relationships are encoded in an "assembly tree" data structure that captures these relationships,[4] pictured in Fig. 3.4. For example, `drive-1` would be a child of the `slot-2` node while inserted there, and `slot-2`, `led-2`, and `dock-switch-2` would be the children of the `dock-module-2` node in the assembly tree.

In order to process this assembly tree data structure, the matching procedure is designed recursively. Each level of recursion compares a demonstration sub-tree with a real-world sub-tree. At the deepest level, the leaves are compared based on

[4]This assembly tree is unrelated to the intention hierarchy.

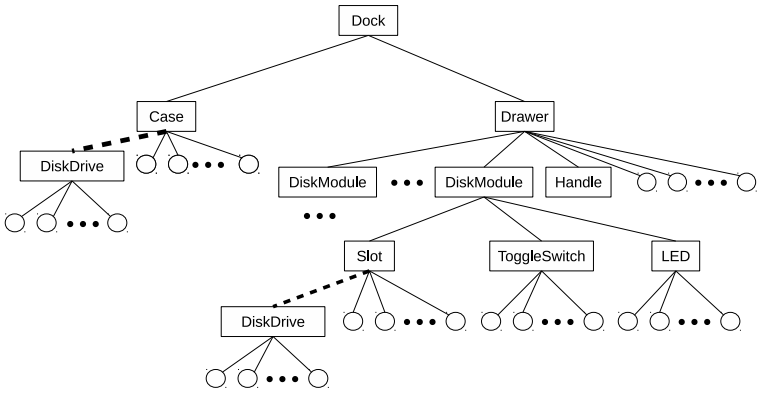

Figure 3.4: A sample tree structure for the dock assembly. Circles indicate the leaves of the tree, which are the underlying shapes (cylinders, rectangular prisms, etc.) combined to form the assembly. Dashed lines indicate sub-assemblies which can be removed - for example, a drive in a slot (lower dashed line), or a drive resting on top of the dock case (dashed line on the left).

shape and color and a quantitative similarity measure is passed up the recursion. At the shallower levels, each child of the current demonstration tree node is recursively compared with each child of the current real-world tree node. The similarity measures recursively computed for each child pairing are used as weights for a weighted bipartite matching, which produces the optimal pairing of demonstration children with real-world children. The summed similarities across this optimal pairing are then passed up the recursion.

This strategy is illustrated in Fig. 3.5. By design, the results of this procedure are such that in the dock example they will match dock modules that are similar with respect to LED colors and slot occupancies, but potentially dissimilar with respect to their spatial position on the dock drawer. For example, suppose that in the demonstration, $\texttt{slot-1}^{(demo)}$ is occupied and $\texttt{led-1}^{(demo)}$ is red, and in the new scene, slots $2^{(new)}$ & $3^{(new)}$ are occupied but only $\texttt{led-3}^{(new)}$ is red. Slots and LEDs $1^{(demo)}$ & $3^{(new)}$ will be matched, rather than $1^{(demo)}$ & $2^{(new)}$, since the configuration of colors, shapes, and part-whole relationships is better preserved. The matching only compares the initial state in the demonstration with the initial state during imitation, but in future work more sophisticated matching that accounts for the entire demonstration may be possible.

3.5 Post-Learning Imitation and Generalization

Once matching is complete, variable substitution is used to update parameters in the high-level intention sequence so that object identifiers from the demonstration

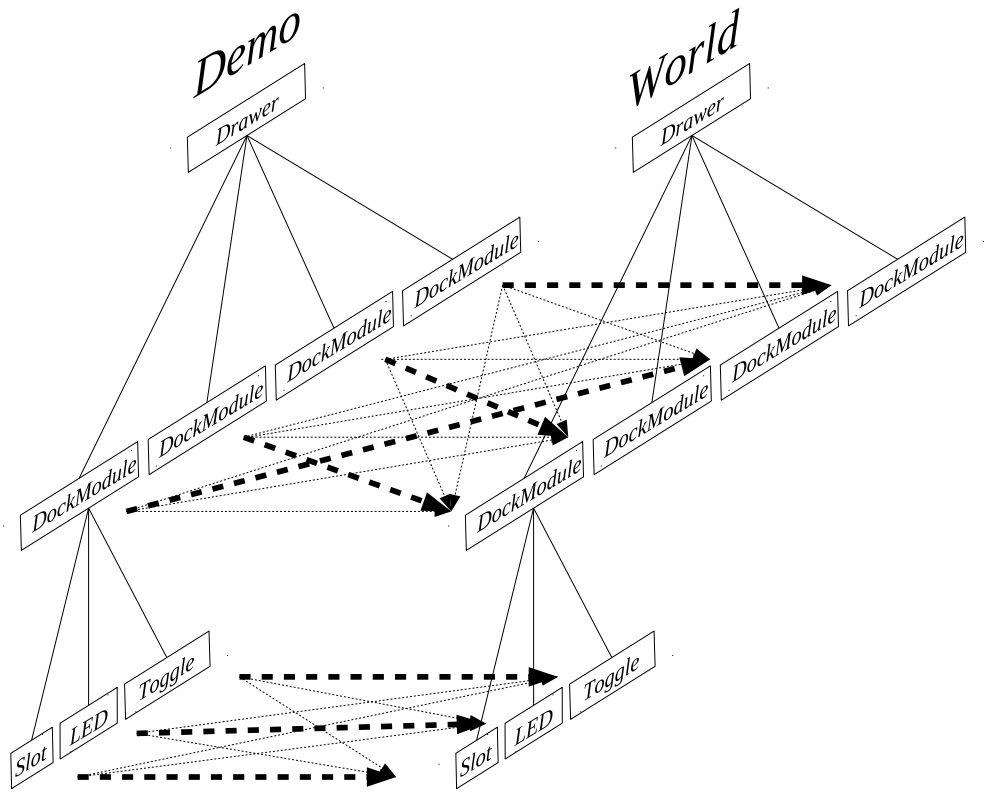

Figure 3.5: An example match produced by the assembly matching algorithm. Solid lines indicate the tree structures of each assembly. Dashed lines indicate possible pairings that preserve tree structure. Bold dashed lines provide an example of bipartite matches found at each level. At the bottom layer, only children with the same assembly type can be paired (e.g., an LED cannot be matched with a toggle). But in the middle, where all assembly types are DockModules, many pairings are possible. The pairing which results in highest color and shape similarity in the leaves is chosen, even if it permutes the order of the children. This may happen if, for example, one LED is red in the demo but a different LED is red in the new initial state.

are replaced with the matched identifiers in the real world. HTN planning is then used in a top-down manner to plan a sequence of low-level motor commands that carry out these learned intentions (Fig. 3.1, C). If more than one intention sequence was inferred by EXPLAIN as a valid explanation for the demonstration, the sequences can be tried in an arbitrary order until planning succeeds, much like the backtracking that is already built in to the HTN planning algorithm.

As elaborated next in Section 3.6, an intention might cause different sub-intention sequences depending on the parameters and current state of the environment - for example, the intention to toggle a dock switch may cause the sub-intention to put down a grasped object, if in the current state the implicated gripper is not already free to press the switch. These different causal branches represent alternate strategies for carrying out the parent intention, some of which may be more or less appropriate depending on the current state of the environment, and some of which may be quite different from the strategy that was actually used by the human demonstrator. The HTN planner can search each branch, simulating the effects of that branch on the environment, and backtrack when necessary to avoid branches that fail. Consequently, the resulting actions planned for the new situation may differ significantly from the observed actions in the demonstration, as indicated by the distinct intention trees on the left and right of Fig. 3.1. Moreover, as described earlier, the lowest-level HTN operators can invoke motion planning routines, which convert target gripper positions into joint angles that avoid obstacles and respect the physical constraints of the robot. As a result, the causal hierarchy can extend deeper than the actions recorded in SMILE, producing concrete motor plans suitable

for physical robot execution (illustrated schematically in Fig. 3.1 by the deeper tree on the right-hand side).

As a detailed example, the most complicated skill that CERIL learned was to swap the positions of two drives, one next to a red LED and the other next to a green LED. In this example, the demonstrator swapped two hard drives by removing one with the left hand radio button selected, the other with the right hand radio button selected, and then inserting them back in opposite slots, one with each hand. However, the slots are difficult for the robot to reach with its left hand due to physical embodiment constraints. When the HTN planner searches a branch that uses the left arm, motion planning fails and the search must backtrack. An alternate search branch succeeds where the right gripper first removes one drive, then hands it to the left which stages it on top of the dock, then moves the second drive to the slot that initially contained the first drive, and finally receives the first drive back from the left and inserts it in the slot that initially contained the second drive. The resulting plan of robotic actions is significantly different from what was demonstrated. In other words, the system successfully generalizes a learned skill on the basis of a single demonstration.

CERIL's capacity for generalization boils down to the synergistic combination of abductive inference, object matching, and planning (arrows A, B, and C in Fig. 3.1). Object matching ensures that intentions are applied to the correct objects. Abductive inference ensures that the most general and plausible intentions are applied, and planning ensures that they are applied in a way that respects the embodiment of the robot, rather than the human demonstration. Causal intention inference is a

crucial component because it identifies intentions that are as high-level as possible given the pre-defined background knowledge. Higher-level intentions are better for generalization because they expose more branches for the HTN planner, resulting in more opportunities for success. In sum, cause-effect inference of parsimonious explanations is central to generalization after learning from a single demonstration.

3.6 Encoding Background Knowledge

The imitation learning pipeline described in Sections 3.2-3.5 relies heavily on a compendium of background knowledge. Before CERIL can engage in imitation learning, this incurs a one-time up-front cost: A domain author must first encode their background knowledge in a machine representation that CERIL understands. There are three important categories of background knowledge, as follows.

First is a detailed model of the environment, including possible objects and their relationships. For example, the author will define the relevant geometry (e.g., vertices, edges, face normals) of a hard drive, as well as the available grasp poses (e.g., the spatial transformations from the hard drive coordinate frame to a coordinate frame 3 inches above where the gripper should be moved before grasping), and then similarly for toggle switches, dock drawers, and so on. Then the author will define various object relationships that are possible, such as hard drives being in slots, on tables, in discard bins, or on top of dock cabinets. Finally the author will define a data structure representing the full *state* of the environment at any given time, including all objects present and which relationships obtain. As described

earlier in Section 3.4, the state representation devised in this work was based on "assembly trees" that capture in/on/part-whole relationships in assemblies of objects,[5] as pictured in Fig. 3.4. The dock is a root of an assembly, the cabinet and drawer are children of the dock, toggle switches and slots are children of the cabinet, drives are children of the slots while they are inserted, and so on. Each parent-child relationship is annotated with the precise spatial transformation that determines the relative positions of parent and child. By composing relative transformations, absolute positions can be retrieved when the robot needs to reach a part of an assembly (e.g., a drive in a slot), and child positions can be updated recursively when the robot manipulates a part (e.g., if a drive is inserted and the robot opens the dock drawer, the drive position must be automatically updated in CERIL's state representation so that its expected position is carried along with the dock drawer). States are depicted in Fig. 3.1 as small 3D block graphics at the bottom of the intention hierarchies. Since SMILE supports arbitrary STL face-vertex data and customizable assemblies of primitive shapes, the object designs provided to CERIL by the domain author can also be imported into SMILE with relatively little additional effort. The state representation also included some more traditional descriptive encodings where convenient (e.g., logical predicates asserting what, if anything, is presently gripped by each gripper; whether each slot is occupied; whether each LED is red, green, or off, etc.).

Next, the domain author defines the available low-level *actions* that CERIL

[5]Assembly trees are unrelated to the intention trees pictured in Fig. 3.1.

can perform to change the state of the environment. Actions are indicated in Fig. 3.1 by boxes labeled "Act". The current version of CERIL uses an *operational* encoding of actions, wherein each action is implemented as a generic computer program and need not conform to a predefined or constrained representation (such as lists of logical predicates that become false or true). This operational encoding was authored using a combination of Python and MATLAB™. The inputs to each action are the current state followed by zero or more additional parameters. The output is the new state expected after the action is performed, or an indication of failure if the action is not a valid option in the current state. For example, one of the actions in the dock domain is `grasp`. The inputs are the current state, an identifier for which arm is doing the gripping (e.g., `left-gripper`), and an identifier for the object being gripped (e.g., `drive-1`). If the specified arm is already gripping something else, the output is `Failure`. Otherwise, the output is a new state data structure reflecting the change (e.g., where the relationship `(gripping, left, nothing)` becomes false and the relationship `(gripping, left, drive-1)` becomes true). Another action is `change-joints`, one of whose inputs are the new joint angles for an arm. The output of this action is a new state in which the joint angles are updated, and any object marked as gripped also has its position updated accordingly. The need to update gripped object positions with geometric computations highlights the advantages of using an operational, rather than descriptive, action representation.

Finally, the author defines the available higher-level *intentions* that CERIL can carry out. CERIL carries out an intention by queuing a sequence of sub-intentions, some of which may be low-level actions. Intentions are depicted in Fig. 3.1 with

boxes labeled "Intent". From the point of view of intention inference, we can think of a parent intention as *causing* its child intentions. These causal relationships are indicated in Fig. 3.1 as solid black arrows from causes to their effects. Parent-child intention relationships are also encoded operationally. The causal knowledge is encoded twice, once for each causal direction:

- *Causes to effects*: Each intention is represented as a generic MATLAB function. As with actions, the input is the current state and zero or more additional parameters, and the output is `Failure` when the intention is not a valid option. However, unlike actions, a successful output is not a new state but instead a sequence of sub-intentions to be queued. The contents of this sequence can be dependent on the function inputs, so that the same parent intention may decompose into different sub-intentions depending on the situation. This representation is equivalent to a "task" in the context of HTN planning, with each decomposition branch corresponding to a "task method," and is used during CERIL's planning process.

  As a concrete example, consider the `press-dock-switch` intention, whose inputs are the current state, an arm to use (e.g., `left-gripper`), an identifier for the toggle to be pressed (e.g., `dock-switch-1`), and the desired setting (e.g., `off`). It returns `Failure` if the desired arm is already gripping an object. Otherwise it returns the sequence of sub-intentions `move-arms`,`close-gripper`, `move-arms`. The first sub-intention positions the gripper above the toggle, the second closes the two gripper fingers so that it is easier to physically push

the toggle switch, and the third lowers the grippers to perform the actual toggle press. The sub-intention sequence also includes the appropriate parameter values for each sub-intention, omitted above for simplicity. For example, the parameters for `move-arms` include spatial transformation matrices describing the target gripper positions. This spatial information must be computed based on the position of the toggle switch in the current state. The need for these computations highlights the utility of using an operational representation, since they are most naturally expressed in computer code. Other technical details are also omitted, such as additional `move-arms` waypoints to reduce the chance of obstacle collisions.

A higher level intention in the dock domain is `set-dock-switch`, which does not specify a gripper, but internally selects the gripper that can more easily reach the toggle switch. If the selected gripper is not currently grasping anything, this intention causes a sub-intention sequence with a single element: `press-dock-switch`, with parameter values propagated accordingly. However, if the gripper is not empty, it causes the sub-intention sequence

```
free-gripper,press-dock-switch,restore-gripper
```

in which the first intention will temporarily put down the gripped object so that the gripper is available to press the toggle, and the third intention will restore the previously gripped object to the gripper. This is an example of how higher level intentions can be composed of lower-level intentions. The sub-intention `free-gripper` is also another example of the benefits of operational

representation: In order to physically put down an object in the real world, a free location needs to be computed in which the object will not collide with any other objects in the environment. This is a complicated computational geometry problem that would be very difficult to express without a generic programming language such as Python or Matlab. In the dock domain this problem was solved with a randomized search that checks many candidate positions until it finds one where no collision is detected.

The highest-level intentions pre-defined in the dock domain knowledge base dock manipulations such as `open-dock` and `set-dock-switch`, as well as a generic `get-object-A-to-object-B` intention, which may cause various sub-intentions such as temporarily emptying grippers and handing off objects between grippers, all depending on the intention parameters and the current state of the environment.

- *Effects to Causes*: The same set of knowledge described above also needs to be encoded in a form suitable for intention inference. In this case, reasoning proceeds in the opposite direction: given a set of observed effects, CERIL must hypothesize potential causes. In other words, given any sequence of sub-intentions, CERIL must be able to query its background knowledge to identify all of the possible parent intentions that could have caused that sequence (if any). This knowledge is also encoded operationally, as a generic function in the Python language. The inputs to the function are a sequence of sub-intentions (including parameter values and intermediate states), and the output is a set

of possible parent intentions, any one of which could have caused the provided input. In the context of PCT, this plays the role of the CAUSES function.

For example, if CAUSES is provided the input sequence

`free-gripper,press-dock-switch,restore-gripper`

then the parent intention `set-dock-switch` (with parameter values bound appropriately) will be a member of the output set, since it is one of the intentions that could have caused the input. In general, the output set may have cardinality zero, if there is no such parent intention, or it may have cardinality greater than 1, if there is more than one such parent intention. For example, suppose the sequence

`move-arms`, `release`

moves the left arm to the dock cabinet and releases `drive-1` there (with parameters and intermediate states omitted for simplicity). This could have been caused by an intention `move-to`(`drive-1`,`dock-cabinet`), if getting `drive-1` to `dock-cabinet` is an important part of the final state. Or, it could have also been caused by the intention `free-gripper`, if putting down `drive-1` was only the means to an end of pressing a toggle switch.

This example also illustrates why it may be important to include intermediate states in the input to CAUSES. The implementation may need to query the state to see what is gripped (e.g., `drive-1`) as it may not be explicit in the parameter values of the sub-intentions (e.g., `release`(`left-gripper`)). It also may need to inspect the spatial positions of objects in order to compute spatial

position parameters in the parent intention (e.g., the position of `drive-1` after the release determines a target spatial position that is supplied as a parameter to the parent `move-to` intention).

The need to encode the same causal knowledge twice is an example of the *dis*advantage of operational encoding. Automatically inverting a computer function in a generic programming language is a complex problem in program analysis that will generally be infeasible. Therefore CERIL must rely on the domain author's expertise and have them implement the inverse themselves. If the knowledge were encoded descriptively, this inversion might be more tractable (e.g., using automated theorem proving techniques). Nevertheless, as explained above, there are good reasons for allowing operational representations and incurring this cost. Codifying the background knowledge is a significant undertaking, but it need only be done once. After it is finished, CERIL is ready to imitate any number of demonstrations. The full set of causal knowledge that was authored for CERIL in this work is detailed in Appendix A.1.

Chapter 4: Causal Reasoning Algorithms for Inferring Intent

Chapter 3 presented the CERIL architecture with an example of the full imitation learning workflow at a conceptual level. An important step in this process is abductive inference of the demonstrator's high-level intentions, as described in Sect. 3.3. This step uses a novel extension of PCT to support intention inference. In particular, to my knowledge, this is the first extension of PCT that simultaneously supports ordered effects (A causes B, then C, then D) and causal chaining (A causes B, B causes C). In addition, it is distinguished from past work by its operational representation of causal knowledge. All of these extensions to PCT were necessary to properly represent and reason about intentions in the context of robotic imitation learning. A core contribution of this dissertation is the development of algorithms to support these extensions, as well as the theoretical analysis that verifies their soundness, completeness, and computational complexity characteristics [69]. These formal verifications make CERIL well understood at a technical level, thereby promoting transparency and trustworthiness. This chapter presents those algorithms and theoretical results in detail.

4.1 Formalizing Causal Knowledge

The formal knowledge representation, algorithms, and proofs rely heavily on a notation for ordered sequences. An ordered sequence of N elements from a set V is denoted $\langle v_i \rangle_{i=1}^{N} = \langle v_1, v_2, ..., v_N \rangle$, where each $v_i \in V$. We denote the set of all such sequences using the Kleene star: V^*. For brevity, arbitrary sequences may be written with subscripts and superscripts omitted, as in $\langle v \rangle$, as long as we have not already used the token v in the current context to refer to an individual element of V.[1] When subscripts and superscripts are omitted, the length of an arbitrary sequence $\langle v \rangle$ is denoted $|\langle v \rangle|$. To denote a *sequence of sequences*, each member sequence is written with a parenthesized superscript: e.g., $\langle v \rangle^{(1)}, \langle v \rangle^{(2)}, ...$

Henceforth let V be any (potentially infinite) set whose elements represent anything that can be a cause or effect. In our context, elements of V correspond to actions and intentions, such as opening the dock or grasping a hard drive. Formally, in CERIL each element $v \in V$ is a tuple of the form $(t, s, \langle x \rangle)$, where t is a label for the intention or action (e.g., `grasp`), s is the current state of the environment immediately before t is carried out, and $\langle x \rangle$ is a sequence of parameter values with which t is carried out (e.g., $\langle$`drive-1`,`left-gripper`$\rangle$). However, all of the algorithms in this chapter are agnostic to the internal structure of any particular v and could potentially be used in other application areas.

A *causal relation* over V is a set $C \subseteq V \times V^*$. An element $(u, \langle v \rangle) \in C$ sig-

[1] E.g., the statement "Given $v \in V$, consider $\langle v \rangle$ ..." refers to a length 1 sequence consisting of v, not an arbitrary sequence.

nifies that u can *cause* the sequence $\langle v \rangle$. This means that u *may* cause $\langle v \rangle$, not that it *must*. However when u actually does cause $\langle v \rangle$ it must cause the full sequence in order. In CERIL, elements of C represent a parent intention causing a sequence of sub-intentions (downward arrows in Fig. 3.1). For example, u might represent getting `drive-1` to `right-gripper`, with two associated causal relationships in C: $(u, \langle v \rangle^{(1)})$ and $(u, \langle v \rangle^{(2)})$. The first effect $\langle v \rangle^{(1)}$ might be a singleton sequence with one sub-intention such as picking up `drive-1` with `right-gripper`. Meanwhile $\langle v \rangle^{(2)}$ might be a sequence with two intentions: first picking up `drive-1` with `left-gripper`, and then handing off from `left-gripper` to `right-gripper`. The u parent intention *may* cause $\langle v \rangle^{(1)}$, or it *may* cause $\langle v \rangle^{(2)}$. But when it causes $\langle v \rangle^{(2)}$, both sub-intentions (the pick-up and the hand-off) must be caused, in order.

In simple examples C can be depicted graphically by drawing elements of V as circles, with vertical arcs connecting causes with effects, and horizontal arrows through vertical arcs to indicate ordering constraints. Several examples shown in Fig. 4.1 are referenced throughout the following. Note that C may be many-to-many: the same u might cause any of several different $\langle v \rangle$'s and vice versa. For example, in Fig. 4.1(a), $\langle v_1, v_2 \rangle$ can be caused by either u_1 or u_2. And u_2 can cause either $\langle v_1, v_2 \rangle$ or $\langle v_3, v_4 \rangle$.

Sequence membership is written with $\in$, as in $v_2 \in \langle v_1, v_2, v_3 \rangle$. Given any sequence

$$\langle (u_1, \langle v \rangle^{(1)}), (u_2, \langle v \rangle^{(2)}), ..., (u_\ell, \langle v \rangle^{(\ell)}) \rangle \in C^*,$$

if $u_{l+1} \in \langle v \rangle^{(l)}$ for all l from 1 to $\ell - 1$, then we refer to the sequence as a *causal*

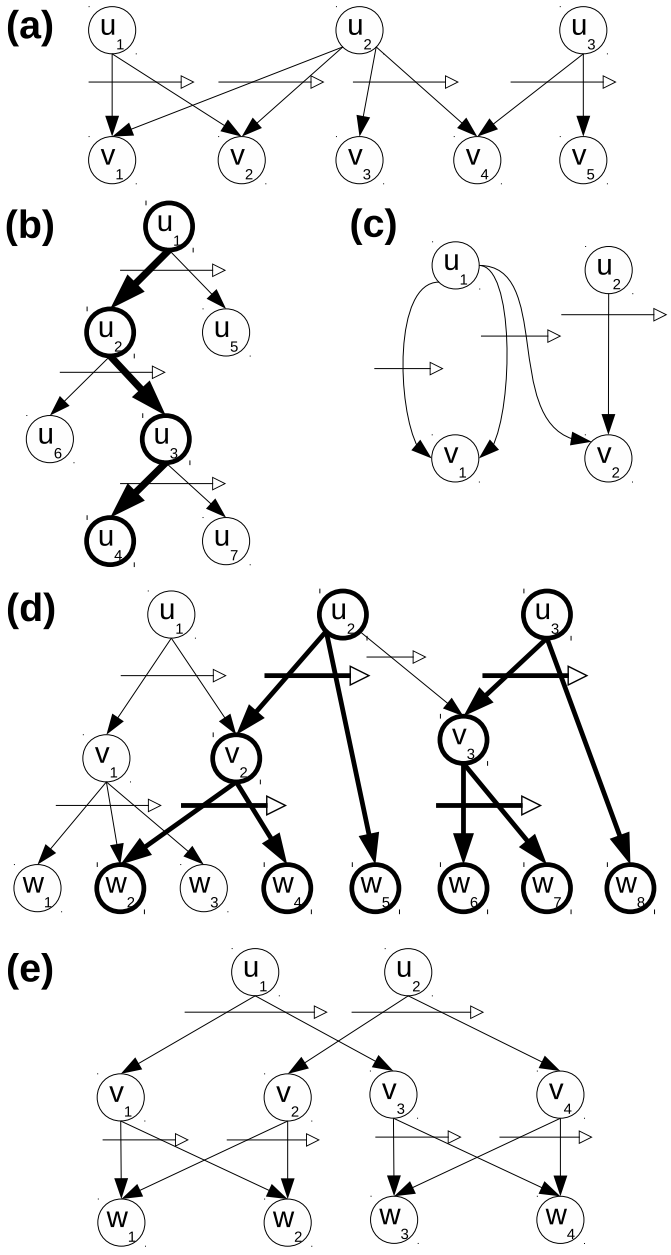

Figure 4.1: Examples of causal relations. Nodes represent elements of V, vertical arrows represent causal relationships, and horizontal arrows represent ordering constraints. See text for further details.

chain with depth ℓ. We restrict our attention to causal relations with no "loops": for any causal chain with depth ℓ, we assume $u_1 \notin \langle v \rangle^{(\ell)}$. An example of a causal chain is shown in Fig. 4.1(b) with the chain in bold: $\langle (u_1, \langle u_2, u_5 \rangle), (u_2, \langle u_6, u_3 \rangle), (u_3, \langle u_4, u_7 \rangle) \rangle$. Concretely, getting `drive-1` to `right-gripper` might cause a pick-up (among other things), which in turn might cause a gripper to close (among other things). This is a causal chain with depth 2 (because there are 2 causal links).

A *covering tree* is an ordered tree in which every ordered parent-child relationship, $(u, \langle v \rangle)$, is a member of C. In Fig. 3.1, each top-level intention is the root of the covering tree below it. The root of any covering tree is called a *singleton cover* of the ordered leaves in the tree. In Fig. 4.1(d), u_2 is a singleton cover of $\langle w_2, w_4, w_5 \rangle$, and u_3 is a singleton cover of $\langle w_6, w_7, w_8 \rangle$. The respective covering trees are shown in bold-face. The root is thought of as "covering" its descendants because it can account for them in an explanation.

Consider I covering trees: u_1 is a singleton cover of $\langle w \rangle^{(1)}$, ... u_i is a singleton cover of $\langle w \rangle^{(i)}$, ... u_I is a singleton cover of $\langle w \rangle^{(I)}$. We call the sequence $\langle u_i \rangle_{i=1}^{I}$ a *cover* of $\langle w \rangle = \langle w \rangle^{(1)} \oplus ... \oplus \langle w \rangle^{(I)}$, where $\oplus$ denotes sequence concatenation. In other words, a cover is formed by (the roots of) an ordered forest of cover trees. Each u_i is referred to as a *singleton sub-cover* of $\langle w \rangle^{(i)}$. In Fig. 4.1(d), $\langle u_2, u_3 \rangle$ is a cover of $\langle w_2, w_4, w_5, w_6, w_7, w_8 \rangle$, and u_2 is the singleton sub-cover of $\langle w_2, w_4, w_5 \rangle$. Note that a singleton cover is simply a cover with a single element.

Let $\langle u \rangle$ be a cover of $\langle w \rangle$. If an associated covering forest has depth at most ℓ, then $\langle u \rangle$ is also called an *ℓ-cover* of $\langle w \rangle$. Any $\langle w \rangle$ is considered a 0-cover, or *trivial cover*, of itself.

The contiguous subsequence relation is written $\sqsubseteq$, as in $\langle v_2, v_3\rangle \sqsubseteq \langle v_1, v_2, v_3, v_4\rangle$. Given any $\langle w\rangle \in V^*$, suppose there is a non-empty subsequence $\langle v\rangle \sqsubseteq \langle w\rangle$ and a $u \in V$ such that $(u, \langle v\rangle) \in C$. In other words, suppose part of $\langle w\rangle$ can be caused by some u. Then $\langle w\rangle$ is referred to as *mid-level*. Otherwise, $\langle w\rangle$ is referred to as *top-level*. In Fig. 4.1(e), $\langle v_1, v_3\rangle$ is a mid-level cover for $\langle w_1, w_2, w_3, w_4\rangle$ because the (full) subsequence $\langle v_1, v_3\rangle$ can be caused by u_1. In contrast, there are four distinct top-level covers of $\langle w_1, w_2, w_3, w_4\rangle$: $\langle u_1\rangle, \langle u_2\rangle, \langle v_1, v_4\rangle$, and $\langle v_2, v_3\rangle$. Fig. 4.1(e) also shows that the roots of a top-level cover are not necessarily top-level nodes: while v_1 is *part of* a sequence caused by u_1, and v_4 is *part of* a sequence caused by u_2, there is no node that can cause the singleton sequence $\langle v_1\rangle$, the singleton sequence $\langle v_4\rangle$, or the sequence $\langle v_1, v_4\rangle$. So $\langle v_1, v_4\rangle$ is a top-level cover.

4.2 Formalizing Parsimonious Explanation

Any cover is considered to be a valid explanation, but it may not be a *good* explanation. In PCT, "good" explanations are identified by choosing a suitable parsimony criterion and discarding all valid explanations that do not satisfy that criterion. Two formal parsimony criteria previously used in PCT are *minimum cardinality* and *irredundancy*, although these do not involve temporal ordering in the original formulation [98]. In our context, $\langle u\rangle$ is a *minimum cardinality* cover of $\langle w\rangle$ if there is no other $\langle v\rangle$ with fewer elements that also covers $\langle w\rangle$. $\langle u\rangle$ is an *irredundant* cover of $\langle w\rangle$ if there is no proper subsequence $\langle v\rangle$ of $\langle u\rangle$ (contiguous or not) that also covers $\langle w\rangle$. Either criteria can be imposed depending on the problem

domain, and only parsimonious covers are considered to be acceptable explanations. However, if minimum cardinality is imposed, irredundancy is automatically imposed as well, since a redundant cover has higher cardinality than the same cover with the redundant elements removed.

From the standpoint of imitation learning, the chief concern is identifying intentions that are as general as possible, so CERIL always imposes another parsimony criterion: $\langle u \rangle$ is considered a *parsimonious* cover of $\langle w \rangle$ only if it is a *top-level* cover of $\langle w \rangle$. We can impose additional criteria such as minimum cardinality (and hence also irredundancy) in addition to top-level-ness if necessary to further reduce the set of valid explanations. Even if minimum cardinality is not imposed, top-level covers will generally satisfy the irredundancy criterion, since removing some roots from a covering forest will result in uncovered leaves.[2] Top-level-ness and irredundancy are also similar in spirit, since both emphasize covers that have been maximally simplified by local modifications: redundant covers are made irredundant by removing a subset; mid-level covers are made top-level by replacing a subsequence with its cause.

A *causal problem domain* (or simply "domain") is a pair $D = (V, C)$ where C is a causal relation over the set V. A *causal inference problem* is a pair $(D, \langle v \rangle)$, where D is a domain and $\langle v \rangle \in V^*$ is an observed sequence to be explained ($\langle v \rangle$ corresponds to an observed demonstration in imitation learning). The problem's

[2]There are contrived exceptions: In Fig. 4.1(c), $\langle u_1, u_2 \rangle$ covers $\langle v_1, v_2 \rangle$, but is redundant, since $\langle u_1 \rangle$ also covers $\langle v_1, v_2 \rangle$. Causal relations like this do not occur in our imitation learning domain and are rare in the Monroe Plan Corpus (Chapter 5), so we consider them pathological in practice.

solution is the set of all parsimonious covers of $\langle v \rangle$. For the purposes of this work, top-level-ness is always imposed; additional criteria can be imposed on the solution as a post-processing step.

Several domain-specific constants are useful for quantifying the size of a domain:[3]

- M, the length of the longest sequence that can be caused by any $u \in V$ (i.e., $M = \sup_{(u,\langle v \rangle) \in C} |\langle v \rangle|$). $M = 3$ in Fig. 4.1(d), and $M = 2$ in Fig. 4.1(a),(b),(c), and (e). This refers to the length of an *ordered effect sequence* (i.e., "horizontally"), not a *causal chain* (i.e., "vertically").

- U, the largest possible number of distinct singleton covers of the same $\langle v \rangle$, taken over all $\langle v \rangle \in V^*$. That is,

$$U = \sup_{\langle v \rangle \in V^*} |\{u \in V \mid \langle u \rangle \text{ covers } \langle v \rangle\}|.$$

  In Fig. 4.1(d), $\langle w_6, w_7 \rangle$ has two distinct singleton covers: u_2 and v_3. All other possible node sequences have 2, 1, or 0 distinct singleton covers, so $U = 2$.

- L, the depth of the deepest causal chain (i.e., $L = \sup_{\gamma \in \Gamma} |\gamma|$, where $\Gamma \subset C^*$ is the set of all causal chains). $L = 1$ in Fig. 4.1(a) and (c), $L = 2$ in Fig. 4.1(d) and (e), and $L = 3$ in Fig. 4.1(b).

Although we allow V (and hence C) to be infinitely large, we restrict our attention to domains where the constants M, U, and L are all finite.

[3] In the following, sup refers to the mathematical supremum; i.e., the least upper bound of some quantity over a given set. When the sets are finite, the supremum is equivalent to the maximum.

4.3 Parsimonious Covering Algorithms

The main technical contribution of this chapter is a set of algorithms that solve the Parsimonious Covering problem as defined above and are both provably correct and effective. Specifically, given the background knowledge stored in C, and an observed sequence of effects (e.g., a sequence of actions), the algorithms compute every parsimonious explanation (e.g., sequences of inferred intentions). In Fig. 3.1, these algorithms correspond to block arrow A, in which the top-level intentions are inferred based on actions recorded in SMILE. The algorithms are provably sound: any $\langle u \rangle$ computed by the algorithms is guaranteed to be a true top-level cover of $\langle w \rangle$. They are also provably complete: any true top-level cover $\langle u \rangle$ of $\langle w \rangle$ is guaranteed to be found by the algorithms. The algorithm outputs can be filtered for additional criteria like minimum cardinality as a post-processing step. The computational complexity of the algorithms can also be derived in terms of the bound M defined above, and they are essentially fixed-parameter tractable with some caveats detailed below. The important point is that they can be expected to run in a reasonable time frame on the vast majority of problem instances, as borne out by experiments in Chapter 5. The algorithms are presented here; the theorems and proofs are presented in Section 4.4.

Given a causal inference problem, the solution (i.e., the set of all top-level covers) is computed by an algorithm called EXPLAIN (Fig. 4.2), which operates in two phases detailed below. The main inputs to EXPLAIN are the background knowledge contained in C, and the sequence of observations $\langle w \rangle \in V^*$. In CERIL,

1: **procedure** EXPLAIN(CAUSES,M,$\langle w_n \rangle_{n=1}^{N}$)

2: $g \leftarrow$ SSCOVERS(CAUSES, M, $\langle w_n \rangle_{n=1}^{N}$) ▷ Phase 1: Singleton sub-covers

3: $tlcovs \leftarrow \{\}$

4: **for** $t \in$ TLCOVERS(g, N, M,$\langle\rangle$,$\langle 0 \rangle$) **do** ▷ Phase 2: Top-level covers

5: $tlcovs \leftarrow tlcovs \cup \{t\}$

6: **end for**

7: **return** $tlcovs$

8: **end procedure**

Figure 4.2: Explaining an observed sequence.

$\langle w \rangle$ is an observed demonstration. The causal background knowledge is represented operationally in EXPLAIN using a PCT CAUSES function, formally defined as follows:

$$\text{CAUSES}(\langle v \rangle) \stackrel{\text{def}}{=} \{u \in V \mid (u, \langle v \rangle) \in C\}.$$

In other words, CAUSES returns all singleton 1-covers of its input. CAUSES is represented operationally and the domain author is responsible for correctly implementing it. It constitutes the interface between EXPLAIN and the causal knowledge base, and EXPLAIN treats CAUSES as a black box. M, the bound defined in Section 4.2, is also expected as input since it can not be automatically determined from an operational encoding.

In implementing CAUSES, the domain author only needs to enumerate the *direct* causal associations $(u, \langle v \rangle)$ that make up C. EXPLAIN automates the work of *composing* causal associations over multiple layers of causation and multiple sequences of effects, much like a parsing algorithm composes individual grammatical

rules. In other words, the domain author is only responsible for specifying the immediate parent-child relationships that are allowed, not for anticipating any of the multi-layer trees that might need to be constructed to form a covering forest. For example, in the upper left of Fig. 3.1, the domain author must specify that the first two actions on the left can be caused by the mid-level intention directly above them, and that the first two mid-level intentions on left can be caused by the intention above them, and so on, but need not specify that the full sequence of 6 low-level actions can be indirectly caused by the full sequence of the 2 top-level intentions. As such, EXPLAIN is a generic, domain-independent algorithm, that can find parsimonious, high-level covers for long, low-level sequences that need not be anticipated by the domain author or explicitly built in to the background knowledge.

Given a causal inference problem, the solution (i.e., the set of all top-level covers) is computed by EXPLAIN (Fig. 4.2) in two phases. The first phase uses dynamic programming to compute every singleton cover of every contiguous subsequence of the observations. The second phase generates every top-level cover by carefully combining singleton sub-covers from the first phase, pruning out the combinations that are not top-level. The first phase relies on the externally provided CAUSES function, which is EXPLAIN's interface to the causal knowledge base.

EXPLAIN combines the two phases to produce the final set of top-level covers for an observed sequence. The output of EXPLAIN is *tlcovs*, the set of all top-level covers. *tlcovs* can be pruned by additional parsimony criteria as a post-processing step. EXPLAIN invokes sub-routines SSCOVERS to perform the first phase and TLCOVERS to perform the second phase (described below). TLCOVERS is treated as

an iterator construct, so that in rare cases when $|tlcovs|$ is very large, it can be terminated early or pruned by additional parsimony criteria online.

The algorithm ssCovers (Fig. 4.3) takes the same inputs as EXPLAIN: the CAUSES function (which encodes the domain), the bound M defined above, and the observed sequence $\langle w_n \rangle_{n=1}^{N}$. A dynamic programming table g is populated in an incremental, bottom-up fashion. The table entry $g_{j,k}^{\ell}$ accumulates the singleton ℓ-covers (i.e., those whose covering trees have depth at most ℓ) of the observed subsequence from index j to k, namely, $\langle w_n \rangle_{n=j+1}^{k}$. Along with each singleton cover u, its immediate children $\langle v \rangle$ in the covering tree are also stored for use in tlCovers (described later). Each outer iteration of ssCovers (lines 5-18) populates the ℓ-covers during the ℓ^{th} iteration, using the $(\ell - 1)$-covers that were found in the previous iteration. Every $g_{j,k}^{0}$ is initialized with the trivial singleton covers, one for each w_k (lines 2-4). Line 6 initializes the ℓ-covers for the current iteration with those from the previous, since any $(\ell - 1)$-cover is also an ℓ-cover.[4] Lines 7-13 check every subsequence $\langle u \rangle$ of every cover found so far to see if it has a higher-level cause $\tilde{u}$. Line 7 limits the search to $\langle u \rangle$ of length $m \leq M$, since by definition no effect sequence longer than M is present in C. Every such $\langle u \rangle$ will partition a subsequence of the original $\langle w_n \rangle_{n=1}^{N}$ into m disjoint, contiguous, consecutive subsequences, each of which are the leaves of the i^{th} covering tree rooted in u_i. That is, u_i covers $\langle w_n \rangle_{k_{i-1}+1}^{k_i}$, with $0 \leq k_0 < ... < k_m \leq N$. Line 8 enumerates every way $\langle w_n \rangle_{n=1}^{N}$ might be so partitioned. Each such $\langle u \rangle$ is then enumerated on line 9, based on the fact that if u_i

[4]This is because, by definition, an $(\ell - 1)$-cover has depth at most $(\ell - 1)$. Therefore it has depth at most ℓ. So it is also an ℓ-cover. However the converse is not true.

1: **procedure** ssCovers(causes, M, $\langle w_n \rangle_{n=1}^{N}$)

2: **for** $0 \leq j < k \leq N$ **do** ▷ Initialize g^0

3: $g^0_{j,k} \leftarrow \{(w_k, \langle\rangle)\}$ **if** $j+1 = k$ **else** $\emptyset$ **end if**

4: **end for**

5: **for** $\ell \in \langle 1, 2, ... \rangle$ **do** ▷ Populate g bottom-up

6: $g^\ell_{j,k} \leftarrow g^{\ell-1}_{j,k}$ **for** $0 \leq j < k \leq N$ ▷ Initialize g^ℓ to $g^{\ell-1}$

7: **for** $m \in \langle 1, 2, ..., M \rangle$ **do** ▷ Every $(\ell-1)$-cover of every $\langle \hat{w} \rangle \sqsubseteq \langle w \rangle$

8: **for** $0 \leq k_0 < ... < k_m \leq N$ **do**

9: **for** $\langle u \rangle \in \{\langle u_i \rangle_{i=1}^{m} \,|\, \forall i \, \exists \langle v \rangle \, (u_i, \langle v \rangle) \in g^{\ell-1}_{k_{i-1},k_i}\}$ **do**

10: $g^\ell_{k_0,k_m} \leftarrow g^\ell_{k_0,k_m} \cup \{(\tilde{u}, \langle u \rangle) \,|\, \tilde{u} \in \text{causes}(\langle u \rangle)\}$

11: **end for**

12: **end for**

13: **end for**

14: **if** $g^\ell_{j,k} = g^{\ell-1}_{j,k}$ **for** $0 \leq j < k \leq N$ **then** ▷ No new covers found, terminate

15: $g \leftarrow \{g^\ell_{j,k} \,|\, 0 \leq j < k \leq N\}$

16: **return** g

17: **end if**

18: **end for**

19: **end procedure**

Figure 4.3: Singleton Sub-Cover Generation.

has already been found to cover $\langle w_n \rangle_{k_{i-1}+1}^{k_i}$, then it will be contained[5] in $g_{k_{i-1},k_i}^{\ell-1}$. The set on this line draws each u_i from $g_{k_{i-1},k_i}^{\ell-1}$, which contains all singleton $(\ell-1)$-covers for the i^{th} subsequence of $\langle w_n \rangle_{n=1}^{N}$ in the current partition. If the current $\langle u \rangle$ has any causes, they each constitute a new ℓ-cover of $\langle w_n \rangle_{n=k_0+1}^{k_m}$ and are added to g_{k_0,k_m}^{ℓ} (line 10). When the current iteration of the outer loop has identified no new covers, the algorithm can be terminated. The current g^{ℓ} includes all singleton sub-covers from the previous iterations, so it is renamed g and returned (lines 14-17).

The second phase is performed by tlCovers (Fig. 4.4), which generates every top-level cover using the output g of ssCovers. Instead of a **return** statement, tlCovers uses **yield** statements[6] which suspend its execution and pass the current top-level cover to its "caller" (typically a for-loop, as in algorithm explain). When the caller requests the next top-level cover, execution resumes where tlCovers left off and continues until **yield** is encountered again. The **yield** keyword makes tlCovers an iterable construct that a for-loop can invoke to receive parsimonious covers one at a time.

tlCovers is also recursive. It takes as input g (as computed by ssCovers), N (the length of the observed sequence), the bound M as defined above, and two "accumulators" $\langle u_i \rangle_{i=1}^{I}$ and $\langle k_i \rangle_{i=0}^{I}$: The current top-level cover is accumulated in $\langle u_i \rangle_{i=1}^{I}$, and the indices at which it partitions $\langle w_n \rangle_{n=1}^{N}$ are accumulated in $\langle k_i \rangle_{i=0}^{I}$.

[5]Technically, we cannot write $u_i \in g_{k_{i-1},k_i}^{\ell-1}$, since each cell in g actually contains pairs $(u, \langle v \rangle)$. At this stage, we only need one such pair containing u_i, hence the $\langle v \rangle$ is existentially quantified in the set on line 9.

[6]The **yield** construct here is borrowed from Python.

1: **procedure** TLCOVERS(g,N,M,$\langle u_i \rangle_{i=1}^{I}$,$\langle k_i \rangle_{i=0}^{I}$)

2: **if** $k_I = N$ **then yield** $(\langle u_i \rangle_{i=1}^{I}, \langle k_i \rangle_{i=0}^{I})$ ▷ Accumulator covers full input

3: **else** ▷ Accumulator covers leading input subsequence, continue covering

4: **for** $k_{I+1} \in \{k_I + 1, ..., N\}$ **do** ▷ Every possible partition point

5: **for** $(u_{I+1}, \langle v \rangle) \in g_{k_I, k_{I+1}}$ **do**

6: **if** $\exists m < M \, \exists \tilde{u} \, (\tilde{u}, \langle u_{I+1-m}, ..., u_{I+1} \rangle) \in g_{k_{I-m}, k_{I+1}}$ **then**

7: **continue** ▷ Skip current u_{I+1} if result is mid-level

8: **end if**

9: **for** $(\langle \hat{u} \rangle, \langle \hat{k} \rangle) \in$ TLCOVERS$(g, N, M, \langle u_i \rangle_{i=1}^{I+1}, \langle k_i \rangle_{i=0}^{I+1})$ **do**

10: **yield** $(\langle \hat{u} \rangle, \langle \hat{k} \rangle)$ ▷ Recursively cover rest of input sequence

11: **end for**

12: **end for**

13: **end for**

14: **end if**

15: **end procedure**

Figure 4.4: Top-level Cover Generation.

These are called "accumulators" because they accumulate a return value over the course of recursion. They are empty at the shallowest layer of the recursion, they contain a leading portion of a top-level cover mid-way down the recursion, and they contain a full top-level cover at the deepest layers of the recursion, at which point the full top-level cover is passed back up the recursive stack via the **yield** statements. The algorithm is initiated at the top-level of the recursion by calling tlCovers(g,N,M,$\langle\rangle$,$\langle 0\rangle$) in a for-loop, as in explain.

The algorithm starts with the base case by checking whether a cover for the full $\langle w_n \rangle_{n=1}^{N}$ has been accumulated, in which case the final partition point k_I will be N, the full sequence length (line 2). If so, it yields the accumulated cover (line 2). Otherwise, only a leading subsequence of $\langle w_n \rangle_{n=1}^{N}$ has been covered so far, and the algorithm enumerates all options for the next partition point in the tail (line 4). For each option k_{I+1}, it enumerates all singleton covers u_{I+1} that could be appended to the cover accumulated so far (line 5). For each singleton, it checks whether appending would result in a cover that is mid-level, with some higher-level cause $\tilde{u}$ (line 6). If so, the current singleton is skipped (line 7). Otherwise, it is added to the accumulator, and the algorithm is called recursively (line 9) to finish accumulating. Each top-level cover that results is yielded to the caller one by one (line 10).

The first iterates yielded by tlCovers are those where the leading covering trees have the fewest leaves and the trailing covering trees have the most. This is an arbitrary byproduct of the loop order on line 4, since the next k_{I+1} is searched from head to tail. If ordering were important, it could be modified, although it was not explored further in this work. For example, if k_{I+1} started at $\lfloor (k_I + N)/2 \rfloor$ and

worked outwards, the first iterates of tlCovers would tend to have leaves more uniformly distributed among the covering trees.

4.4 Theoretical Results

Here we prove that Algorithm explain is correct and establish its computational complexity characteristics. N, M, U, and L are as defined in Sect. 4.1.

Theorem 1 shows that ssCovers is sound and complete: after the algorithm terminates, each entry in the dynamic programming table $g^{\ell}_{j,k}$ contains all and only the singleton ℓ-covers of $\langle w_n \rangle_{n=j+1}^{k}$.

Theorem 1. *Let g be the return value of* ssCovers(causes, $M, \langle w_n \rangle_{n=1}^{N}$)*, and let $\langle w_n \rangle_{n=j+1}^{k}$ be any subsequence of $\langle w_n \rangle_{n=1}^{N}$. For every $(u, \langle v \rangle) \in g_{j,k}$, the singleton $\langle u \rangle$ covers $\langle w_n \rangle_{n=j+1}^{k}$* (soundness)*. For every singleton cover $\langle u \rangle$ of $\langle w_n \rangle_{n=j+1}^{k}$, $(u, \langle v \rangle) \in g_{j,k}$ for some $\langle v \rangle$* (completeness)*.*

Proof. The proof is by induction on ℓ. After line 4, each $g^{0}_{j,k}$ contains all and only the 0-covers of $\langle w_n \rangle_{n=j+1}^{k}$, namely, the trivial covers where each w_k covers itself. Now assume the inductive hypothesis: on the ℓ^{th} iteration of lines 5-18, every $g^{\ell-1}_{j,k}$ contains all and only the $(\ell-1)$-covers of $\langle w_n \rangle_{n=j+1}^{k}$.

For soundness, consider $(u, \langle v \rangle) \in g^{\ell}_{j,k}$. If it was acquired via line 6, it was already in $g^{\ell-1}_{j,k}$ and covers $\langle w_n \rangle_{n=j+1}^{k}$ by the inductive hypothesis. Otherwise, it was added on line 10 when $k_0 = j$ and $k_m = k$, in which case $(u, \langle v \rangle)$ is in C by the definition of causes. Moreover, $\langle v \rangle$ is in the set on line 9. Therefore each v_i is stored in some $g^{\ell-1}_{k_{i-1},k_i}$, and is a singleton cover by the inductive hypothesis. So

all parent-child relationships in the sub-trees rooted at each v_i are in C as well. Therefore the tree formed by making $\langle v \rangle$ the ordered children of u is also a covering tree, and consequently u is a singleton cover of $\langle w_n \rangle_{n=j+1}^{k}$.

For completeness, suppose that $\tilde{u}$ is a singleton ℓ-cover of some subsequence $\langle w_n \rangle_{n=j+1}^{k}$. Fix an associated covering tree and let $\langle u_i \rangle_{i=1}^{\tilde{m}}$ be the ordered children of $\tilde{u}$ in the tree. By definition, each u_i is the root of a sub-tree with depth at most $\ell - 1$, and $\langle w_n \rangle_{n=j}^{k}$ is the concatenation of the ordered leaves of each sub-tree. Let $\tilde{k}_{i-1} + 1$ and $\tilde{k}_i$ be the starting and ending indices of the leaves of the i^{th} sub-tree. By the inductive hypothesis, each u_i is stored in $g_{k_{i-1},k_i}^{\ell-1}$. Therefore $\langle u_i \rangle_{i=1}^{\tilde{m}}$ will be included in the set on line 9 when $m = \tilde{m}$ and each $k_i = \tilde{k}_i$. Since $(\tilde{u}, \langle u_i \rangle_{i=1}^{\tilde{m}})$ is a parent-child relationship in the covering tree, it is also in C, and by the definition of CAUSES, it will be added to $g_{k_0,k_m}^{\ell} = g_{j,k}^{\ell}$ on line 10. □

To bound the computational complexity of ssCOVERS, we first establish a lemma that bounds the number of singleton covers in each $g_{j,k}^{\ell}$.

Lemma 1. *The cardinality of any $g_{j,k}^{\ell}$ is at most $\mathcal{O}(MU^{M+1}N^{M-1})$.*

Proof. There are $\sum_{m=1}^{M} \binom{k-j-1}{m-1}$ partitions of the form $j = k_0 < ... < k_m = k$ splitting $\langle w_n \rangle_{n=j+1}^{k}$ into m subsequences. $\sum_{m=1}^{M} \binom{k-j-1}{m-1}$ is $\mathcal{O}(\sum_{m=1}^{M} \binom{N-1}{m-1})$, which simplifies[7] to $\mathcal{O}(MN^{M-1})$. For every such partition, each of the m subsequences has at most U singleton covers, so there are at most U^m associated covers. Therefore there are $\mathcal{O}(MU^M N^{M-1})$ possible covers $\langle v \rangle$ of $\langle w_n \rangle_{n=j+1}^{k}$. Each element of $g_{j,k}^{\ell}$ pairs one of at most U singleton covers u with one of these $\langle v \rangle$. Therefore $|g_{j,k}^{\ell}|$ is

[7] The term $\binom{A}{B}$ is $\mathcal{O}(A^B)$ since it has a numerator with B factors, each at most A.

$\mathcal{O}(MU^{M+1}N^{M-1})$. □

The upper bound in Lemma 1 is rather loose. It counts every possible pairing of u's with $\langle v \rangle$'s, ignoring whether each $(u, \langle v \rangle)$ is actually in C, resulting in a significant overestimate. Equipped with Lemma 1, we can prove that the runtime of ssCovers is polynomial in N.

Theorem 2. *The worst case complexity of* ssCovers(causes, $M, \langle w_n \rangle_{n=1}^{N}$) *is polynomial in* N.

Proof. Lines 2-4 take $\mathcal{O}(N^2)$ steps since j and k range from 0 to N. Line 10 takes time constant in N to compute causes. The cardinality of the resulting set is $\mathcal{O}(U)$ so the union operation takes time $\mathcal{O}(U)$ using an efficient (e.g., hash-based) set implementation for each $g_{j,k}^{\ell}$. The cardinality of the set on line 9 is $\mathcal{O}((MU^{M+1}N^{M-1})^M) = \mathcal{O}(M^M U^{M^2+M} N^{M^2-M})$, since each u_i is chosen from one of at most M sets $g_{k_{i-1},k_i}^{\ell-1}$, and each $g_{k_{i-1},k_i}^{\ell-1}$ contains $\mathcal{O}(MU^{M+1}N^{M-1})$ options for u_i by Lemma 1. Line 8 iterates over $\binom{N+1}{M+1}$ partitions which simplifies to $\mathcal{O}(N^{M+1})$. Line 6 copies $\mathcal{O}(N^2)$ sets each of size at most $\mathcal{O}(MU^{M+1}N^{M-1})$, again by Lemma 1. Therefore the total complexity of lines 6-17 is

$$\mathcal{O}(MU^{M+1}N^{M+1} + M^{M+1}U^{M^2+M}N^{M^2+1}(U + X)),$$

where X is the complexity of causes. Although line 5 has no termination condition, all singleton covers have depth at most L, where L is the bound on causal chain depth defined previously, assumed to be finite.Since correctness was shown in Theorem 1 by induction on ℓ, the termination check on line 14 will be satisfied after at most L

iterations. Therefore the total complexity of the algorithm is

$$\mathcal{O}(L(MU^{M+1}N^{M+3} + M^{M+1}U^{M^2+M}N^{M^2+1}(U+X))).$$

□

The exponential dependence on M^2 is a theoretical concern but we do not consider it a serious defect in practice. The main reason is that our empirical run times are very reasonable even on fairly long demonstrations (see Sect. 5.1). In addition, we believe that in practice M is typically rather small. It is at most 6 in our own robotics domain and the unrelated Monroe Plan Corpus (see Chapter 5), and most child sequences have length closer to 2 or 3.

Next, we prove that the second phase of causal inference is also correct. Note that top-level-ness is our primary parsimony criteria, and implies irredundancy except in rare pathological cases. As such, Theorem 3 shows that tlCovers yields all and only the parsimonious covers of an observed sequence.[8]

Theorem 3. *Let g be the return value of* ssCovers(causes, $M, \langle w_n \rangle_{n=1}^{N}$). *Iteration over* tlCovers($g, N, M, \langle\rangle, \langle 0 \rangle$) *yields all* (completeness) *and only* (soundness) *the top-level covers of $\langle w_n \rangle_{n=1}^{N}$.*

Proof. For completeness, suppose $\langle u_i \rangle_{i=1}^{\tilde{I}}$ is a top-level cover of $\langle w_n \rangle_{n=1}^{N}$. We can show that $\langle u_i \rangle_{i=1}^{\tilde{I}}$ is yielded by a reverse induction on I ranging from $\tilde{I}$ to 0. Let $0 = \tilde{k}_0 < ... < \tilde{k}_{\tilde{I}} = N$ be the partition of $\langle w_n \rangle_{n=1}^{N}$ induced by $\langle u_i \rangle_{i=1}^{\tilde{I}}$, i.e., u_i covers

[8] Before pruning with additional parsimony criteria such as minimum cardinality, which can be done in linear time.

$\langle w_n \rangle_{n=\tilde{k}_{i-1}+1}^{\tilde{k}_i}$. For the base case, when $I = \tilde{I}$, the call TLCOVERS$(g, N, M, \langle u_i \rangle_{i=1}^{I}, \langle \tilde{k}_i \rangle_{i=1}^{I})$ will execute line 2 since $\tilde{k}_{\tilde{I}} = N$. For the inductive case, suppose that TLCOVERS$(g, N, M, \langle u_i \rangle_{i=1}^{I+1}, \langle \tilde{k}_i \rangle_{i=1}^{I+1})$ yields $\langle u_i \rangle_{i=1}^{\tilde{I}}$; we must show the same for TLCOVERS$(g, N, M, \langle u_i \rangle_{i=1}^{I}, \langle \tilde{k}_i \rangle_{i=1}^{I})$. During this call, the iteration on line 4 will eventually reach $k_{I+1} = \tilde{k}_{I+1}$ since $\tilde{k}_{I+1} \in \{\tilde{k}_I + 1, ..., N\}$. Since u_{I+1} is a singleton cover of $\langle w_n \rangle_{n=\tilde{k}_I+1}^{\tilde{k}_{I+1}}$, it is stored in $g_{\tilde{k}_I+1, \tilde{k}_{I+1}}$ and will be included in the iteration on line 5. Since $\langle u_i \rangle_{i=1}^{\tilde{I}}$ is top-level, the check on line 6 will fail. Therefore the iteration on line 9 will be reached, and by the inductive hypothesis, it will yield $\langle u_i \rangle_{i=1}^{\tilde{I}}$, which gets passed up the recursion. Running the induction through to $I = 0$, we have that TLCOVERS$(g, N, M, \langle\rangle, \langle 0 \rangle)$ yields $\langle u_i \rangle_{i=1}^{\tilde{I}}$.

For soundness, let $\langle u_i \rangle_{i=1}^{\tilde{I}}$ be any iterate yielded by TLCOVERS$(g, N, M, \langle\rangle, \langle 0 \rangle)$. We must show that it is a top-level cover. Each u_i was accumulated at some recursion depth as some iterate u_{I+1} in line 5. Since u_{I+1} is drawn from $g_{k_I, k_{I+1}}$, it is a singleton cover of $\langle w_n \rangle_{n=k_I+1}^{k_{I+1}}$, and therefore $\langle u_i \rangle_{i=1}^{\tilde{I}}$ is a cover for the full $\langle w_n \rangle_{n=1}^{N}$. If it were mid-level and not top-level, there would be some subsequence $\langle u_i \rangle_{i=I-m}^{I+1} \sqsubseteq \langle u_i \rangle_{i=1}^{\tilde{I}}$ for some $m < M$ and $I < \tilde{I}$ that could be covered by some higher-level cause $\tilde{u}$. But then the check on line 6 would be true at recursion depth I, so $\langle u_i \rangle_{i=1}^{I+1}$ would have never been passed to the recursive call on line 9, and $\langle u_i \rangle_{i=1}^{\tilde{I}}$ would have never been yielded. Therefore $\langle u_i \rangle_{i=1}^{\tilde{I}}$ must be top-level. □

Having shown correctness, it remains to characterize the complexity of TLCOVERS.

Theorem 4. *The worst-case complexity of* TLCOVERS$(g, N, M, \langle\rangle, \langle 0 \rangle)$ *is polyno-*

mial in N and T, where T is the number of top-level covers.

Proof. The recursive execution trace of TLCOVERS$(g, N, M, \langle\rangle, \langle 0\rangle)$ can be viewed as a tree, where each node corresponds to some $\langle u_i\rangle_{i=1}^{I+1}$ that gets checked on line 6, and if it passes the check, gets passed to the recursive call on line 9. Note there is no connection between this recursion tree and the notion of a covering tree. The depth in this tree is equal to the depth in the recursion. The node for any $\langle u_i\rangle_{i=1}^{I+1}$ is a child of the node for $\langle u_i\rangle_{i=1}^{I}$, and the root is associated with $\langle\rangle$. Line 6 compares a length-$\mathcal{O}(M)$ sequence against each of $\mathcal{O}(MU^{M+1}N^{M-1})$ elements in each of M sets $g_{k_{I-m},k_{I+1}}$ by Lemma 1, so the worst-case run time of the algorithm is proportional to the size of the tree times $M^3U^{M+1}N^{M-1}$.

The leaves of the tree can be split into two groups. The "good" leaves are top-level covers of the full $\langle w_n\rangle_{n=1}^{N}$, which get yielded and passed up the recursion on line 2. The "bad" leaves are mid-level covers of leading subsequences of $\langle w_n\rangle_{n=1}^{N}$, which fail the check on line 6 and prevent a recursive call. Likewise, the nodes of the tree can be split into two groups: the "good" nodes are those from which good leaves are reachable, and the "bad" nodes are the rest. It follows that all descendants of a bad node are also bad.

There are T good leaves, and each is reachable from at most N nodes along the path from the root, since the length of a cover is never more than the length of the covered sequence. So there are $\mathcal{O}(TN)$ good nodes. Lines 4 and 5 enumerate $\mathcal{O}(MU^{M+1}N^{M})$ iterates by Lemma 1, so each good node has $\mathcal{O}(MU^{M+1}N^{M})$ bad children, and each bad child is the root of a sub-tree containing only bad nodes.

Moreover, every bad node is in some such sub-tree. The depth of these sub-trees is at most M, since line 6 identifies any mid-level cover within M recursive steps. The branching factor of the sub-trees is again $\mathcal{O}(MU^{M+1}N^M)$, so the size of the sub-tree is $\mathcal{O}(M^M U^{M^2+M} N^{M^2})$. Therefore there are at most $\mathcal{O}(TM^M U^{M^2+M} N^{M^2})$ bad nodes. The total size of the entire tree is the sum of good and bad node counts, so the total complexity of the algorithm is

$$\mathcal{O}(M^3 U^{M+1} N^{M-1}(TN + TM^M U^{M^2+M} N^{M^2})). \qquad \square$$

Another theoretical concern is that the number of top-level covers T is not independent from the length of the demonstration sequence N, and may have exponential dependence on N, thereby concealing a worse-than-polynomial run time. Indeed, our empirical results do show that in complex domains, T can be very large in some cases (Sect. 5.1.2). However, T was rarely large enough that EXPLAIN ran for an impractical amount of time, and the size of the output $|tlcovs|$ could be mitigated by pruning with additional parsimony criteria.

Lastly, an important corollary of the results in this section is that, if CAUSES formally inverts the HTN planning operators, then EXPLAIN formally inverts the HTN planning algorithm. To our knowledge, this is the first provably correct inversion of HTN planning.

Chapter 5: Experimental Validation

The previous chapter presented the algorithms used by CERIL for intention inference and derived their formal properties. However, theory does not always translate to practice. This chapter presents physical and simulation experiments that empirically validate CERIL as a performant, pliable system for cognitive-level imitation learning in practice. The physical experiments were conducted using a Baxter robot with the disk drive dock and other objects we fabricated for testing. These experiments focused primarily on the overall imitation learning pipeline. The simulation experiments focused primarily on CERIL's abductive inference algorithms in particular. In addition to using demonstration data from our own assembly and maintenance domain, the simulation experiments used a large 3^{rd}-party dataset called the Monroe Plan Corpus (described below), which is a standard benchmark in the field of plan recognition. Sect. 5.1 presents these results.

In addition to assessing performance, an empirical study was conducted to compare different parsimony criteria. These experiments showed that the number of parsimonious explanations for a demonstration was highly sensitive to the parsimony criterion applied. Furthermore, some criteria that are preferred in other application areas such as medical diagnosis proved sub-optimal for intention inference, and a

new criterion introduced here proved particularly advantageous in certain regards. These results are presented in Sect. 5.2.

5.1 Overall System Performance

5.1.1 Imitation Learning Trials

With regards to robot imitation learning and execution as a whole, this work focused on two questions: How quickly can a robot interpret and imitate a demonstration? How often and at what points in the process does it fail? These questions were answered with a series of imitation learning trials using the dock maintenance scenario and a toy block stacking scenario. The tasks used in these trials are intentionally simple, intended solely to establish that EXPLAIN can work effectively in a real-world robotics application.

First, for the dock maintenance scenario, in each trial, the Baxter robot interpreted a demonstration of a maintenance skill, and then imitated that skill in a new situation, where the initial configuration of drives, slots, and LED indicators was different than what had been observed in the demonstration. Each trial was timed, and successes and failures were recorded.

More specifically, four different skills were taught to the robot:

- Discarding a faulty drive (i.e., a drive next to a red LED)
- Discarding a faulty drive and replacing it with a spare drive on top of the dock
- Discarding a faulty drive and replacing it with a functional drive (i.e., a drive

next to a green LED)

- Swapping the slot positions of a faulty drive and a functional drive.

Each skill was demonstrated twice in SMILE, using different initial states for the dock each time, and in some cases different action sequences, to introduce more variety into our testing set. EXPLAIN was used to infer intentions in each demonstration (i.e., learning was always done with a single demonstration). Finally, the robot was asked to imitate each demonstration 4 times, again using different initial physical dock states each time. The result is 8 distinct demonstrations total and 32 imitation trials total (8 demonstrations for learning $\times$ 4 new initial states per demonstration for imitation). In every demonstration and trial, the initial dock states were automatically and randomly generated, varying the number and position of spare drives, which slots were occupied, and which LEDs were red. Fig. 5.1 depicts a SMILE demonstration and the robot imitating it.

Second, in the toy block scenario, blocks were stacked to form letters of the English alphabet. Three demonstrations were recorded in SMILE: (1) Forming "IL," (2) forming "AI," and (3) forming "UM." Again, the robot was asked to imitate each demonstration 4 times, using different random initial block placements each time. The result is 12 additional trials. Snapshots of a SMILE demonstration and subsequent imitation for the "UM" block stacking skill are shown in Figs. 5.2 and 5.3

In both dock and block trials, every trial is independent from the others: The robot does not use other demonstrations or previous trials to improve its perfor-

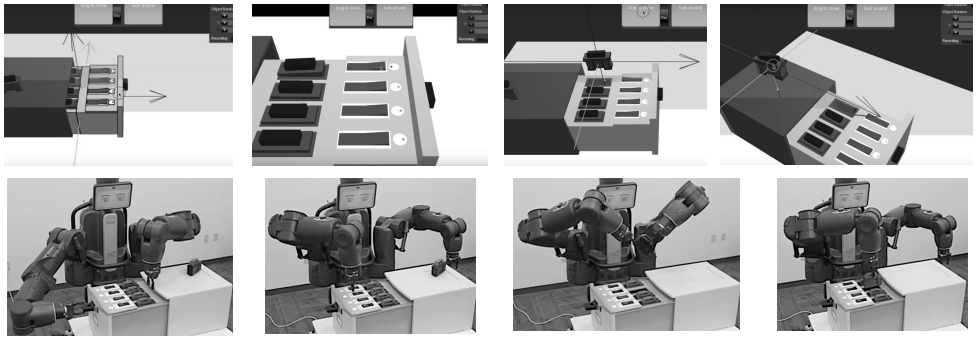

Figure 5.1: Screenshots from a task demonstration in SMILE (top row) and the corresponding subsequent robotic imitation of the task (bottom row). Time flows from left to right. During imitation, a different LED is red corresponding to a different drive that should be replaced than the one observed in SMILE, and the robot correctly generalizes to this situation. Also, among other instances of bimanual coordination, the robot generalizes by performing a hand-off required by its embodiment that was not demonstrated in SMILE (bottom row, third panel from left).

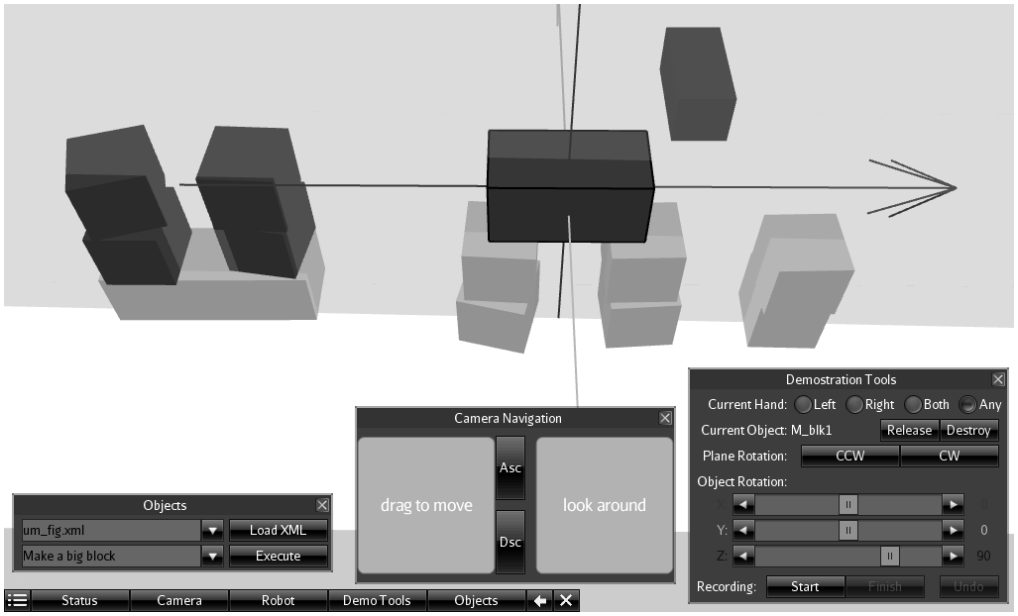

Figure 5.2: A snapshot of the "UM" block stacking demonstration recorded in SMILE.

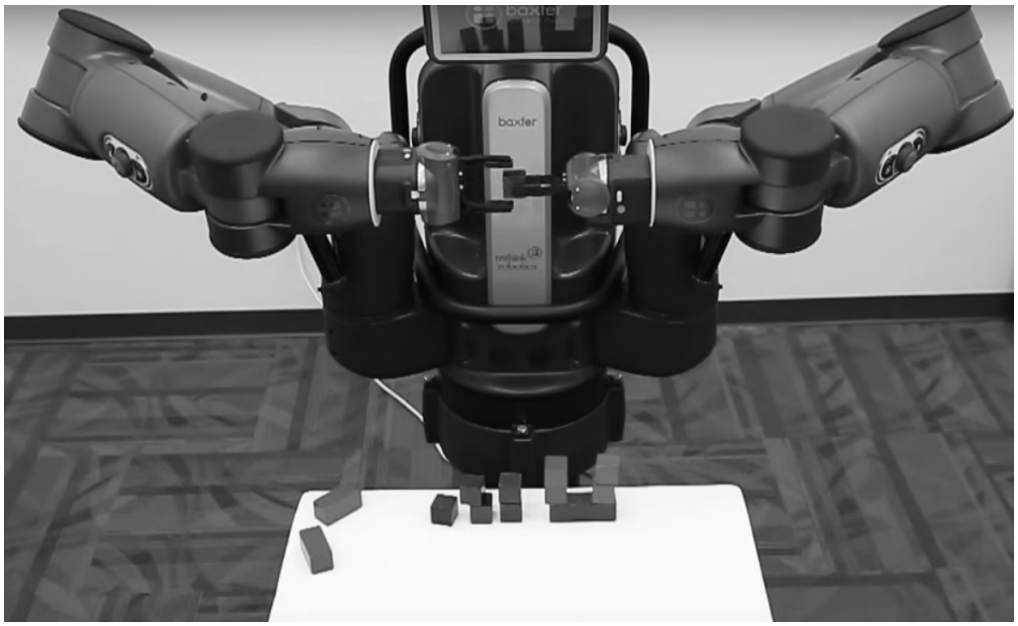

Figure 5.3: A snapshot of the "UM" block stacking skill imitated by a Baxter robot using CERIL.

mance, it always imitates "from scratch" using only the human authored domain knowledge and a single demonstration. Multiple trials for each demonstration were used solely to increase the sample size for the experiment.

Related work by colleagues at the University of Maryland verified that these skills are reasonable targets to be learned from a single demonstration. In that work, a small sample of human participants ($n = 5$) performed similar dock maintenance and block stacking tasks.[1] It was found that about 85% of the time they would correctly perform each task after observing only a single demonstration of it [69].

The first step of robotic imitation is interpretation of the demonstration through causal inference (arrow A in Fig. 3.1). This step is performed by EXPLAIN. Table 5.1 shows the results of running EXPLAIN on every demonstration. N is the size of the input, i.e., the number of discrete steps (grasps, releases, toggle switches) recorded in the SMILE event transcript. Runtime is the running time in seconds of EXPLAIN.[2] TL indicates the total number of top-level covers found.[3] MC indicates the number of minimal cardinality top-level covers found.

On every demonstration, EXPLAIN terminated in under 1 second (Table 5.1), so time complexity was very reasonable in practice, at least for the simple tasks that we used here. This is especially acceptable given that intention inference is

[1]Thanks to Theresa Hauge and Rodolphe Gentili for designing and administering these experiments with human participants.

[2]Run times were measured on one 2.4GHz Intel Core i7 CPU.

[3]As mentioned in Chapter 4, top-level covers will generally also be irredundant, except in rare pathological cases like that shown in Fig. 4.1(c).

Table 5.1: EXPLAIN Performance

Demonstration	N	Runtime	TL	MC
Remove red drive (1)	7	0.011992	4	2
Remove red drive (2)	10	0.012352	4	2
Replace red with green (1)	15	0.032965	8	2
Replace red with green (2)	15	0.033109	8	2
Replace red with spare (1)	14	0.032204	8	2
Replace red with spare (2)	14	0.032235	8	2
Swap red with green (1)	16	0.025078	2	1
Swap red with green (2)	16	0.025020	2	1
Toy Blocks (IL)	24	0.124730	256	1
Toy Blocks (AI)	30	0.281449	1024	1
Toy Blocks (UM)	39	0.971143	8192	1

only required once per demonstration, and then immediately allows generalization to a variety of new situations. We also found that while the number of top-level covers can become rather large, pruning by minimum cardinality was sufficient to nearly uniquely determine a suitable cover.[4]

After EXPLAIN processed each demonstration, an arbitrary minimum cardinality top-level cover was selected as the robot's representation of the learned skill. This cover was then used for imitation (i.e., object matching, HTN planning, and physical robot execution) in the new initial conditions for the current trial. Manual inspection confirmed that in all dock and block trials, the robot was generating a suitable, correct plan of low-level actions to execute. The plan was correct in that, barring sensorimotor errors such as failed grasps, the planned actions would accomplish the skill that had been demonstrated. Sample videos of SMILE demonstrations

[4]In the cases where there were two minimum cardinality covers, the difference was inconsequential: see the example in Sect. 3.3.

and the corresponding robotic executions of learned skills can be found online.[5]

Although the plans were correct, the physical robot failed mid-way through plan execution in 3 of the 32 dock maintenance trials (∼9%) due to sensorimotor errors. In one failed trial, a drive was poorly aligned with a slot, and became stuck halfway down during insertion. In another, a drive was dropped during a hand-off, and in the third, a drive on top of the dock was knocked over during an arm motion. These issues are due to a combination of the simplistic sensorimotor processing routines and limited accuracy in Baxter's motor control as compared to more expensive robots (although people can make these sorts of errors too). Regardless, the key result is that the cognitive-level learning process and subsequent imitation plans were correct in 100% of the trials. Since sensorimotor processing is not the primary focus in this work, the execution fail rate of the physical robot is arguably not a significant objection to the CERIL framework (although efforts are ongoing to improve the sensorimotor processing).

In the block stacking trials, since each letter is made of three to eight blocks (as shown in Figs. 5.2 and 5.3), these demonstrations tend to involve more steps than dock maintenance and took longer for EXPLAIN to process. The increased number of steps also results in combinatorially more top-level covers, as evident in Table 5.1. Nevertheless, minimum cardinality was an effective parsimony criterion for mitigating this effect. Also, although block stacking is conceptually simpler than dock maintenance, it poses a greater sensorimotor challenge, because the blocks

[5]See `https://www.cs.umd.edu/~reggia/supplement/index.html`

are smaller than the drives and leave less room for error in visual processing and motor control. Stacking multiple blocks is also more difficult since small noise in the placement of a bottom block makes successful placement of the blocks above less likely. Lastly, the increased number of actions required to complete the task presents more opportunities for sensorimotor errors. We found that across all block stacking trials, 97% of the individual pick and place block movements were successful, 87% of the individual letters were successfully constructed, and 75% of the trials were successfully completed in full. Again, these were errors during physical execution, not errors in the plans that CERIL produced for imitation, which were always valid plans for successfully imitating the skill.

5.1.2 Monroe Plan Corpus Experiments

A more extensive battery of experiments[6] was conducted on the much more complex Monroe Plan Corpus (MPC), a well-known benchmark from the field of plan recognition [18], to systematically assess the performance of EXPLAIN. These experiments focused on several questions: Are the theoretical results in Chapter 4 consistent with empirical evidence in a larger, more complex problem domain? What is the empirical average case complexity of EXPLAIN? Is EXPLAIN effective on problem domains designed by other third party domain authors who were not involved in this work? These experiments did not involve the use of a physical robot, planning, or execution; they only measure the performance of EXPLAIN during

[6]The remaining experiments reported below were all run on a workstation with twelve 3.5GHz Intel Xeon CPU cores and 32GB of RAM, taking approximately three days in total.

intention learning.

The MPC is based on an HTN planning domain for emergency response in Monroe County, New York. Top-level goals, such as clearing a car wreck or repairing a power line, are accomplished through sequences of lower-level tasks such as navigating a snow plow or calling a power company. The corpus consists of 5000 planning examples, each of which is the HTN plan tree for a randomly chosen top-level goal in a randomly generated initial state. We refer to the top-level goal in each example as the "original" or "ground-truth" top-level goal. The low-level actions in each plan tree served as the input "demonstration" to EXPLAIN (the ground-truth top-level goal and intermediate tree nodes were withheld). The ground-truth top-level goal served as the target output that EXPLAIN was expected to compute.

In order to use EXPLAIN on this corpus, it was necessary to implement a CAUSES function. The implementation I produced in this work is conceptually equivalent to the original HTN planning domain written by the third party author of the MPC, with case-by-case logic for each planning operator, the only difference being that causes (i.e., parent tasks) are computed from their effects (i.e., child tasks) and not vice versa. A listing of the full set of causal relations is available in Appendix A.2.

Another issue is that the MPC does not retain the initial or intermediate states present when the example was generated. Only the planned tasks and actions are included, without any state information. However, the formalization of intentions as described in Sect. 4.1 expects state information to be included, because some causes cannot be uniquely determined without it. The MPC is no exception. For exam-

ple, there is a task `(clean-up-hazard ?from ?to)` where variables `?from` and `?to` are locations specifying a road with hazardous conditions. This task may cause a single action `(!call fema)`. Since the parent parameters do not occur in the child, they cannot be inferred solely from the `!call` action. One possible work-around is to enumerate every possible `?from`, `?to` pair in the domain, and return every possible pairing in the output of CAUSES. This will lead to the correct explanation but also many other explanations, reducing the precision of EXPLAIN. This is compounded combinatorially when several such instances occur in sequence. In contrast, inspecting the state can reveal which road was hazardous, enabling CAUSES to only return the correct parent and no others. Fortunately, since the MPC uses a descriptive encoding, it is possible to partially reconstruct the states using an automated procedure which cross-references the planning operator descriptions in the domain definition (detailed in Appendix A.3). This procedure was applied as a preprocessing step to every example in the corpus. Each partially reconstructed state was paired with its corresponding low-level action, according to the formalization described in Sect. 4.1, before being passed as input to EXPLAIN.

EXPLAIN was tested on each and every example in the corpus as follows. The ground truth top-level goal (and plan tree) were withheld, and EXPLAIN was only provided with the low-level state and action sequence as input. The output of EXPLAIN, *tlcovs*, was a set of top-level covers for the actions. To gauge correctness, the experiment checked whether *tlcovs* included the ground-truth top-level goal. To gauge precision, the experiment counted how many other top-level covers were also included.

Running times of EXPLAIN on each test example were also recorded. If EXPLAIN was still running after 10 minutes, it was terminated early. This occurred in 162 of the 5000 examples (3.2%), which are excluded from the results reported below. Manual inspection revealed that these long-running cases occurred when partially reconstructed states still did not possess adequate information to significantly narrow down possible parent parameters, leading to the combinatorial explosion described above in the most extreme cases. Aside from this small portion of the dataset, run time was generally quite reasonable: On the 4838 examples allowed to run to completion, EXPLAIN required an average of 20 seconds. It was found that *tlcovs* included the original top-level goal in 4796 testing examples out of the 4838 that did not time out (∼99.1% correct).[7]

On the other hand, *tlcovs* was very imprecise, often containing more than 1000 possible explanations. Again, the combinatorial explosion of top-level covers occurred because some MPC plan operators are designed such that many different parameterizations of a parent task can cause the same child task, as mentioned. When this occurs multiple times in a sequence, the number of valid covers at the

[7]Given the theoretical results in Chapter 4, if CAUSES is implemented correctly, then *tlcovs* should really include the ground-truth 100% of the time. Manual inspection of a sample of failures suggests that the error is in fact in the MPC itself, and not in the implementation of CAUSES. In some of the original plan trees, the parameters of some child tasks are inconsistent with their parents and would not accomplish the goal (see Appendix A.4). Unfortunately this data set is no longer supported, and since the repair of such errors in this dissertation might be viewed as introducing a post hoc bias, we can simply note here that the 99.1% success rate should be viewed as a lower bound on true performance, which may even be somewhat better than reported.

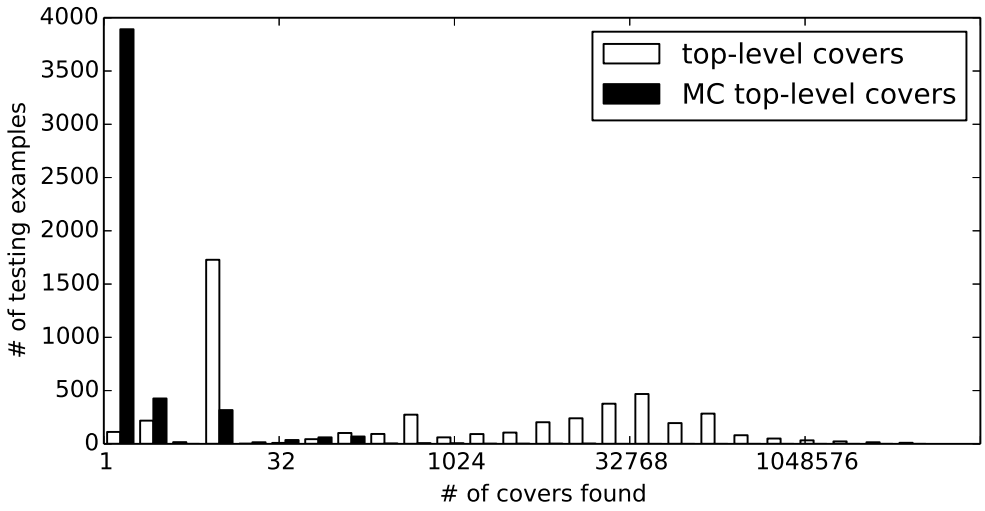

Figure 5.4: Histogram comparing the precision of *tlcovs* before and after pruning based on minimum cardinality.

next layer up grows exponentially. By pruning *tlcovs* with the additional parsimony criterion of minimum cardinality, precision was substantially improved: in 3898 examples of the 4838 that ran to completion (∼80%), there was only one minimum cardinality top-level cover (the ground-truth one); in 4354 examples (∼90%), there were at most 12. This improvement is illustrated in Fig. 5.4.

Pruning *tlcovs* by minimum cardinality did not come with any price to correctness either: this is because every example in the corpus is generated from a single top-level task. Since all ground-truth top-level covers are singletons, it is necessarily true that they will be included in the minimum cardinality subsets of *tlcovs*. By the same reasoning, all ground-truth top-level covers will necessarily be included in the irredundant subset of *tlcovs*. Moreover, minimum cardinality covers are necessarily irredundant. So, despite the importance of irredundancy in previous work on PCT,

there was no perceived need to prune by irredundancy in these experiments.

Table 5.2 is included as a point of reference, which lists performance metrics for past work using the MPC. However, a direct comparison is difficult since past work used different experimental setups and metrics. The accuracy metric reported by Raghavan and Mooney [104] gives partial credit for predictions when not all parameters are correct. The method of Tecuci and Porter [126] only measures correctness of top-level task *schema* (i.e., task names but not parameter values). The same is true for the metric reported by Blaylock and Allen [19]; in cases where the schema were correct, on average only 41.4% of the parameters were also correctly identified. On the other hand, these metrics were all calculated with methods that make ≤ 4 predictions per example, so in many cases they are substantially more precise than the PCT approach used in this work.

Table 5.2: Performance Comparison, Monroe Plan Corpus

Method	Performance
EXPLAIN	99.1%
Raghavan and Mooney [104]	98.9%
Tecuci and Porter [126]	99.8%
Blaylock and Allen [19]	97.4%

5.2 Empirical Comparison of Parsimony Criteria

The limited precision of EXPLAIN reported above stems from the fact that in any causal reasoning problem, there may be more than one valid explanation for the observed evidence. Some of these explanations are better than others, but the formal criterion for "goodness" that is most appropriate is often unknown a priori

and potentially domain-specific. The intention inference problem is no exception, since depending on the current world state, the same high-level intentions/goals may cause different action sequences. Similarly, different high-level intentions/goals can potentially cause the same action sequence. This ambiguity can lead to a plethora of valid explanations and makes the intention inference problem non-trivial, even when actions and world states are perfectly observable and causal relations are provided as background knowledge. It has long been recognized that when applying abductive inference systems to new problem domains, it is important to conduct a systematic comparison of various technical criteria of plausibility to determine which is optimal [98, 130]. However, in the experiments of Sect. 5.1, the causal reasoning system was only tested using one of the simplest such criteria (i.e., minimum cardinality), without considering other more nuanced alternatives from the literature, or new previously unconsidered alternatives.

This section describes a systematic comparison conducted as part of this dissertation work, which empirically compares several alternative technical notions of plausibility, old and new, in the context of intention inference [68]. PCT is particularly advantageous here as the framework for these experiments, since it casts plausibility in terms of *parsimony*, which allows one to easily shift between different criteria for exactly what makes an explanation parsimonious. The comparison conducted here uses both the robotic imitation learning domain and the third party Monroe County Corpus. It was found that the choice of criterion can have a significant impact on performance, but when the optimal criterion is employed, PCT

is quite effective[8]. In particular, while there are good theoretical reasons for basing explanation construction primarily on the form of parsimony known as irredundancy [98], it was found that in the case of intention inference, criteria other than irredundancy are almost as accurate and qualitatively more precise. The optimal criteria also depends on whether plans are always caused by a single top-level intention or might be caused by a top-level *sequence* of intentions. Most significantly, it was found that a previously unconsidered criteria, defined below, is competitive with and often superior to other previously considered criteria from the literature.

5.2.1 Parsimony Criteria Considered

The following criteria were compared in this experiment, some of which were used in past work on PCT:

- *minimum cardinality* (MC): The *cardinality* of a cover $\langle u \rangle$ is simply the number of elements in the sequence.
- *irredundancy* (IR): A cover $\langle u \rangle$ of $\langle w \rangle$ is *irredundant* if no proper subsequence of $\langle u \rangle$ is also a cover of $\langle w \rangle$.
- *maximum depth* (MD): A *causal chain* is a path from a root to a leaf. The

[8]In PCT, the optimality of a particular explanation is based on a parsimony criterion. But the optimality of a criterion itself is not measured by how parsimonious it is (which would be circular) - it is measured with independent metrics such as accuracy (i.e., how often the parsimonious explanations, according to this criterion, match a known ground truth) or specificity (i.e. how many valid parsimonious explanations there are).

covers with the deepest causal chains are considered most parsimonious. The idea is that the roots of deeper chains encode more information (i.e., a larger covering tree) per root node.

- *minimax depth* (XD): For each cover we can measure its shallowest causal chain. Then we can compare this to the shallowest causal chains of other covers. The covers whose shallowest causal chains are deepest overall are considered most parsimonious.

- *minimum parameters* (MP): For each cover we can count the number of distinct parameter values that occur. The covers with the fewest distinct parameter values are considered most parsimonious. For example, in the Monroe County Corpus, the cover

  ```
  (get-to wcrew1 mendon),(clear-road-wreck hamlin rochester)
  ```

  has four distinct parameter values (`wcrew1`, `mendon`, `hamlin`, and `rochester`), whereas the cover

  ```
  (get-to wcrew1 mendon),(clear-road-wreck mendon rochester)
  ```

  only has three (`wcrew1`, `mendon`, and `rochester`), so the latter is more parsimonious. This criterion favors more cohesive explanations, as can be seen in the foregoing example: getting the work crew to `mendon` is related to clearing a road wreck near `mendon`, whereas it would be unrelated to clearing a road wreck near `hamlin`.

- *maximum forest size* (FSX): For each cover, we count all nodes in the covering forest. The idea is that covers with the maximal node count encode more information (i.e., more non-root nodes) in the same number of roots.

- *minimum forest size* (FSN): Instead of maximal forest size, minimal forest size is considered the most parsimonious, in the sense that the total number of causal links contributing to the explanation is smallest.

In all experiments, any given criterion from the list above was always applied to the set of *top-level* covers returned by EXPLAIN.

Minimum cardinality and irredundancy have been widely used in past abductive AI systems. Some automated planning research has considered minimum forest size, but only as related to planning from high-level tasks *to* low-level actions - not in the opposite direction as is done here [129]. To my knowledge, the other alternative criteria - in particular, minimum parameters - are new and explored here for the first time. Regardless of which criterion is adopted, PCT may still return more than one valid, parsimonious explanation for a given plan, rather than a single "optimal" interpretation, so an empirical comparison is needed.

5.2.2 Testing Data and Performance Metrics

The parsimony comparison experiments used the same two domains from Section 5.1: the robotic imitation learning domain developed as part of this work [67,69], and the third-party Monroe County Corpus, a well-known benchmark for plan recognition [18]. As described in previous sections, the robotic domain models a tabletop

workspace environment. Demonstrated skills involve stacking toy blocks and maintaining a hard-drive dock. This domain includes low-level observable actions such as grasping and releasing objects, and higher-level intentions/goals such as opening drawers, toggling switches, handing objects between grippers, and so on. Causal interpretation of observed demonstrations produces novel top-level intention sequences that were not pre-specified in the knowledge base, constituting a newly learned skill. The modest set of test data contains eleven examples of observed demonstrations: eight for various dock maintenance skills, and three for stacking various block configurations. As described in Sect. 5.1.2, the Monroe domain models an emergency response team based in Monroe County in upstate New York. For example, the intention to clear a car wreck might cause a sequence of sub-intentions such as getting patients into an ambulance and getting the ambulance to a hospital. The sub-intention of getting patients into an ambulance might cause its own sequence, such as getting an EMT into the ambulance and driving the ambulance to the scene. The corpus contains a knowledge base defining all of these causal relationships, and 5000 automatically generated plans (i.e., sequences of observed actions).

In both datasets, every observed plan is annotated with the ground truth top-level intentions from which it was generated, which can be used for quantitative comparison. To evaluate a given criterion, the ground truth top-level goals were withheld, and EXPLAIN was invoked on the observed actions. The parsimonious covers found by EXPLAIN were then compared with the ground-truths. Two performance metrics were used to evaluate each criterion. The first metric was *accuracy*: how often do the "parsimonious" covers of an observed plan, according to that crite-

rion, include the ground truth explanation? The second metric was *specificity*: how many parsimonious covers are found in all? Many more may be found than just the single ground truth. A perfectly accurate and specific criterion should compute a single top-level cover, namely the ground truth one, and no others.

For each criterion, EXPLAIN was invoked on every observed plan in each dataset, and the performance metrics for that criterion were calculated. In the large majority of cases, an individual plan was processed in a matter of seconds, although as mentioned earlier, on 162 (3.2%) of the 5000 plans in the Monroe corpus, EXPLAIN was still running after 10 minutes and was terminated early. These plans were excluded from computation of the performance metrics.

5.2.3 Learning Novel Intention Sequences in the Monroe Domain

In the original Monroe corpus, each plan was generated from a ground truth consisting of just a single top-level intention. In other words, the ground truth "sequence" $\langle u \rangle$ covering a given plan only has a single entry u_1. This limits the relevance of the dataset to real world scenarios where an agent may have multiple top-level intentions. Particularly, in robotic imitation learning, it is preferable if the robot can learn novel *sequences* of intentions from demonstrations, so that the useful sequences do not all have to be anticipated and hand-coded beforehand in an exhaustive knowledge base. In other words, intention inference should be a *constructive* process that leads to structured explanations, rather than a pattern classification task.

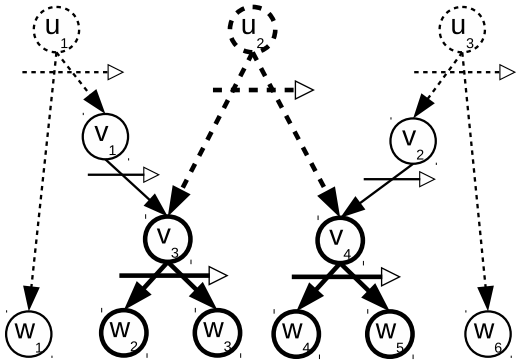

Figure 5.5: An example of a modified causal relation after stripping top-level causes from the knowledge base, as was done for the Monroe county corpus. Bold-faced nodes indicate a covering tree for a particular observed plan; dashed lines indicate causal relationships stripped from the knowledge base.

To provide more challenging problems along these lines, a modified variant of the Monroe domain was produced by stripping the top-most singleton intentions from the knowledge base. This transformation is illustrated in Fig. 5.5. The dashed nodes and edges depict top-level causes (u_1, u_2, and u_3) defined in the original knowledge base but removed in the modified one. The bold-faced nodes and edges represent a covering tree for a particular observed plan ($\langle w_2, w_3, w_4, w_5 \rangle$). Whereas the singleton u_2 was a top-level cover for this plan using the original causal relation, when using the modified causal relation, this plan can only be covered by a top-level *sequence* (namely, $\langle v_1, v_2 \rangle$). All performance assessments described above were repeated on this more challenging modified corpus.

However, since the original ground-truth covers have been removed from the causal relation, this methodology raises the question of what should be considered

the new "ground-truth" covers against which accuracy can be measured. Since full plan trees are provided for each plan in the original corpus, a default "ground-truth" could be the child sequence immediately below the original root. In Fig. 5.5, this would be the sequence $\langle v_2, v_3 \rangle$. Unfortunately, this new "ground-truth" is not guaranteed to be top-level in the modified causal relation. A counter-example is evident in Figure 5.5: $\langle v_2, v_3 \rangle$ has at least one higher-level cover ($\langle \langle v_1, v_2 \rangle \rangle$), and in larger examples, it may have many. In cases like this, which were found to be quite common, there is no single incontrovertible ground truth against which to measure accuracy. Consequently, the parsimony comparisons on the modified dataset were restricted to the plans where this issue did not arise.

5.2.4 Results

In the robotic domain, the parsimonious covers always included the ground truth (100% accuracy), except for the forest size-based criteria: FSN was only 45% accurate, and FSX was in fact 0% accurate.[9] On the other hand, several criteria were quite nonspecific, returning a large number of parsimonious covers in addition to the ground truth. Table 5.3 shows the specificity results on each plan in the

[9]This is because the knowledge base includes a top-level `move-to(object, destination)` intention, which accepts grippers as final destinations. So any top-level cover containing `move-to(object, destination)` remains valid when the occurrence is replaced with `move-to(object, gripper)`,`move-to(gripper destination`. The ground truths generally conformed to the former possibility, while the FSX covers conformed to the latter possibility, since there were more nodes in the covering forest.

dataset. Each column shows the total number of top-level covers retained after pruning by the respective criterion (abbreviations are as defined above). The 11 plans are ordered according to increasing length (i.e., the number of actions in the observed sequence). It was found that only MC and MP remain highly specific when the plan length and total number of top-level covers gets large. FSN and FSX are omitted because they were so inaccurate.

Table 5.3: Number of covers found by each criterion on each demonstration in the robotic domain.

Demo	TL	MC	IR	MD	XD	MP
Remove red drive (1)	4	2	4	2	4	1
Remove red drive (2)	4	2	4	2	4	1
Replace red with green (1)	8	2	8	4	8	1
Replace red with green (2)	8	2	8	4	8	1
Replace red with spare (1)	8	2	8	4	8	1
Replace red with spare (2)	8	2	8	4	8	1
Swap red with green (1)	2	1	2	2	2	1
Swap red with green (2)	2	1	2	2	2	1
Toy Blocks (IL)	256	1	256	256	256	1
Toy Blocks (AI)	1024	1	1024	1024	1024	1
Toy Blocks (UM)	8192	1	8192	8192	8192	1

For the original Monroe domain (without top-level intentions stripped from the knowledge base), accuracy of each criterion is shown in the second column of Table 5.4. In addition to the 162 timeouts, there was the small number of plans (42) in which the top-level covers found by EXPLAIN failed to include the ground-truth, even before filtering by any criteria. However, as mentioned earlier, there is evidence that the error might in fact be in the corpus itself rather than EXPLAIN (see Appendix A.4). Accuracy measurements were restricted to the remaining 4796 plans in the corpus.

Table 5.4: Accuracies on the original and modified corpus.

Criterion	Accuracy (original)	Accuracy (modified)
MC	4796 of 4796 (100.0%)	2859 of 2861 (99.9%)
IR	3397 of 3397 (100.0%)	2470 of 2470 (100.0%)
MD	4796 of 4796 (100.0%)	2856 of 2861 (99.8%)
XD	4796 of 4796 (100.0%)	2861 of 2861 (100.0%)
MP	4464 of 4796 (93.1%)	2724 of 2861 (95.2%)
FSN	1105 of 4796 (23.0%)	634 of 2861 (22.2%)
FSX	2087 of 4796 (43.5%)	2363 of 2861 (82.6%)

For most criteria, the set of all top-level covers found for a given plan can be filtered in linear time. The extremal value (minimum cardinality, maximum depth, etc.) can be found in one pass through the covers, and then the most parsimonious ones can be extracted in a second pass. However, filtering by irredundancy is more subtle. We performed irredundancy pruning based on the following proposition: Given two top-level covers t_1 and t_2, if t_1 is a sub-sequence of t_2, then t_2 is redundant. This idea can be used to filter out redundant top-level covers in quadratic time, which proved impractical in some cases. Using another time-out of 5 minutes, the irredundancy filter ran to completion on 3397 of these, so irredundancy metrics are computed relative to this subset.

We found that minimum cardinality, irredundancy, and depth-based criteria were all perfectly accurate: if the ground truth was included in the full set of top-level covers for a plan, it was retained after filtering by any of these criteria. The minimum parameters criterion was good but not perfect, while forest size-based criteria again performed poorly. These results are fairly unsurprising, given that every ground-truth is a singleton sequence, as described earlier.

More interesting is the comparison of specificity for each criteria, shown in

Figure 5.6. Most criteria could be grossly nonspecific in the worst case, retaining thousands of covers or more. This could be attributed to a combinatorial explosion in the number of possible parameterizations of the various intentions. As mentioned previously, the Monroe knowledge base is designed in such a way that many different parameterizations of a parent intention can cause the same child intentions. For example, recall that the intention `(clear-road-hazard ?from ?to)` causes the action `(call fema)`. There is a quadratic number of possible assignments of `?from` and `?to` to locations in Monroe County, which would all qualify as valid covers for `(call fema)`. When this occurs multiple times in a sequence, the number of valid covers at the next layer up grows exponentially. In contrast, as mentioned in the previous section, the minimum cardinality criterion was quite effective at mitigating this effect. In 3898 of the 4838 plans that did not time out (80.6%), it was maximally specific (the ground-truth interpretation was the sole minimum cardinality cover); in 4354 plans (90%), there were at most twelve minimum cardinality covers.

Finally, experimental results are reported on the modified Monroe dataset, in which the top-most intentions were stripped from the knowledge base. As described above, the issue arises in many plans that no unambiguous ground truth is available. Specifically, this was found to occur in 2139 of the 5000 modified plans, which were thus excluded from the accuracy comparison. Specificity was still measured on all 4842 modified plans that did not time out.

Accuracy on the remaining 2861 plans is shown in Table 5.4; as in the original corpus, all criteria except those based on forest size are highly accurate. As before, the IR filter timed out on some examples, so metrics for IR were only calculated on

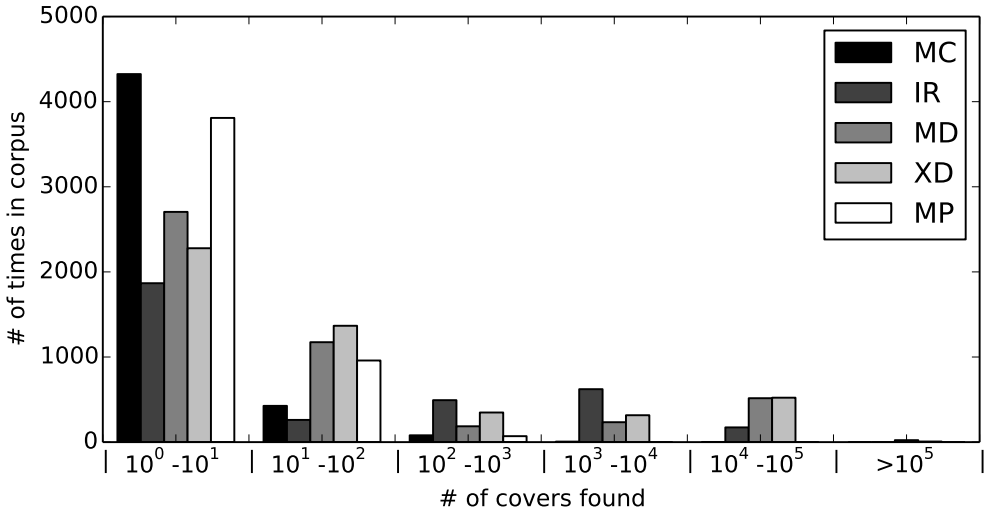

Figure 5.6: Distribution of the number of parsimonious covers found for plans in the original corpus.

the 2470 plans where it ran to completion.

Specificity histograms for each criteria on the modified corpus are shown in Figure 5.7. In contrast with the original corpus, it was found that like most other criteria, minimum cardinality become much less specific when inferring intention sequences, with counts over 100 covers for 42.3% of the plans, and some counts in the thousands in the worst cases, although this was still favorable compared with IR. On the other hand, MP proved quite specific on the modified corpus: it was maximally specific in 2756 of 4842 plans (56.9%), and in 4375 of the plans (90.4%), there were at most 16 MP covers found. As can be seen in Table 5.4, the improved specificity did come with some cost to accuracy.

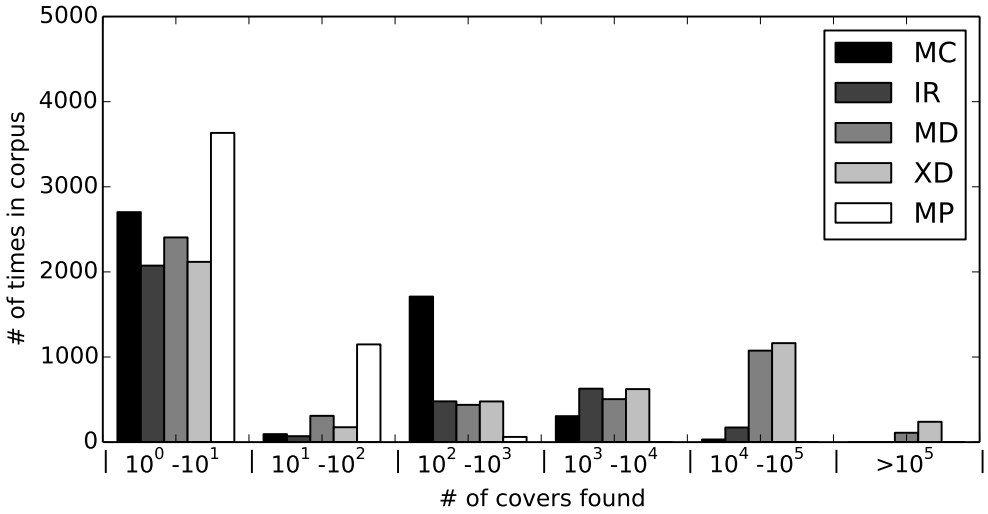

Figure 5.7: Distribution of the number of parsimonious covers found for plans in the modified corpus.

5.2.5 Discussion

The PCT approach to intention inference used in this work comes with strong formal guarantees of soundness, concreteness, and complexity characteristics (Chapter 4), but at the cost of assuming detailed background knowledge of the demonstrator, and perfect observability of their low-level actions. However, meeting the perfect observability requirement becomes entirely reasonable and realistic in practice once a virtual demonstration environment like SMILE is adopted [64]. Moreover, no more background knowledge is required than already needed by HTNs and other similar hierarchical planning formalisms.

This section has presented a systematic comparison of several relevant parsimony criteria, using multiple data sets for testing. In so doing, the original Monroe

domain has been augmented to test plan recognition when observed actions can only be explained by a *sequence* of top-level intentions/goals, rather than a single top-level intention. This situation may be more relevant for certain applications, such as robotic imitation learning.

The results suggest that in the context of intention inference, irredundancy is very accurate, but perhaps only because it is so nonspecific. When plans can always be explained by a single top-level intention/goal, minimum cardinality is equally accurate and qualitatively more specific. In the modified Monroe domain, minimum cardinality remains highly accurate, but becomes too nonspecific for practical use in about one fourth of the test cases. Since the new Minimum Parameters criterion proposed here remains highly specific even in these cases, it may sometimes be a preferable criterion, in spite of the 5% accuracy reduction. We can surmise that minimizing the number of distinct parameter values in a sequence can mitigate the combinatorial explosion described earlier, since covers with unrelated parameters in the constituent intentions will be filtered out. These results are in contrast with other work on abductive inference outside the context of intention inference, where irredundancy is theoretically the gold standard, although there have been many applications where it also produces too many covers to be useful. For example, minimum cardinality was also found to be a preferable criterion to irredundancy in a medical diagnosis application [130].

Even the best criteria identified in this work still suffer from some nonspecificity in the worst case. Finding better criteria for intention inference is an important future research direction. One possibility is to combine the promising criteria - for

example, prune first by MP, and then by MC. The incorporation of probabilistic methodology is another promising strategy. In fact, PCT has been extended to incorporate probability theory in the past, and a theoretical analysis led to conclusions about whether a given criterion would tend to include the most likely explanation [98]. For example, it was shown that the minimum cardinality covers would tend to include the most likely explanation as long as **(a)** the prior probabilities of the possible causes were small and roughly equal, and **(b)** the probabilities that any given u causes any given $\langle v \rangle$ are relatively large. One might argue that in intention inference, a large number of possible parameterizations makes the probability of any particular grounded intention rather low, satisfying **(a)**, and that if any particular grounded intention u causes one of at most a small handful of possible sub-intention sequences $\langle v \rangle$, the causal probability of each is relatively high, satisfying **(b)**. On the other hand, the probabilistic causal model previously developed for PCT only treated the special case where the causal relation is bipartite (i.e., all causal chains have length 1), and there are no ordering constraints, so it is unclear to what extent those results carry over to intention inference.

At any rate, evaluating probability-based criteria requires that the prior and causal probabilities are already known before the likelihood of any cover can be computed. Although the Monroe corpus does provide prior probabilities for top-level intentions *with ungrounded parameters*, the parameter distributions and causal probabilities appear difficult to ascertain, in this and possibly other planning domains. Fortunately, much work on HTN planning and plan recognition has produced methods for the *inductive* problem of learning causal relations, and estimating their prob-

ability distributions, on the basis of large training sets, in contrast to the *abductive* problem considered here, namely, constructing an explanation for a particular problem instance when the causal relation is already fully known. Future work should leverage these inductive methods in combination with the probabilistic causal model of PCT, suitably extended to handle causal chaining and ordering constraints.

Lastly, the results in this section may be sensitive to the particular HTN encoding, as authored by a human, of the given domains. Different encodings of the same domains might produce different results. Future work should include experiments that characterize and quantify this sensitivity.

Chapter 6: Causal explanation of planned actions

A major source of opacity in many autonomous systems is their inability to explain their actions to humans in an intuitive way. Lack of transparency in autonomous systems can foster mistrust by humans and limit user adoption. This issue has driven research interest in so-called "Explainable AI" (XAI) [36].

A promising strategy that has been used for XAI endows the autonomous system with a question answering ability, with which it can justify its actions when prompted by a human user. The question-answering ability is sometimes coupled with a graphical interface that includes visualizations of the autonomous system's operating environment. However, past approaches have relied on elaborate logic programs for representing knowledge or hand-coded databases of questions and domain-specific answering procedures (e.g., [33, 131, 137]).

A key element of CERIL is its use of *parsimony criteria* to identify good explanations for a demonstrator's actions. The results in Chapter 5 suggest that CERIL's emphasis on parsimonious causal reasoning may capture an important and general functional aspect of human cognition. On that basis, we might hypothesize that the same principles used to explain a *demonstrator's* actions could also be used by CERIL to explain its *own* actions to a human end user, thereby promoting

transparency.

The present chapter explores this hypothesis by extending CERIL to include a question-answering mechanism, based on the causal plan graphs (CPGs, detailed below) generated during its planning process. The CPGs model the causal relationships between CERIL's various intentions and goals that ultimately cause it to perform the actions that are executed. The extension proposed here uses CPGs to generate convincing and parsimonious explanations for what CERIL plans to do, in the form of causal chains. Compared with past approaches, this results in a simpler graph-based and domain-independent methodology for XAI. The utility of causal chains has been recognized before, in applications such as automated medical diagnosis [96, 102], but not in the context of robotic action justification. Inspired by these previous works, this chapter explores new ways to leverage causal chaining for the purposes of XAI.

Properly testing the hypothesis that parsimony principles used here actually produce compelling answers will ultimately require careful end user studies. However, the raw CPGs generated by CERIL on real-world examples are complex enough that there is significant information overload and an unwieldy number of valid explanations for any particular action. Before experiments with human participants can be performed, a non-trivial causal reasoning mechanism is needed that filters the possible explanations to a plausible, reasonably sized subset, and presents the information in a manageable way. This chapter presents such a mechanism and describes its operating principles. It also includes some initial empirical results to characterize the reduction in complexity in some real-world test cases.

6.1 CERIL's XAI Mechanism

Fig. 6.1 shows a typical screenshot of the target output for CERIL's XAI mechanism. CERIL has observed a demonstrator remove a hard drive next to a red LED and then insert a spare drive in its place. Based on that demonstration, CERIL has used its PCT algorithms to infer plausible intentions that could explain the observed actions. Finally, CERIL has planned new actions to carry out these intentions in a new situation, using its HTN planner. For simplicity, the actions and intentions in the hierarchy are collectively referred to as "aims."

The XAI interface shown in Fig. 6.1 allows users to navigate through all of CERIL's planned aims. To orient users, 3D visualizations of the expected environment immediately before and after the current aim are shown. It is assumed that users want a justification for the current aim, which is displayed as a question. Possible end goals that could account for the current aim can be selected from a drop-down menu. As shown in Fig. 6.1, an answer is provided in the form of a causal chain that was constructed by the underlying causal inference mechanism and explains how the current aim ultimately enables the currently selected end goal to be achieved. The following sections describe this justification mechanism in more detail.[1]

[1] A video screencast of the interface being used is available at `https://youtu.be/KpgLvCWwdNg` (for best viewing set quality to 720p).

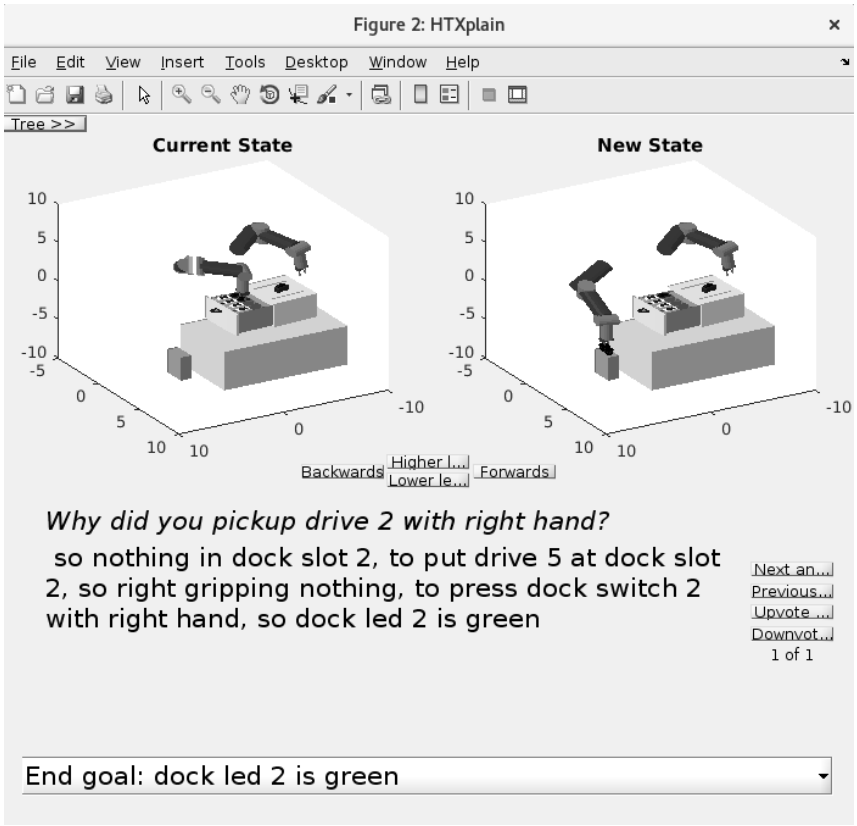

Figure 6.1: A typical screenshot of the XAI interface. In this example, CERIL has planned to imitate a demonstration in which a faulty drive ("drive 2") is replaced with a spare ("drive 5").

6.2 Causal Plan Graphs

CERIL's XAI mechanism is based on a domain-independent data structure proposed here called a "causal plan graph" (CPG). A CPG is a generic execution trace of a hierarchical planning algorithm, containing the trees of aims and sub-aims that ultimately produce the plan (i.e., low-level robot-executable action sequence) followed by the robot. It also contains the initial, intermediate, and final states of the environment that the robot expects before and after every action is performed. Each action and state are associated with a discrete "time-step" over the course of plan execution. The state at any given time-step is represented in the CPG as a list of first-order logic predicates. Each predicate represents a fact that the planner intends to make true at that point in time. Figs. 6.2 and 6.3 show examples of CPGs.

CPGs are similar to Goal Graphs [57], but they include higher-level aims in addition to the bottom level actions. They are also reminiscent of plan-space planners that use "causal links" between pairs of actions [124], although CPGs use a different notion of causal links as elaborated in the following. Most hierarchical planners, including Hierarchical Task Networks (HTNs), Hierarchical Goal Networks (HGNs), and Goal Task Networks (GTNs), construct a CPG (at least implicitly) over the course of their search procedure [5, 50, 116]. Implementations of these planning formalisms can be augmented in reasonably straightforward ways to return a CPG explicitly if they do not do so already.

More formally, a CPG is a graph with two types of nodes: *aims* and *predicates*.

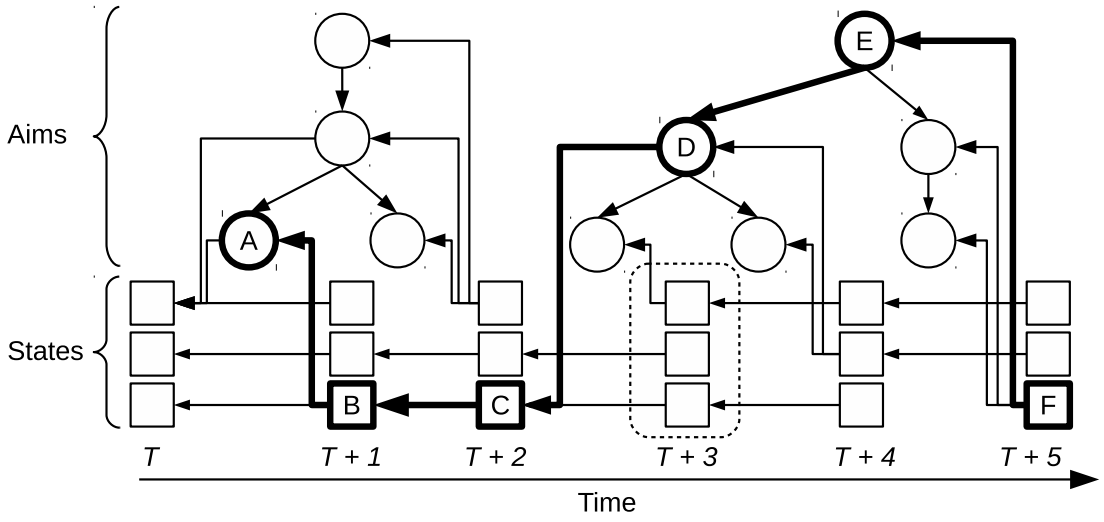

Figure 6.2: A generic CPG. Circular nodes are aims and square nodes are predicates (facts that CERIL intends to make true). Aims and sub-aims are grouped into trees; predicates are grouped into states at a given point in time (e.g., dashed rounded rectangle). Time proceeds from left to right. Although actions precede their postconditions, the *intention* to make those postconditions true is what causes the *intention* to perform those actions. Therefore arrows denoting causal intention relationships point backwards in time. A causal chain is shown in bold. The labels A through F are referenced in the text.

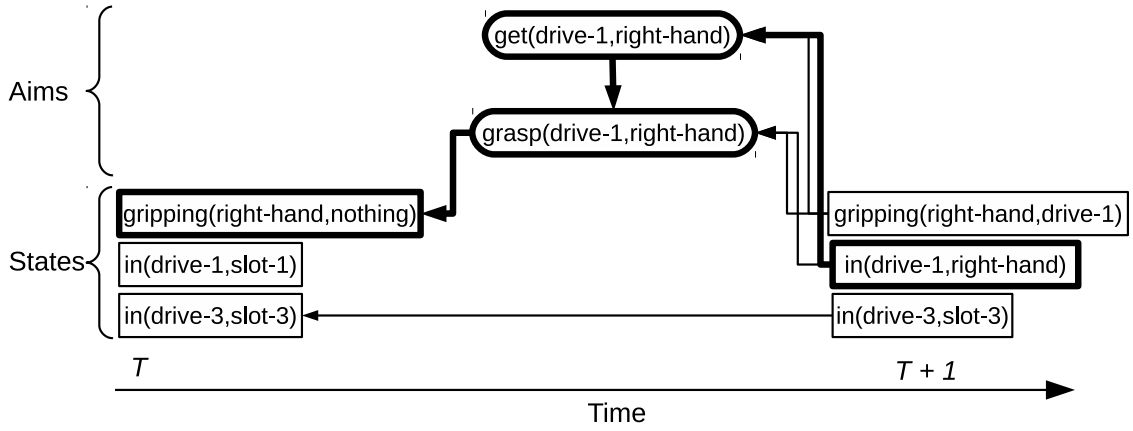

Figure 6.3: Another more concrete example of a portion of a CPG in the dock maintenance domain. The intention for `drive-1` to be in the `right-hand` causes the `get-to` aim, which in turn causes the `grasp` sub-action. The intention to grasp between times T and $T+1$ causes the ancillary goal that the `right-hand` be empty at time T, so that it is available for the grasp.

Aims represent the intentions, sub-intentions, and ultimately low-level actions that were queued by the planning algorithm while it searched for a viable plan. The generic term "aim" is meant to include task methods in the case of HTNs, goal methods in the case of HGNs, or a heterogeneous mix of both in the case of GTNs.[2] The sequence of top-level aims constitutes the inputs provided to the planning algorithm; the sequence of bottom-level aims constitutes the plan that is output by the algorithm. In CERIL, the top-level aims were previously inferred from a demonstration.

Predicates represent the facts expected to be true in a particular state over the course of the plan. Predicates can become false (deleted), become true (added),

[2]For the technical distinction between task and goal methods, see Chapter 2 or [5, 50, 116].

or remain true from one state to the next depending on which action is performed. The predicates in the final state are referred to as *terminals*. In Fig. 6.2, nodes A, D, and E are aims, and nodes B, C, and F are predicates. Node F is a terminal.

It is important to note that predicates are viewed as facts that the planner *intends* to make true. Although an action may cause a predicate to become true, the *intention* to make the predicate true is what causes the *intention* to perform the action. The latter is what gets captured by the CPG. As a result, causal links appear to flow backwards in time; this is somewhat counterintuitive, but not paradoxical when nodes represent *intentions*. In keeping with these semantics, a CPG has four types of causal links, as follows. The first type was used previously by CERIL to interpret a demonstration and infer the top-level intentions. The other three are new and extend CERIL's causal reasoning abilities.

- **Aim links** ($aim \rightarrow aim$). Each parent aim has a causal link to its immediate child aims, since the system's intention to carry out the parent aim causes its intention to carry out the child aims. The link from E to D in Fig. 6.2 is an example of an aim link.

- **Precondition links** ($aim \rightarrow predicate$). Each aim has a link to its *preconditions*, which are the predicates that are required to be true in the state immediately prior in order for the aim to be an admissible option for the planner. These links indicate that the system's intention to carry out the current aim caused its intention to make the relevant preconditions true. The link from D to C in Fig. 6.2 is an example of a precondition link.

- **Postcondition links** (*predicate* → *aim*). Each aim has a link **from** its *post-conditions*, which are the predicates that become true in the state immediately after the aim is carried out. Normally we think of an aim as causing its post-conditions, but for the purposes of action justification, the causal relation runs in reverse: the *intent* to make those postconditions true causes the *intent* to carry out the preceding aims. The links from B to A and from F to E in Fig. 6.2 are examples of postcondition links.

- **Persistence links** (*predicate* → *predicate*). Each predicate that *persists* during a bottom-level aim (i.e., doesn't change truth value) retains a link from its corresponding node immediately after the aim to its corresponding node immediately prior. This captures the idea that if the system intends for a predicate p to be true at time $T + 1$, and p is not made true by the action between time T and $T + 1$, then the system should also intend for p to already be true at time T. The link from C to B in Fig. 6.2 is an example of a persistence link.

Postcondition and persistence links do not need to be explicitly included in the CPG returned by the planning algorithm; they can be added as a post-processing step by comparing the sets of predicates at time steps T and $T+1$, which we can denote $P^{(T)}$ and $P^{(T+1)}$. Post-conditions can be calculated from the set difference $P^{(T+1)} - P^{(T)}$. Persistence links can be calculated from the set intersection $P^{(T+1)} \cap P^{(T)}$.

Persistence links are related to the frame problem [109], but since they can be calculated *a posteriori* with a set intersection, CPGs are agnostic to how the frame

problem is solved by any particular domain-specific knowledge base. Postconditions are explicitly modeled in goal-centric planners such as HGNs and GTNs, but often implicit in task-centric HTNs. So when HTNs are used (as in CERIL's current implementation), postconditions must also be determined with this post-processing step.

6.3 Justifying Actions with Causal Chains

Consider the CPG shown in Fig. 6.2 and suppose that a human user wanted the autonomous system to explain why it planned the aim labeled A. There are several ways the CPG might be used to accomplish this. One possibility is to show the entire CPG to the user as in Fig. 6.2 and let them inspect on their own to make sense of the relevant causal relationships. However, in practice it was found that typical CPGs can be quite large, resulting in information overload that is barely intelligible even to the designers of the autonomous control system.

Another possibility is for the system to covertly traverse the CPG in the background and locate the terminals in the final state that could have eventually caused A. One such predicate is the node labeled F. The bold links show a causal chain connecting F to A (in general, there may be more than one). To minimize information overload, the system could omit the causal chain and simply provide F as a justification. However, this may be unsatisfactory to a human user: it indicates *what* the system wanted to accomplish, but not *why* in particular A was an important part of the plan, as opposed to some other aim instead of A. For example, in Fig.

6.1, the end goal "dock led 2 is green" by itself is not a particularly illuminating answer to the question "why did you pickup drive 2 with right hand?" In contrast, the intermediate causal chain displayed to the user explains how picking up drive 2 out of the slot was a necessary step in order to make room for the spare drive and ultimately restore the LED to a non-faulty state.

On the other hand, there may be several terminal predicates that indirectly cause any particular action, and there may be many causal chains from each such terminal to that particular action, so some filtering is necessary to extract the most useful causal chains and discard the rest. Some filtering can also be applied to individual causal chains to suppress uninformative or redundant nodes. For example, in Fig. 6.2, if nodes E and F had redundant human-readable representations, the system could keep E but exclude F when converting the causal chain to a natural language answer (e.g., see step (5) below).

Empirical data from human subjects is needed to determine what makes a causal chain more or less useful. A basic experiment would involve humans ranking multiple causal chains, but to conduct this experiment in the first place, the plethora of causal chains must be reduced to a smaller, intelligible subset. We can accomplish this reduction by applying parsimony criteria to filter the set of causal chains. As a starting point, causal chain length can serve as the primary parsimony criterion for extracting a minimal intelligible subset. The shortest causal chains between any particular aim and terminal are considered most human-friendly and displayed to the user. All other causal chains are discarded. To compute all pairwise shortest paths between aims and terminals, a Floyd-Warshall-style dynamic programming

algorithm is used [42]. Some care is taken in heuristically assigning costs to each link before computing causal chain "lengths" (i.e., net costs). Pre- and post-processing stages are also used to further promote the principle of parsimonious explanation. Specifically, the parsimony filtering involves the following steps:

1. Prune the CPG by removing the aim sub-trees where sub-symbolic changes occur (such as robot arm motions), but no changes occur at the predicate level. Sub-symbolic changes would make the explanations less parsimonious without adding human-friendly information.

2. Use set operations as described earlier to add persistence links and postcondition links to the CPG if they are not already present. The additional links provide more opportunity for parsimonious (i.e. shorter) causal chains.

3. Assign a cost of 1 to each link, with the following exceptions:

   - Persistence links have a cost of 0. This is because they are redundant and can be collapsed into a single entry of a human-readable answer without reducing its information content or increasing its complexity.

   - Precondition links from a predicate p to an aim a have a cost of $1 - (1/2)^{d(a)}$, where $d(a)$ is the depth of aim a below the top level. All things equal this favors causal chains that pass through higher-level aims in the CPG. This is essentially a heuristic which produces reasonable results on our test domain, but may not generalize well to other domains. A more interesting possibility is that end user feedback could be combined with

reinforcement learning to adjust link costs automatically.

Using these costs, compute all-pairs shortest paths.

4. At each aim in the CPG, enumerate each connected terminal and separately cache the shortest causal chains from each one. Caching the chains makes the user interface more responsive, and grouping the shortest chains by terminal facilitates the "end goal" drop-down menu with which the user can explore different candidate explanations. Keeping separate explanations for separate terminals also fosters parsimony because it avoids trying to explain too much at once.

5. Convert each shortest causal chain to a human-readable format, using a per-node "toString" function supplied by a domain author, with the following adjustments:

    - When the chain includes contiguous repetitions of a persistent predicate, only include the first occurrence of the predicate in the human-readable output.

    - When an aim node is followed by a *namesake* postcondition, omit the postcondition from the human-readable output. For example, the aim "put drive 1 at slot 1" has postconditions "drive 1 at slot 1" and "right gripping nothing". The first postcondition is considered a *namesake* and is essentially redundant in a human-readable answer, but the second is not. So only the first would be omitted from a human-readable answer.

Since this redundancy is generally not automatically detectable from the machine representation, it is screened by a "namesake function" that must also be supplied by the domain author.

Clearly, both adjustments promote parsimony without discarding any relevant information.

6.4 Graphical User Interface

Similarly to some other XAI approaches, a graphical user interface (GUI) is employed to facilitate end user interaction with the system. The GUI has been implemented as a MATLAB application, pictured in Fig. 6.4. In this example, the robot was shown a demonstration where a cartridge next to a red LED (`drive-1`) had its position swapped with a cartridge next to a green led (`drive-3`). The robot planned an action sequence to carry out this skill in a new situation, and the resulting CPG was loaded into the GUI so that the planned actions could be justified to an end user. In this snapshot, the robot is justifying why at one point it moves `drive-1` to the right gripper. The answer explains how this was necessary to make `drive-1`'s old slot available for `drive-3` to be moved there, which in turn makes `drive-3`'s old slot available for `drive-1`, enabling the swap.

The GUI is initialized with the CPG and from then on maintains a notion of the "current aim" that is being inspected and justified. The system assumes the user is asking why the current aim was planned (A). The user can qualify their question relative to one of the causally connected terminals in a drop-down menu (B). The

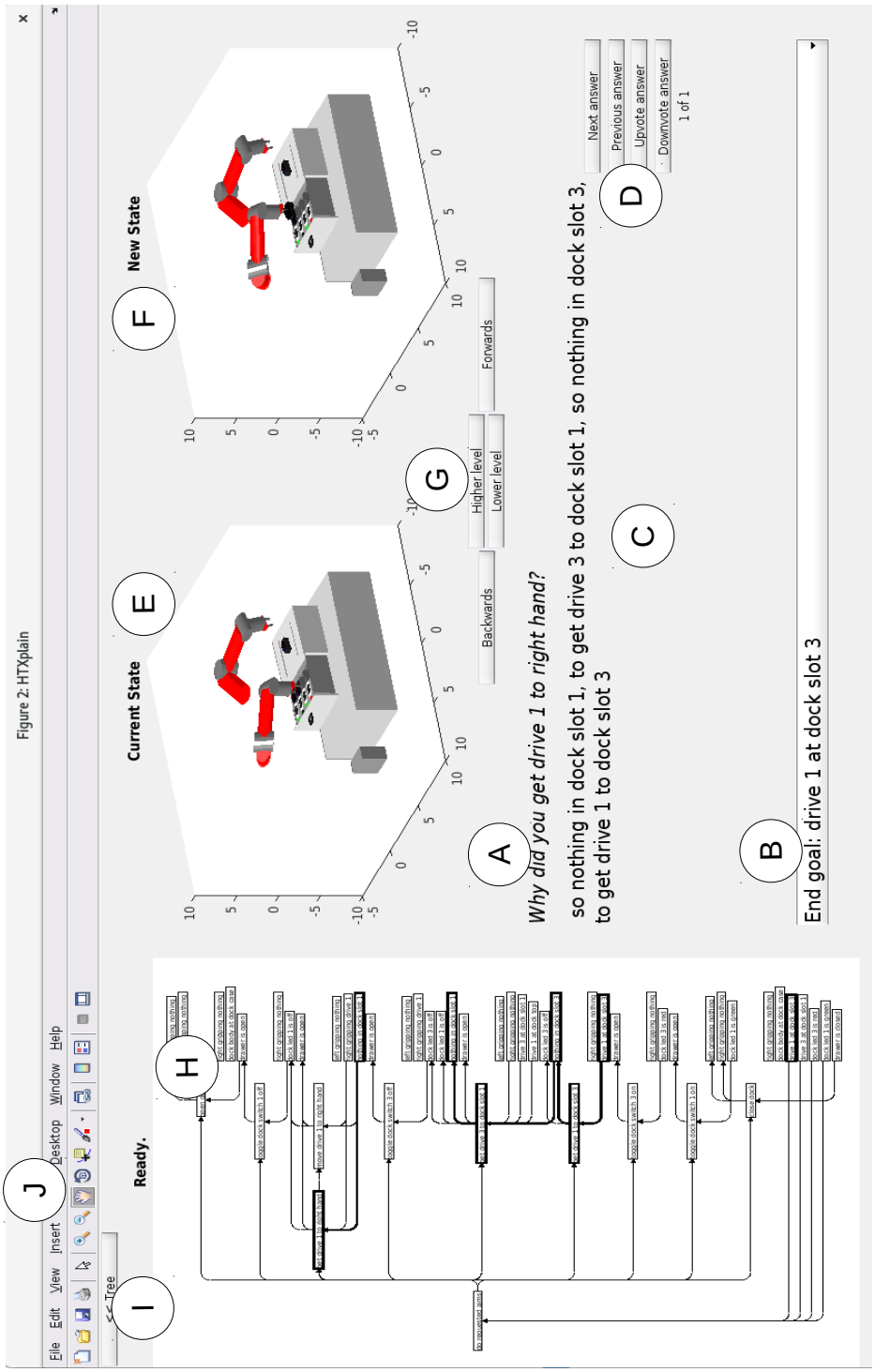

Figure 6.4: Screenshot of the action justification GUI. Labels (A) through (J) are described in the text.

system answers the user's question with one of the causal chains that connects the current aim to the currently selected terminal (C). When there is more than one shortest chain connecting the current aim and terminal, the user can click the control buttons on the right (D) to cycle through the answers and up- or down-vote each one. This voting system serves as a hook that can be used later to inform more sophisticated criteria for filtering the causal chains.

The current aim is visualized by showing 3D views of the states immediately before and after (E and F) the current aim is carried out. To change the current aim, the user can click the navigation control buttons (G) to step forwards or backwards in time, or shift to higher or lower levels of the CPG.

For advanced users, the full CPG can be revealed (H) with a toggle button in the upper left (I). The CPG can be zoomed and panned with the built-in MATLAB figure controls (J). Each aim node can be clicked to collapse/expand the child aims and also select it as the current aim to be justified. In response all visuals and text on the right of the GUI update accordingly. Likewise, user interactions on the right of the GUI affect where the CPG on the left is centered, which aims are expanded, and which chain is bold. For the most effective use of space, the CPG layout in the GUI has time advancing top to bottom, with high-level aims on the left and low-level actions/states on the right, in contrast to Fig. 6.2. This viewing option can be useful for detailed inspection of the CPG, but is also clearly susceptible to information overload, motivating the use of the simpler and more human-friendly elements on the right of the GUI.

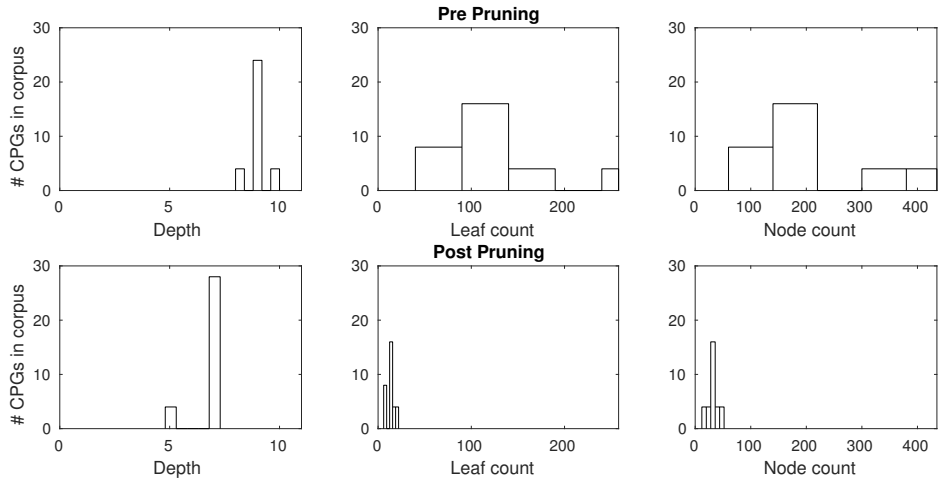

Figure 6.5: Histograms showing the depths, leaf counts, and node counts across the 32 CPGs in the test corpus, before and after pruning.

6.5 Initial Experimental Results

Initial empirical experiments quantified the complexity of the CPGs in real-world examples and measured the reduction in causal chain counts after filtering for shortest paths. These experiments used the dock maintenance test corpus, which consists of 32 imitation trials, each with its own CPG.

The first experiment quantified the reduction in CPG complexity resulting from pruning sub-symbolic aims (e.g., robot arm motions) as described in step (1) above. Reduction in complexity was measured by the depth of the aim trees, the number of leaf aims, and the total number of all aim nodes. Fig. 6.5 shows the results before and after pruning. Even after pruning, the CPGs remained fairly large and unwieldy from the perspective of an end user, with 16 to 50 aim nodes.

The next experiment used the pruned CPGs to compare the causal chain

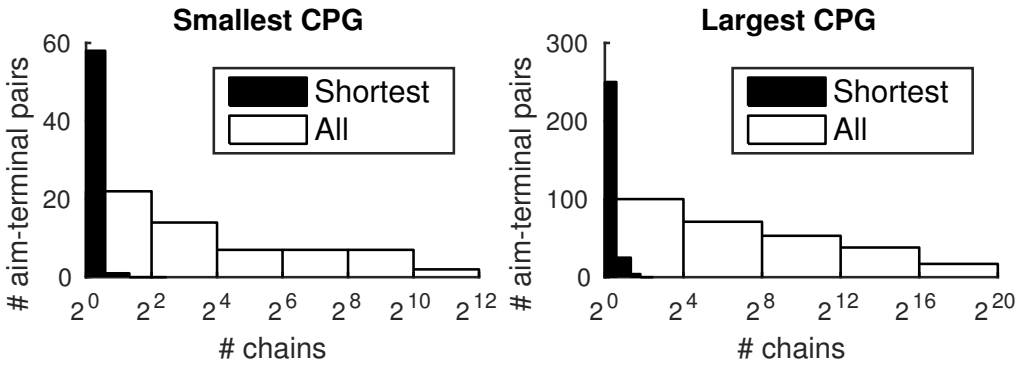

Figure 6.6: Histograms of causal chain counts before (white) and after (black) shortest path filtering. Distributions of causal chain counts are computed across all ⟨aim, terminal⟩ pairs in the CPGs.

counts at each aim node both before and after filtering for shortest paths. Since the CPGs are typically highly connected, there can be combinatorially many distinct paths between any two nodes. Therefore restricting to the shortest paths is very important for reducing the number of justifications to a reasonable amount. Fig. 6.6 verifies this empirically on two CPGs: one of the smallest and one of the largest in the dock corpus. When non-shortest paths are included, the number of causal chains is easily 1000 or more in the worst case.[3] Whereas the number of shortest paths to any aim is always between 1 and 3.[4] This highlights the importance of using

[3]The total *number* of paths to a node can be calculated efficiently without explicitly computing the paths themselves, which quickly becomes intractable on modestly sized CPGs. The path count to any given node is the sum of path counts to each of its incoming neighbors, which is a straightforward addition to the standard Floyd-Warshall approach.

[4]When there is more than one shortest chain between a particular aim-terminal pair, additional criteria will be needed to choose between them. Currently the chains are listed in arbitrary order, but as mentioned above, more sophisticated criteria can be developed using end user feedback.

an efficient all shortest paths algorithm like Floyd-Warshall, which can compute the shortest paths without enumerating all of them. It also parallels the previous results concerning parsimonious explanation of a *demonstrator's* actions in Chapter 5. This supports the hypothesis that parsimony can serve as a general unifying principle for cause-effect reasoning in AI.

6.6 Discussion

This chapter has presented a new method for XAI based on parsimonious causal inference. The method leverages causal plan graphs, which are domain-independent data structures that capture causal relationships between an autonomous system's intentions, goals, and actions. A procedure based on all-pairs shortest paths can extract a small, intelligible subset of parsimonious causal chains that justify any given aim invoked during a hierarchical planning process. Manual inspection verifies that the resulting justifications are reasonably compelling and intuitive for a human end user. Quantitative empirical experiments also verify that in at least one domain, the justification procedure successfully filters an otherwise combinatorially large set of answers to a small intelligible subset. These results suggest that parsimony holds promise as a useful, general principle for causally-driven XAI.

Future work should check these results on other problem domains and with other hierarchical planners. Future work should also conduct end user studies with human participants who use and evaluate this XAI mechanism. The contribution of the present chapter is to provide a platform that makes those experiments possible,

paving the way for an improved understanding of human XAI users that can inform the next generation of transparent XAI systems.

Chapter 7: Locating Fixed Points in Neural Networks

An important future research direction to improve the performance and adaptability of the robotic imitation learner used in this work is to re-implement the core causal reasoning components used in CERIL with state-of-the-art neural computation. This work is already in progress at the sensorimotor level, where the inverse kinematics are computed with brain-inspired methods [47]. On the other hand, there are several opportunities for neural computation in the cognitive-level causal inference, which is currently symbolic. For example, deep learning may be a promising strategy for learning new causal knowledge from experience rather than a human-authored domain, and cognitive architectures like GALIS may be a promising strategy for implementing the causal inference algorithms [122]. However, deep networks are sometimes unpredictable and almost always opaque, hindering transparency and trustworthiness. GALIS also suffers from unreliability and unpredictability when the number of attractors required to encode desired instruction sequences approaches or exceeds the capacity of the underlying networks. This leads to trained networks that possess undesirable spurious attractors or lack desired attractors from the training data. It is often very difficult at present to analyze and understand the behavior of complex dynamical systems of this sort.

Consequently, an important first step towards a neural, P&T implementation of CERIL is to improve the transparency and reliability of neural networks. In particular, it would be especially useful to have a mathematically well-understood method that can verify whether desired attractor dynamics are present and undesired dynamics are absent in a trained network. To this end, I developed a new method, described in the following, for systematically locating fixed points in recurrent neural networks [70, 71]. This chapter presents the mathematical analysis of the approach and describes computer experiments that show that it consistently locates many fixed points in many networks with arbitrary sizes and unconstrained connection weights. Comparison with a traditional method shows that this strategy is competitive and complementary, often finding larger and distinct sets of fixed points. This work provides a theoretical basis for further analysis and suggests next steps for developing the method into a more powerful solver.

7.1 Fixed Points of Neural Attractor Dynamics

One of the most basic properties of any dynamical system is the location of its fixed points. However, in non-linear, high-dimensional dynamical systems, such as recurrent neural networks (RNNs), ascertaining this information can be very challenging. Fixed points of RNNs can represent many things, including stored content-addressable memories [58], solutions to combinatorial optimization problems such as the Traveling Salesperson Problem [59], and unstable waypoints of non-fixed dynamics [103]. Consequently, a global fixed point solver has the potential to

improve our understanding and engineering of RNNs in all of these use cases. In addition, such a solver could provide information useful in comparing the effects of different learning rules, and might reveal new strategies for solving other non-linear systems of equations in general.

Attractive and unstable fixed points are both highly relevant to many neurocomputational phenomena, ranging from low-level motor control and tool use (e.g., [2, 136]) to high-level cognitive functions such as problem solving and decision making (e.g., [103, 122, 132]). Fixed points are also related to waypoints along non-fixed attractors under slight perturbations to the network weights (see Fig. 7.1).

As mentioned in Chapter 2, despite some remarkable past work, there remains no efficient procedure that precisely locates every fixed point of RNNs with arbitrary connectivity. Since global fixed point location remains an important open problem, it is worth developing new solvers that, if not global themselves, are complementary to existing solvers and provide new perspective on the global problem. This chapter presents a novel strategy for locating fixed points in a broad class of dynamical systems, including RNNs with arbitrary size and no symmetry constraint on the weights. First the approach is presented in general terms, and then it is applied to RNNs. Next, it is shown empirically that the method consistently locates many fixed points in many randomly sampled networks. Comparison with an existing method shows that this strategy is both competitive and complementary, often locating different and larger sets of fixed points. Finally, future directions for improving the solver's performance are discussed.

While the approach presented here is *not* a global solver, it does present a new

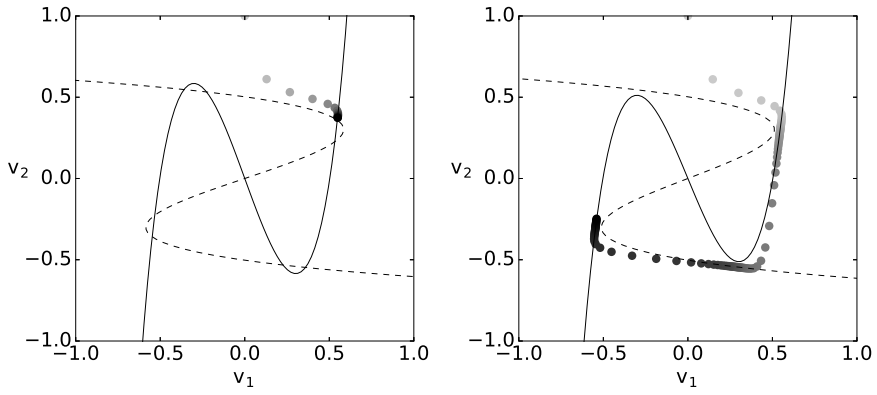

Figure 7.1: Phase spaces, trajectories, and nullclines for a pair of two-neuron networks with nearly identical connection weights (the precise network model used here is defined in Sect. 7.3.1). Each coordinate axis corresponds to the activity of each neural unit, denoted v_1 and v_2. The nullclines where the network update leaves v_1 fixed or v_2 fixed are indicated by solid or dashed curves, respectively. The intersections of the nullclines are fixed points. Solid gray circles show successive points along an arbitrarily chosen trajectory, with lighter shades further back in time. Under a small perturbation to the connection weights, the fixed points of the original system (left) degenerate into waypoints along a non-fixed attractor in the perturbed system (right).

strategy towards the eventual goal of developing a global solver, and derives some preliminary theoretical groundwork. Whether or not this strategy can ultimately succeed depends on several theoretical questions that as yet remain open. In addition to proposing and evaluating the method, one of the main contributions here is to pose and discuss these open questions and suggest future directions for improving the solver's performance.

7.2 Theoretical Groundwork

7.2.1 Notation

In this chapter, $\mathbb{N}$ denotes the natural numbers and $\mathbb{R}$ denotes the reals. $N \in \mathbb{N}$ denotes the dimensionality of a dynamical system and $\mathbf{0} \in \mathbb{R}^N$ denotes a column vector containing all zeros. For any matrix M, $M_{i,j}$ denotes the $(i,j)^{th}$ entry, and $M_{i,:}$ denotes the i^{th} row. The D prefix denotes multivariate differentiation. For example, if $f\colon \mathbb{R}^N \to \mathbb{R}^N$ is a differentiable function, then Df is its Jacobian, i.e., $(Df(v))_{i,j} = df_i(v)/dv_j$ for $v \in \mathbb{R}^N$. Commas and semicolons inside square brackets denote horizontal or vertical concatenation, respectively, of matrices and vectors. For example, $[A, B]$ is the horizontal concatenation of A and B. The vector and induced matrix 2-norm are denoted $||\cdot||_2$.

7.2.2 Directional Fibers

The fixed point solver proposed here is based on mathematical objects introduced in this work called *directional fibers*. To my knowledge, the concept of directional fibers is new, and their utility for locating fixed points has not been recognized previously, either in RNNs or other dynamical systems. Whereas a standard mathematical fiber is the inverse image of some constant *point*, a directional fiber is the inverse image of some constant *direction*. This concept is well defined for any function whose codomain is a vector space. Directional fibers have several mathematical properties useful for locating fixed points, described below. This section

presents directional fibers in general terms; Sect. 7.3 shows how the method can be applied to RNNs.

Definition 1. *Given* $f\colon \mathbb{R}^N \to \mathbb{R}^N$, *and* $c \in \mathbb{R}^N - \{\mathbf{0}\}$, *the* ***directional fiber*** *of* c *under* f, *denoted by* $\gamma^{(c)}$, *is defined as:*

$$\gamma^{(c)} \stackrel{\text{def}}{=} \{v \in \mathbb{R}^N \,:\, f(v) \textit{ is parallel to } c\}. \tag{7.1}$$

Consider a discrete-time dynamical system with states in $\mathbb{R}^N$, and state transitions given by

$$\Delta v = f(v), \tag{7.2}$$

where $v \in \mathbb{R}^N$ is the current state, and $f\colon \mathbb{R}^N \to \mathbb{R}^N$ is a function that specifies Δv, the change in state after one dynamical update. Given a direction vector c, the directional fiber $\gamma^{(c)}$ is the set of all states where the dynamical update moves in the direction of $\pm c$. Fig. 7.2 shows an example.[1]

If $f(v)$ is parallel to c, then $f(v) = \alpha c$ for some $\alpha \in \mathbb{R}$, and it is convenient to make α explicit. Let $F^{(c)} : \mathbb{R}^N \times \mathbb{R} \to \mathbb{R}^N$ be the function defined as:

$$F^{(c)}(v, \alpha) \stackrel{\text{def}}{=} f(v) - \alpha c. \tag{7.3}$$

Then the directional fiber is equivalent to the following set:

$$\Gamma^{(c)} \stackrel{\text{def}}{=} \{(v, \alpha) \in \mathbb{R}^N \times \mathbb{R} \,:\, F^{(c)}(v, \alpha) = \mathbf{0}\}. \tag{7.4}$$

[1]Directional fibers may also be understood in relation to nullclines. If a point updates in the direction of c, then it is stationary along any direction orthogonal to c. So directional fibers can be thought of as intersections of $N - 1$ nullclines in a rotated coordinate system.

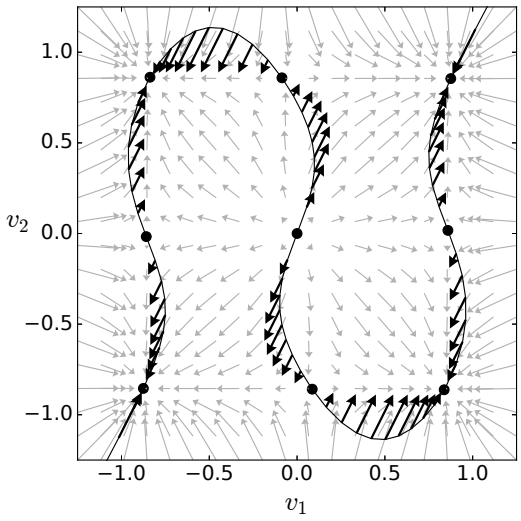

Figure 7.2: The phase space for an arbitrary two-neuron network, with an arbitrary directional fiber superimposed. The network model is defined in Sect. 7.3.1. Light gray arrows indicate $f(v)$ at regularly spaced points $v \in \mathbb{R}^2$. Solid black circles indicate fixed points. The black curve is a directional fiber: the set of all v that update along the same constant direction $\pm c$. The direction $\pm c$ is emphasized by additional arrows showing $f(v)$ at regularly spaced points along the fiber, colored black and scaled by a factor of 2.

It will sometimes be convenient to write $(v, \alpha) \in \mathbb{R}^N \times \mathbb{R}$ as a point $x \in \mathbb{R}^{N+1}$.

If f is differentiable and Df satisfies certain rank conditions, detailed below, it turns out that a typical directional fiber is a one-dimensional manifold containing every fixed point. The practical significance of this property is that a directional fiber can be *numerically traversed* to locate fixed points. More formally, we have the following definition and propositions. The main mathematical tools used in the proofs are Sard's Theorem [111] and the Inverse Function Theorem (IFT) [74].

Differentiability of f (and hence of $F^{(c)}$, for every c) is assumed throughout.

Definition 2. *A direction vector c is **regular** if $DF^{(c)}$ is full rank at every $x \in \Gamma^{(c)}$. Otherwise, c is **critical**.*

Proposition 1. *If Df is full rank at every fixed point, then the set of critical c has Lebesgue measure 0 in $\mathbb{R}^N$.*

Proof of Proposition 1. Consider the function $h : \mathbb{R}^N \times \mathbb{R} \to \mathbb{R}^N$ defined by $h(v, \beta) = \beta f(v)$. Then $Dh = [\beta Df, f]$. Let $\mathcal{H} = \{h(v, \beta) : \text{rank}(Dh(v, \beta)) < N\}$. By Sard's Theorem, $\mathcal{H}$ has Lebesgue measure 0 in $\mathbb{R}^N$.

Let c be any critical direction. Then by definition, there is some $(v, \alpha) \in \Gamma^{(c)}$ such that $\text{rank}(DF^{(c)}(v, \alpha)) < N$. Since $DF^{(c)} = [Df, -c]$, this implies that $\text{rank}(Df(v)) < N$ as well. If α were 0, then $f(v)$ would be $\mathbf{0}$, since $F^{(c)}(v, \alpha) = f(v) - \alpha c = \mathbf{0}$ for all $(v, \alpha) \in \Gamma^{(c)}$. But then Df would be less than full rank at a fixed point, contradicting the antecedent of the proposition. So, given that $\alpha \neq 0$, we can rearrange $f(v) - \alpha c = \mathbf{0}$ to write $c = f(v)/\alpha = h(v, 1/\alpha)$. So $DF^{(c)}(v, \alpha) = [Df(v), -f(v)/\alpha]$, which has the same rank as any matrix obtained by non-zero rescaling of its columns, including $Dh(v, 1/\alpha) = [(1/\alpha)Df(v), f(v)]$. Therefore $Dh(v, 1/\alpha)$ has rank less than N, and so $c = h(v, 1/\alpha)$ is in $\mathcal{H}$. Hence the set of critical directions is a subset of a zero measure set, and so also zero measure. □

Proposition 2. *For any regular c, $\Gamma^{(c)}$ is a one-dimensional manifold.*

Proof of Proposition 2. Given that c is regular, $DF^{(c)}(x^{(0)})$ is full rank at any $x^{(0)} \in \Gamma^{(c)}$, and since $F^{(c)}$ maps $\mathbb{R}^{N+1}$ to $\mathbb{R}^N$, the full rank $DF^{(c)}(x^{(0)})$ must have a one-

dimensional null space. Let z be a unit vector spanning that null space and define a function $G : \mathbb{R}^{N+1} \to \mathbb{R}^{N+1}$ by

$$G(x) = [F^{(c)}(x); z^T(x - x^{(0)})].$$

Since z spans the null space of $DF^{(c)}(x^{(0)})$ and has unit length, the Jacobian

$$DG(x^{(0)}) = [DF^{(c)}(x^{(0)}); z^T]$$

is invertible and z is also the $(N+1)^{th}$ column of $(DG(x^{(0)}))^{-1}$. By the IFT, G is a homeomorphism between a neighborhood $\mathcal{U}$ of $x^{(0)}$ and a neighborhood $\mathcal{V}$ of $G(x^{(0)})$. In particular, for $x \in \mathcal{U} \cap \Gamma^{(c)}$, $F^{(c)}(x)$ vanishes and $G(x) \in \mathcal{V}$ has the form $[\mathbf{0}; z^T(x - x^{(0)})]$, and hence $\mathcal{U} \cap \Gamma^{(c)}$ is homeomorphic to $\mathbb{R}$. Therefore $\Gamma^{(c)}$ is locally one-dimensional around $x^{(0)}$, and effectively parameterized by the $(N+1)^{th}$ coordinate of $G(x)$. The tangent to $\Gamma^{(c)}$ at $x^{(0)}$ is the derivative of G^{-1} with respect to this parameter, which by the IFT is precisely the $(N+1)^{th}$ column of $(DG(x^{(0)}))^{-1}$, namely z. Given that c is regular, $DF^{(c)}$ is full rank at every such $x^{(0)} \in \Gamma^{(c)}$, and so $\Gamma^{(c)}$ is globally a one-dimensional manifold. □

Proposition 3. *Any directional fiber contains every fixed point: Fixed points occur precisely when $\alpha = 0$.*

Proof of Proposition 3. Given any c, and any fixed point v, we have $F^{(c)}(v, 0) = f(v) - 0c = \mathbf{0} - \mathbf{0} = \mathbf{0}$, so $(v, 0) \in \Gamma^{(c)}$. □

In summary, on the assumption that Df is full rank at every fixed point, almost any directional fiber can be numerically traversed to locate fixed points. Given that

this assumption was never falsified by the empirical experiments in Section 7.4, one might reasonably conjecture that this assumption holds for almost every RNN in the family studied here, with respect to some reasonably defined measure. This conjecture remains an open question, although recent results may be relevant [119]. As a caveat, even if the set of RNNs where this assumption fails has measure zero, it still includes some important special cases such as networks with line attractors, which necessarily contain fixed points where Df is not full rank [14].

Although a regular directional fiber is a one-dimensional manifold containing every fixed point, it is not necessarily *path connected*. Therefore, whether or not *every* fixed point can be located by traversing a *single* fiber is an open question and depends on the method for choosing c. It is entirely possible that sometimes *no* choice of c will result in a fully connected fiber, and determining when this occurs may be intractable or undecidable. For the RNNs studied here, these questions remain open, but Section 7.3.3 elaborates on the issue and Section 7.4.3 provides relevant experimental results.

7.2.3 A Fiber-Based Fixed Point Solver

This section presents the method for locating fixed points. The key idea is to choose a suitable directional fiber and numerically traverse it, adding fixed points to a running list as they are encountered. Choosing a suitable fiber, deriving a reliable numerical update scheme, and identifying starting and stopping conditions are all system-dependent problems, covered in Section 7.3 for the case of RNNs.

The traversal process is codified in algorithm TRAVERSE (Fig. 7.3), which operates as follows. Line 1 initializes the traversal at a valid starting point $(v, \alpha) \in \Gamma^{(c)}$. Line 2 initializes V^*, a running list that starts out empty and accumulates fixed points over the course of traversal. At each iteration, line 4 invokes a numerical update scheme to compute a discrete step from the current point (v, α) to the next point $(\tilde{v}, \tilde{\alpha})$ along the fiber. The dedicated sub-routine for this numerical update, called TAKE-STEP (line 4), is described in more detail in Sect. 7.2.4. After the update, lines 5-8 check whether α has changed sign, in which case it has crossed 0 and the update is passing through a new fixed point, by Prop. 3. The precise location of the new fixed point, denoted v^*, is found via local optimization seeded with v on line 6 before it is added to the running list V^* on line 7. Finally, the current point is updated to the new position before the next iteration (line 9). Traversal continues until a stopping condition is satisfied on lines 10-12.

7.2.4 The TRAVERSE Update Scheme

Numerical steps along $\Gamma^{(c)}$ use the unit tangent vector at the current point (v, α), denoted by z. The proof of Prop. 2 shows that z is the unique (up to sign) unit vector satisfying

$$DF^{(c)}(v, \alpha)z = \mathbf{0}. \tag{7.5}$$

In other words, the tangent spans the null space of $DF^{(c)}$, which is one-dimensional as long as $DF^{(c)}$ is full rank. Eq. 7.5 can be solved for z with standard linear algebra routines.

TRAVERSE(f, c)

1: Initialize $(v, \alpha) \in \Gamma^{(c)}$ with starting condition
2: $V^* \leftarrow \{\}$
3: **loop**
4: $\quad(\tilde{v}, \tilde{\alpha}) \leftarrow$ TAKE-STEP(f, c, (v, α))
5: $\quad$**if** $\text{sign}(\tilde{\alpha}) \neq \text{sign}(\alpha)$ **then**
6: $\quad\quad$Solve $f(v^*) = \mathbf{0}$ for v^* seeded with v
7: $\quad\quad V^* \leftarrow V^* \cup \{v^*\}$
8: $\quad$**end if**
9: $\quad(v, \alpha) \leftarrow (\tilde{v}, \tilde{\alpha})$
10: $\quad$**if** stopping condition is satisfied **then**
11: $\quad\quad$**return** V^*
12: $\quad$**end if**
13: **end loop**

Figure 7.3: Fiber-based traversal routine.

TAKE-STEP(f, c, (v, α))

1: Compute a suitable step size θ^*
2: Solve Eq. 7.6 for $x^{(\theta^*)} = (\tilde{v}, \tilde{\alpha})$, seeded with $x^{(0)} = (v, \alpha)$
3: **return** $(\tilde{v}, \tilde{\alpha})$

Figure 7.4: Numerical step sub-routine.

While Eq. 7.5 can be used to compute a tangent vector, it does not prescribe exactly how that tangent vector should be used. One possibility is to use a numerical integration scheme such as the Euler or Runge-Kutta methods, but the computed path may diverge from the true mathematical fiber over time. Since we have not only the tangent vector, but also the implicit equation $F^{(c)}(v, \alpha) = \mathbf{0}$ which defines $\Gamma^{(c)}$, we can go further. Using θ^* to denote the current step-size, and $x^{(0)}$ and $x^{(\theta^*)}$ to denote (v, α) and $(\tilde{v}, \tilde{\alpha})$, respectively, we can solve

$$G(x^{(\theta^*)}) = [\mathbf{0};\, \theta^*] \tag{7.6}$$

for $x^{(\theta^*)}$, seeded with $x^{(0)}$, where $G : \mathbb{R}^{N+1} \to \mathbb{R}^{N+1}$ is defined by

$$G(x) \stackrel{\text{def}}{=} [F^{(c)}(x);\, z^T(x - x^{(0)})]. \tag{7.7}$$

Eq. 7.6 simultaneously maintains $F^{(c)}(x^{(\theta^*)}) = \mathbf{0}$, which keeps $x^{(\theta^*)}$ in $\Gamma^{(c)}$, and enforces $z^T(x^{(\theta^*)} - x^{(0)}) = \theta^*$, which moves the traversal forward by a distance of θ^* in the tangent direction. This update scheme is a variant of *numerical path following*, for which other methods exist [6], but it has several novel aspects discussed in Sect. 7.5.

The step size must be derived with care. If too large, the traversal can "leap" to a remote point on $\Gamma^{(c)}$ or inadvertently reverse direction. If small enough, one can guarantee that $x^{(\theta^*)}$ converges to the same point that would have resulted from the mathematically ideal traversal: that is, the traversal where $x^{(0)}$ flows continuously along $\Gamma^{(c)}$, by a distance of θ^*, in the direction of z. However, if θ^* is *too* small, the traversal can be very slow, so θ^* should be maximized as much as possible while maintaining correctness. This may require leveraging particular properties of

the dynamical system at hand, which is done here for the specific case of RNNs. Theorem 5 and its proof sketch below outline the general derivation strategy. The theoretical and implementation details of the RNN-specific derivation are available in Appendix A.5.

Fig. 7.5 depicts the key quantities referenced by Theorem 5. Each $x^{(\theta)}$ solving $G(x^{(\theta)}) = [\mathbf{0}; \theta]$ is a point in $\Gamma^{(c)}$ that lies a distance of θ from $x^{(0)}$ in the direction of the tangent z. The fact that the map $\theta \mapsto x^{(\theta)}$ is a continuous bijection for all $\theta \in [0, \theta^*]$ guarantees that the same $x^{(\theta)}$ would result from the mathematically ideal traversal. It also guarantees that the transported tangent direction at $x^{(\theta)}$ must have non-negative dot-product with z, which can be used to avoid inadvertently reversing direction. θ^* is the largest step-size for which Theorem 5 makes these guarantees. This is the step-size used in the TAKE-STEP sub-routine. The step-size is adaptive, since it depends on the current point $x^{(0)}$.

Theorem 5. *Given any regular c and any $x^{(0)} \in \Gamma^{(c)}$, one can construct a neighborhood $\mathcal{U}^*$ around $x^{(0)}$ and a $\theta^* > 0$, such that for each $\theta \in [0, \theta^*]$, there is a unique $x^{(\theta)} \in \mathcal{U}^*$ solving $G(x^{(\theta)}) = [\mathbf{0}; \theta]$, and Newton's method will converge to $x^{(\theta)}$ when seeded with $x^{(0)}$. Furthermore, the resulting bijection $\theta \mapsto x^{(\theta)}$ is continuous on $[0, \theta^*]$.*

Proof of Theorem 5 (sketch). Define a norm $||\cdot||$ that will facilitate tight bounds, and given some θ, let $x^{(n)}$ denote the n^{th} Newton iterate. Combining formulas for Newton iterations and Taylor's theorem, derive a recurrence relation of the form

$$-DG(x^{(n)})(x^{(n+1)} - x^{(n)}) = R^{(n-1)}(x^{(n)} - x^{(n-1)}), \tag{7.8}$$

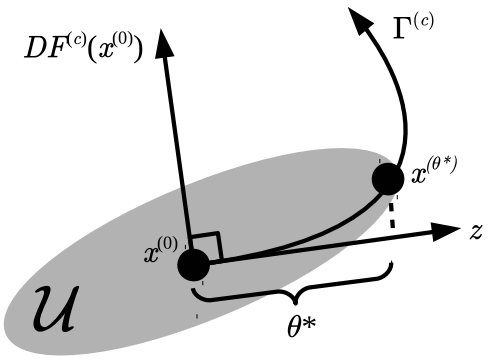

Figure 7.5: Key quantities in Theorem 5. $x^{(0)}$ is the current point on the fiber $\Gamma^{(c)}$. z is the tangent vector, which is orthogonal to the rows of $DF^{(c)}(x^{(0)})$ (in higher dimensions there are multiple row vectors). Steps in $\mathcal{U}$ are certified, θ^* is the maximal such step size, and $x^{(\theta^*)}$ is the new point after the step.

where $R^{(n-1)}(\cdot)$ is a second-order Taylor remainder. Taking norms on both sides of (A.31), obtain a bound of the form

$$||x^{(n+1)} - x^{(n)}|| \leq \rho^{(n)}||x^{(n)} - x^{(n-1)}||^2, \tag{7.9}$$

where $\rho^{(n)}$ is an expression whose contents depend on the particulars of $||\cdot||$, $DG(x^{(n)})$ and $R^{(n-1)}(\cdot)$. One can check that $x^{(1)} - x^{(0)} = \theta z$, which relates the recurrence to θ in the base case. If all the $\rho^{(n)}$ can be bounded by a single ρ, then iterating (7.9) from this base case gives

$$||x^{(n+1)} - x^{(n)}|| \leq (\rho\theta||z||)^{2^n}/\rho. \tag{7.10}$$

If, in addition,

$$\rho\theta||z|| < 1, \tag{7.11}$$

then (7.10) will imply that $x^{(n)}$ is a Cauchy sequence and hence convergent, and will also allow us to bound $x^{(n)}$ near $x^{(0)}$ as follows:

$$||x^{(n)} - x^{(0)}|| \leq \sum_{k=0}^{n-1} ||x^{(k+1)} - x^{(k)}|| \tag{7.12}$$

$$\leq \frac{1}{\rho} \sum_{k=0}^{n-1} (\rho\theta||z||)^{2^k} \tag{7.13}$$

$$\leq \frac{1}{\rho} \sum_{k=1}^{n} (\rho\theta||z||)^{k} \tag{7.14}$$

$$\leq \frac{1}{\rho} \left(\frac{\rho\theta||z||}{1-\rho\theta||z||} \right), \tag{7.15}$$

where (7.12) follows from the triangle inequality, (7.13) follows by substituting (7.10), (7.14) follows readily from (7.13), and (7.15) follows from the formula for geometric series.

Now consider any $\delta > 0$, which determines a neighborhood $\mathcal{U} = \{x : ||x - x^{(0)}|| < \delta\}$. For $x \in \mathcal{U}$, since x is bounded near $x^{(0)}$, $DG(x)$ is bounded near $DG(x^{(0)})$. If δ is not too large, $DG(x)$ can also be kept full rank, since it will be near $DG(x^{(0)})$, which is evaluated at a point on $\Gamma^{(c)}$ and hence full rank. Using these bounds on $DG(x)$, one can determine a single ρ that bounds any $\rho^{(n)}$ for which $x^{(n-1)}$ and $x^{(n)}$ fall inside $\mathcal{U}$. With this ρ fixed, consider any θ that satisfies (7.11) as well as

$$\frac{1}{\rho} \left(\frac{\rho\theta||z||}{1-\rho\theta||z||} \right) < \delta. \tag{7.16}$$

Using (7.12-7.16), show by mathematical induction that $x^{(n)} \in \mathcal{U}$ and $\rho^{(n)} < \rho$ for *all* iterates, so that (7.10) is always satisfied and the $x^{(n)}$ do indeed converge. Let $x^{(\theta)}$ denote their limit. To show that $x^{(\theta)}$ does in fact solve $G(x^{(\theta)}) = [\mathbf{0}; \theta]$, take

norms in the Newton iteration formula to get

$$||[\mathbf{0};\theta] - G(x^{(n)})|| \leq ||DG(x^{(n)})|| \cdot ||x^{(n+1)} - x^{(n)}||. \tag{7.17}$$

Given full rank DG in $\mathcal{U}$, we have $||DG(x^{(n)})|| > 0$, while $||x^{(n+1)} - x^{(n)}||$ approaches 0. So $G(x^{(n)})$ approaches $[\mathbf{0};\theta]$.

Finally, consider all such δ, each of which determines a corresponding $\mathcal{U}$, ρ and θ. If the expressions for ρ and θ in terms of δ are continuous, then the maximal θ can be taken as θ^*, and the corresponding neighborhood as $\mathcal{U}^*$. It remains to show that for any $\theta \in [0, \theta^*]$, $x^{(\theta)}$ is unique in $\mathcal{U}^*$, and a continuous function of θ. Consider $\theta_1, \theta_2 \in [0, \theta^*]$, and use Taylor's theorem, the full rank of DG in $\mathcal{U}^*$, and the fact that $||G(x^{(\theta_1)}) - G(x^{(\theta_2)})||_2 = |\theta_1 - \theta_2|$ to obtain

$$|\theta_1 - \theta_2| \geq \lambda ||x^{(\theta_1)} - x^{(\theta_2)}||_2, \tag{7.18}$$

where λ is a non-zero lower bound on the least singular value of DG. (7.18) shows that $\theta \mapsto x^{(\theta)}$ is continuous, and that if $x^{(\theta_1)} \neq x^{(\theta_2)}$, then $\theta_1 \neq \theta_2$. □

7.3 Application to Recurrent Neural Networks

This section describes how directional fibers can be applied to find fixed points of RNNs. Fixed points of RNNs can represent many things, but of particular relevance in this work is their utility as waypoints along itinerant attractor sequences. Using ideas from the GALIS framework, such attractor sequences can be used to "program" neural networks with cognitive-level behaviors. This makes them particularly relevant to a GALIS-based neural reimplementation of CERIL. The fixed

point solver described here is a tool that can be used to improve our technical understanding of neural attractor dynamics, thereby promoting transparency in neural implementations of CERIL and other systems.

7.3.1 Neural Network Model

As a starting point, this work is focused on a discrete-time, continuous-valued neural network model with no external stimuli. The update rule for a network with N units is:

$$v \mapsto \sigma(Wv), \tag{7.19}$$

where $v \in \mathbb{R}^N$ is a column vector of real-valued neural activations, $W \in \mathbb{R}^{N \times N}$ is a matrix of connection weights, and σ is the hyperbolic tangent function, applied coordinate-wise. Hence, for any given W, the function f is given by

$$f(v) = \sigma(Wv) - v. \tag{7.20}$$

An ideal solver should locate every v satisfying $f(v) = \mathbf{0}$. The following example characterizes the complexity of this problem. Suppose W is diagonal. Then $f(v) = \mathbf{0}$ reduces to N independent one-dimensional problems of the form $\sigma(W_{i,i}v_i) - v_i = 0$. If $W_{i,i} > 1$, then this equation has three solutions, which is apparent from plotting $\sigma(W_{i,i}v_i) - v_i$ as a function of v_i (Fig. 7.6). Therefore the full system has 3^N solutions. Consequently, any solver that enumerates every fixed point has worst-case complexity at least exponential in N. However, we can still ask that a good solver have low *work complexity*, defined as *the time spent per fixed*

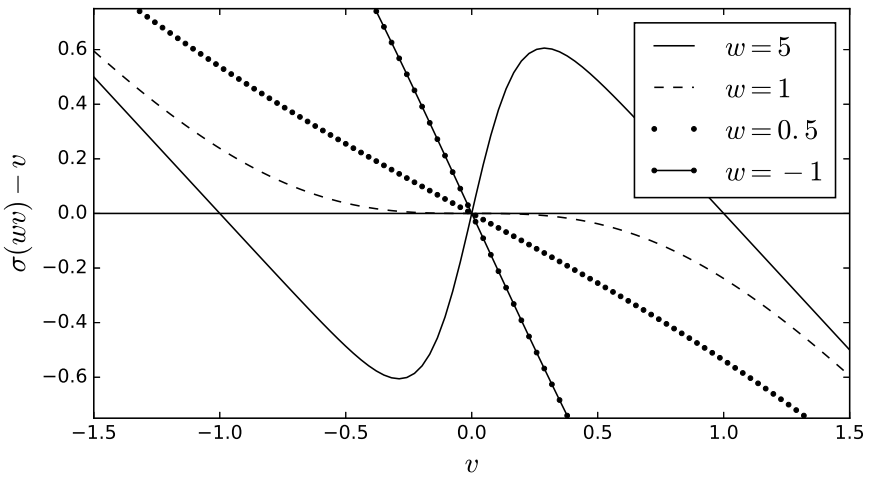

Figure 7.6: $\sigma(wv) - v$ for various w in the one-dimensional case.

point found. Section 7.4 characterizes the empirical work complexity of the solver both when the number of fixed points found is small and when it is large.

7.3.2 Applying TRAVERSE

To use TRAVERSE with f as in Eq. 7.20, Df must be full rank at every fixed point, as required by Props. 1 and 2. While this property has not been formally verified in general, it was apparently satisfied by every randomly sampled W in the experiments of Sect. 7.4 below. The RNN-specific derivation of Theorem 5 in Appendix A.5 also requires that W be invertible, but there are no other structural constraints (such as symmetry).

In addition to a suitable step-size, TRAVERSE also requires RNN-specific starting and stopping conditions. For the starting condition, note that $F^{(c)}(\mathbf{0}, 0) = \sigma(W\mathbf{0}) - \mathbf{0} - 0c = \mathbf{0}$, so the origin is always a valid initial point, for any W and c. For the stopping condition, we can prove the following:

Proposition 4. *Given fixed W, suppose that c is regular and that $W_{i,:}c \neq 0$ for all i. For any $(v, \alpha) \in \Gamma^{(c)}$, with tangent vector $z = (\dot{v}, \dot{\alpha})$, if*

$$|\alpha| > \frac{\sigma^{-1}\left(\sqrt{1 - \min\left\{1, \frac{1}{||W||_2}\right\}}\right) + \sum_j |W_{i,j}|}{|W_{i,:}c|} \tag{7.21}$$

for all i, then $\dot{\alpha} \neq 0$.

Proof of Proposition 4. Let $(v, \alpha) \in \Gamma^{(c)}$ satisfy Eq. 7.21. Writing $DF^{(c)}$ more explicitly in Eq. 7.5, we have

$$\Sigma W \dot{v} - \dot{v} - \dot{\alpha} c = \mathbf{0}, \tag{7.22}$$

where Σ is diagonal with $\Sigma_{i,i} = \sigma'(W_{i,:}v)$ and σ' denotes the derivative of σ with respect to its argument. If $\dot{\alpha} = 0$, then (7.22) implies that $||\Sigma W \dot{v}||_2 = ||\dot{v}||_2$. So

$$||\Sigma W \dot{v}||_2 < ||\dot{v}||_2 \text{ implies } \dot{\alpha} \neq 0. \tag{7.23}$$

By properties of $||\cdot||_2$, the antecedent of (7.23) is true whenever

$$\max_i \sigma'(W_{i,:}v) < 1/||W||_2 \tag{7.24}$$

is true. So it remains to show that Eq. 7.21 in Prop. 4 implies (7.24). Since $(v, \alpha) \in \Gamma^{(c)}$, for each i we have:

$$v_i = \sigma(W_{i,:}v) - \alpha c_i \tag{7.25}$$

$$|W_{i,:}v| \geq \Big||\alpha W_{i,:}c| - |W_{i,:}\sigma(Wv)|\Big| \tag{7.26}$$

$$\sigma'(W_{i,:}v) \leq \sigma'(|\alpha W_{i,:}c| - \textstyle\sum_j |W_{i,j}|), \tag{7.27}$$

where (7.25) follows by rearranging $F^{(c)}(v, \alpha) = \mathbf{0}$, (7.26) follows by multiplying both sides of (7.25) by W and using properties of $|\cdot|$, and (7.27) follows by applying

σ' to both sides of (7.26) and using properties of σ. So (7.24) is satisfied if

$$\sigma'(|\alpha W_{i,:}c| - \sum_j |W_{i,j}|) \leq \min\{1, 1/||W||_2\} \tag{7.28}$$

for all i. Using the fact that $\sigma' = 1 - \sigma^2$ and isolating $|\alpha|$ in (7.28) shows that (7.28) is equivalent to Eq. 7.21. □

The significance of Prop. 4 is that, as long as $\dot{\alpha} \neq 0$, α can not reverse direction. In particular, once $|\alpha|$ satisfies (7.21), α cannot return to 0, so no more fixed points will be encountered. Prop. 4 indicates at least one constraint on c: it must be chosen so that $W_{i,:}c \neq 0$ for each i. Since c is chosen randomly in the experiments here, this will be true with probability 1.

Although the range of σ is $(-1, 1)$, f is well-defined for any $v \in \mathbb{R}^N$, so directional fibers can leave and return to $(-1, 1)^N$ several times during traversal. However, since $\sigma(Wv) \in (-1, 1)^N$, any v solving $f(v) = \mathbf{0}$ must be in $(-1, 1)^N$. So there is no risk of encountering "fixed points" outside of the neural state space. Additionally, since σ is an odd function, directional fibers are always symmetric about $\mathbf{0}$. In particular, $-v$ is a fixed point whenever v is. So traversal need only proceed in one direction from $\mathbf{0}$, and the negations of each fixed point found can be added afterward.

7.3.3 Topological Sensitivity of Directional Fibers

By Prop. 3, any directional fiber contains every fixed point. However, the main hurdle faced by this approach is that for some choices of c, the fiber $\Gamma^{(c)}$ is not necessarily *connected*. Traversal of one connected component will fail to identify

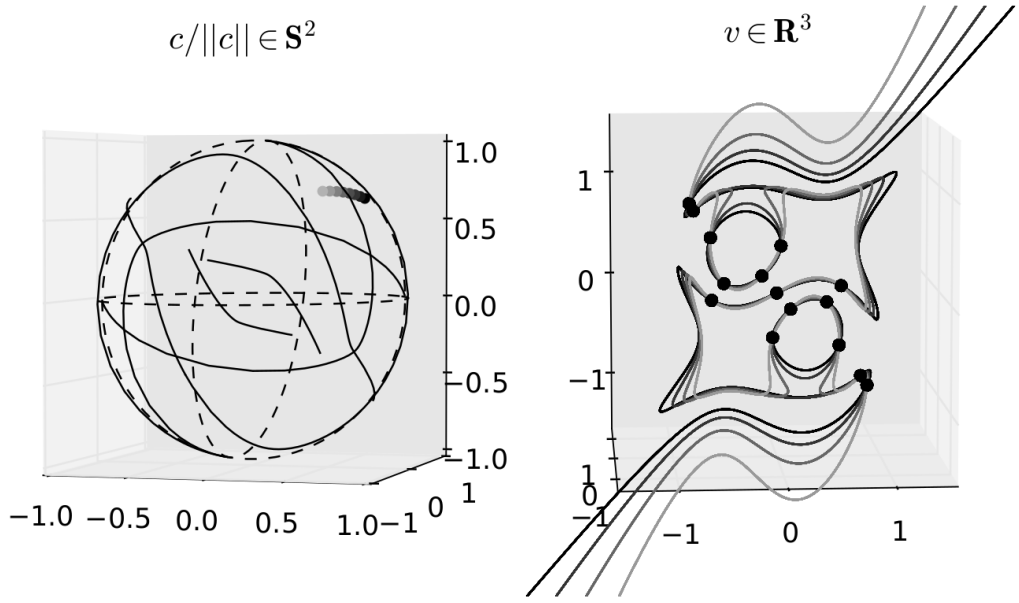

Figure 7.7: Topology of $\Gamma^{(c)}$ changes when c passes through a critical direction. Here an arbitrary 3-unit network is used for illustrative purposes. **Left**: The space of all choices of c, normalized to unit length ($\mathbf{S}^2$ denotes the unit sphere). The shaded circles varying from light to dark gray in the upper right indicate a particular sequence of c's that cross over the critical set. **Right**: The corresponding $\gamma^{(c)}$ for several c's in that sequence, in the same shade of gray, superimposed on the neural phase space. Additional details provided in the text.

fixed points contained in another connected component. This effect is illustrated in Fig. 7.7. As c is varied through a critical direction, the topology of $\Gamma^{(c)}$ can change: closed loops can form that are disconnected from the component of $\Gamma^{(c)}$ that contains the origin. Any fixed points on these closed loops will never be encountered by TRAVERSE when these c are used.

In effect, the set of all possible choices of c can be viewed as the unit hy-

persphere $\mathbb{S}^{N-1}$, and the critical directions partition the sphere into disjoint open sets where c is regular. Let us refer to these disjoint open sets as the *regular regions*. This is illustrated in Fig. 7.7. On the left of the figure, the dashed lines are bad choices where $W_{i,:}c = 0$ for some i (i.e., where the stopping condition (7.21) is undefined). The solid black curves are critical directions - bad choices for which $DF^{(c)}$ is singular at some $x \in \Gamma^{(c)}$. The critical sets were approximated using brute force methods on a finely sampled grid, which is not feasible in general, but possible in this low-dimensional example. The critical sets have zero measure, as expected from Prop. 1. The shaded circles varying from light to dark gray in the upper right indicate a particular sequence of c's that cross over the critical set. On the right of the figure, the corresponding $\gamma^{(c)}$ for several c's in that sequence, in the same shade of gray, are superimposed on the neural phase space. As c crosses over the critical set, two closed loops disconnect from the main body of the directional fiber. Solid black circles indicate fixed points of the network, which are at risk of being isolated on disconnected components of $\gamma^{(c)}$ for some choices of c.

In sum, choices of c in different regular regions can induce different topologies for $\Gamma^{(c)}$. If there is always one regular region in which $\Gamma^{(c)}$ is fully connected, and if there is an efficient algorithm that is always guaranteed to compute some c within this region, then the combination of this hypothetical algorithm with TRAVERSE would constitute a provably correct, global fixed point solver. Even if a result this strong cannot be obtained, it still may be possible that a *small subset* of regular regions can be identified, such that repeating TRAVERSE on a choice of c from each one, and taking the union of fixed points found by each repetition, can be guaranteed

to locate *most* of the fixed points of the network. It may even be possible that a single random choice of c will always locate a relatively large subset of fixed points with relatively high probability. The question of how and whether these possibilities can actually be realized in practice is an open problem for future study. Sect. 7.4.3 provides a preliminary experiment that sheds some light on this question.

7.4 Computer Experiments

A reference version of TRAVERSE was implemented for RNNs and subjected to a battery of computer experiments to gauge its efficacy.[2] In the first set of experiments, the approach is compared with the commonly used baseline of repeating local optimization on a large number of randomly sampled seeds. The second set of experiments compares the output of TRAVERSE using different choices of c.

7.4.1 Experimental Methods

7.4.1.1 Sampling Distribution for W

All experiments were performed on several randomly sampled networks, with network size N between 2 and 1024. At each N, several W's were randomly sampled: 50 at each $N \in \{2, 4, 7, 10, 13, 16\}$, 10 at each $N \in \{24, 32, 48, 64\}$, and 5 at each $N \in \{128, 256, 512, 1024\}$. Each W was constructed as follows: First, an $N \times N$ matrix V was randomly sampled with uniform i.i.d. entries in the interval $(-1, 1)$.

[2]The Python code for TRAVERSE, the computer experiments, and the figures are open-source and freely available at `https://www.github.com/garrettkatz/rnn-fxpts`.

Next, W was calculated with the formula $W = \sigma^{-1}(V)V^{-1}$. Substituting this formula for W into Eq. 7.19 shows that each column of V is a fixed point of the resulting network. This property was useful for comparing the output of different solvers with a set of fixed points that was known a priori. In the following these are referred to as the "known fixed points." Typically these networks possess many other initially unknown fixed points, in addition to those known by construction.

7.4.1.2 Counting Unique Fixed Points

To accurately compare the outputs of the solvers, it is important to accurately count the number of unique fixed points found by each. Determining whether a point should be considered fixed, and whether two fixed points should be considered distinct, are non-trivial problems in finite-precision arithmetic. The computed values of $f(v)$ at "fixed points" were generally a few multiples of machine precision and rarely identically $\mathbf{0}$. Likewise, any pair of "identical" fixed points were generally a few multiples of machine precision apart, and rarely identically equal.

Using an error analysis of $f(v)$ in finite-precision arithmetic, a test was devised that could decide either "no" or "maybe" as to whether a true fixed point existed within machine precision of a floating-point approximation. As such, strictly speaking, the fixed point counts reported here are upper bounds on the true counts. However, empirical evidence suggests that these bounds are tight. The details of this test and empirical evidence of its accuracy are provided in Appendix A.6.

Given two points $v^{(1)}$ and $v^{(2)}$ both classified as fixed, they were marked as

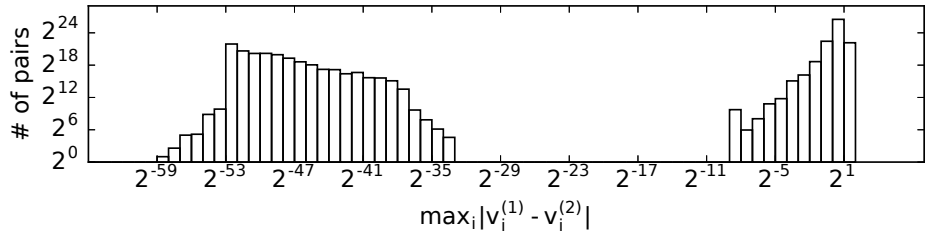

Figure 7.8: Pair-wise distances between fixed points before filtering for duplicates. Distances are computed within a large sample of pairs across all networks and solvers tested. $v^{(1)}$ and $v^{(2)}$ denote any such pair of fixed points.

duplicates if $\max_i |v_i^{(1)} - v_i^{(2)}| < 2^{-21}$. While simple, this method proved reliable, as confirmed empirically by computing pair-wise distances within the sets of points found by each solver on each network tested, before filtering for duplicates. Fig. 7.8 shows a histogram of these pair-wise distances,[3] aggregated across all solvers and all networks. 2^{-21} clearly separates by a large margin those distances that are near machine precision from those that are not. Several thousand pairs also had distances $\sim 2^{-1024}$ (data not shown), when both $v^{(1)}$ and $v^{(2)}$ were within machine precision of $\mathbf{0}$.

7.4.2 Comparison with a Baseline Solver

The common approach of solving $f(v) = \mathbf{0}$ with repeated, randomly initialized local optimization was used as a baseline for comparison with TRAVERSE. The first baseline implementation sampled the initial points uniformly, and then solved

[3]When a solver returned more than 1000 points, pair-wise distances were computed within a random 1000-point subset.

$f(v) = \mathbf{0}$ with Newton's method. However, on all but the smallest networks, almost every initial point converged to the trivial solution $v = \mathbf{0}$. In response, the final baseline implementation used here adopted the more sophisticated technique used by Sussillo and Barak [120]. Their technique starts by running the network dynamics and randomly sampling initial points along the observed trajectories. Each sample is then used as an initial point for an independent run of a local optimization routine, with $\frac{1}{2}||f(v)||_2^2$ as the objective function.[4] This objective function attains a minimum value of 0 at any fixed point v. It can also attain non-zero local minima at so-called "slow points" that are not fixed.[5] The optimization routine used is the trust-region Newton conjugate gradient method, provided with the Jacobian and the Gauss-Newton approximation to the Hessian. This technique is referred to as "the baseline method" for the remainder of this chapter.

The comparative study was conducted as follows. For each network in the test data, TRAVERSE was first run with a random choice of c. The choice of c was sampled with i.i.d Gaussian entries and then normalized to unit length, which results in a uniform distribution over the surface of the unit hypersphere. TRAVERSE was allowed to run either until the stopping condition was met or a maximum number

[4]Strictly speaking, Sussillo and Barak used a continuous-time network model, so they applied the minimization to the analogous continuous-time differential rather than a discrete-time difference.

[5]Slow points have proved useful in identifying non-fixed dynamical features such as line attractors [81]. Points along a directional fiber where $|\alpha|$ achieves a non-zero local minimum can be viewed as candidate slow points, so the method may also prove relevant in this regard. Exploring this possibility is an important future research direction.

of steps had been taken,[6] so that the run would terminate in a reasonable time frame. Next, the baseline method was applied to the same network. It was allowed to continue sampling and optimizing random points until the same amount of time had elapsed as had been spent by TRAVERSE. T is used to denote the set of fixed points found by TRAVERSE, and B is used to denote the set found by the baseline.

For each $v \in B$, its negative $-v$ was also added to B, for fair comparison with TRAVERSE which does the same. B was then post-processed by removing any slow points that were not fixed, and any fixed points that were duplicates.[7] Next, several set operations were performed on the processed outputs:

- $|T \cap B|$, to count the fixed points found by both methods;
- $|T \cup B|$, to count the fixed points found by either method;
- $|T - B|$ and $|B - T|$, to count the fixed points found by one method but not the other.

Fig. 7.9 shows the results. On average, for $N \leq 16$, $|B-T|$ was somewhat larger than $|T - B|$, indicating that the baseline was finding more fixed points. However, as N grew, $|T-B|$ was often significantly larger than $|B-T|$, indicating that TRAVERSE was finding more fixed points. In addition, $|T \cap B|$ approached 1, indicating that the fixed points found by each method were largely disjoint save for one common element, which turned out to be the trivial fixed point $\mathbf{0}$. Both methods found

[6] 2^{20} steps for $N \leq 256$, 2^{17} or 2^{15} for $N = 512$ or 1024, respectively.

[7] T does not include any duplicates or slow points by design, but was post-processed similarly as a sanity check.

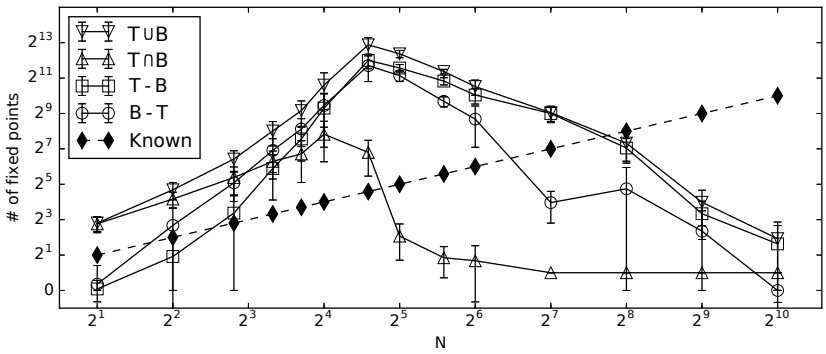

Figure 7.9: Fixed points found by TRAVERSE as compared with the baseline. The x-axis indicates network size N and the y-axis indicates set cardinalities, both on a log-scale. T and B denote the sets of fixed points found by TRAVERSE and the baseline, respectively. Each datapoint in the plot is the average cardinality of one set (either $|T \cap B|$, $|T \cup B|$, $|T - B|$, or $|B - T|$ as indicated), where the average is taken over all networks of a given size. Standard deviations of these cardinalities at each N are shown with error bars. The dashed line indicates the number of known fixed points at each N.

similar portions of the known fixed points, ranging from ∼100% at $N = 2$ to ∼0% at $N \geq 32$.

For $N \geq 32$, TRAVERSE consistently reached the maximum step count and terminated early. This effect was exaggerated after $N = 256$ when the step limit was reduced. The upward trend in fixed points found might have continued if TRAVERSE had run to completion, but this was not tested further due to computational cost. Both methods use many repetitions of matrix operations that are expensive for large N, and the trials for $N = 1024$ ran for several days. However, TRAVERSE often had lower *work* complexity, as shown in Figs. 7.10 and 7.11. In particular,

Fig. 7.11 shows that when run to completion, TRAVERSE (and the baseline) had runtime roughly proportional to fixed point counts, whether few or many total fixed points were found. On the other hand, for larger N, early termination renders the runtime roughly constant, so in this case both methods must be scaled up further before any conclusions can be made about absolute work complexities. Even so, the *relative* work complexity of TRAVERSE appears favorable at this scale, at least when run time must be limited. The space requirements of TRAVERSE are also lower, since it records each unique point at most once as it proceeds along the fiber. In contrast, before post-processing, the baseline had found many duplicates of the same fixed points (when different seeds converged to the same local optimum), and also many non-fixed slow points. The baseline typically stored ∼2-4 times as many points as TRAVERSE before post-processing. While this could have been counteracted by screening each candidate point online, rather than post-processing at the end, this would require additional time, and the baseline would find fewer total fixed points in the same timeframe. Since this approach could have been viewed as biasing the results towards TRAVERSE, it was not pursued here.

The disjointedness of T and B at larger N raises the question of whether each is an essentially random subset of the network's fixed points, or if their respective distributions in phase space differ in some more ordered way. This question was studied by calculating the average distance around the mean, and the average distance to the nearest corner of the state space $(-1, 1)^N$, across all points in T and in B. Fig. 7.12 shows the results. For $N \geq 32$, the baseline points were often farther from the means and closer to the corners. One might expect that this greater

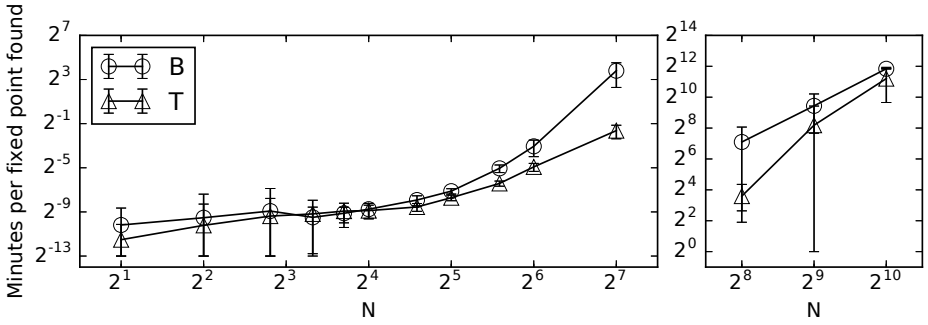

Figure 7.10: Comparison of work complexity (time spent per fixed point found, including post-processing) for each solver. Each datapoint is the average taken over all networks of a given size. Standard deviations are shown with error bars. Two y-axis scales are used for improved legibility.

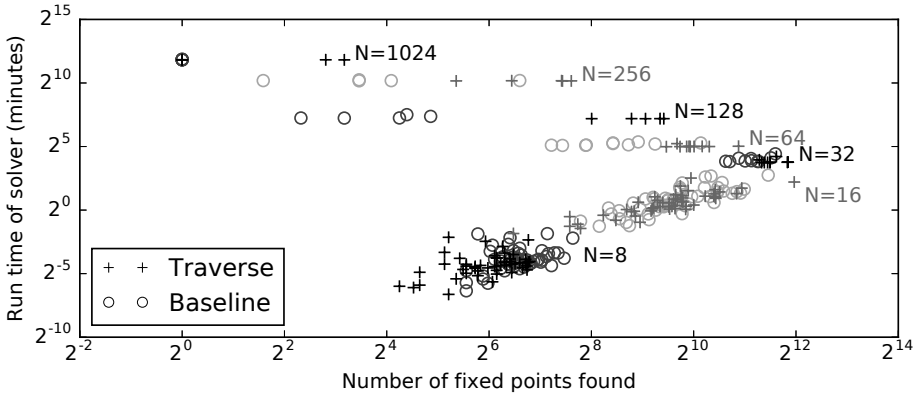

Figure 7.11: Scatter plot of absolute run times and fixed point counts. Each datapoint shows the run time and number of fixed points found by one solver on one network. Data across several network sizes N are shown, with different N labeled and shown in alternating shades of gray.

proximity to the corners is correlated with stability, which was checked using the eigenvalues of Dm at each fixed point found, where $m(v) = \sigma(Wv)$ is the update mapping. Eigenvalues with magnitude less than one (resp., greater than one) in-

dicate stable (resp., unstable) directions. Fixed points where all eigenvalues have magnitude less than one are therefore stable. As expected, the fixed points with larger norms tended to be more stable, as shown for an example network in Fig. 7.13. Fig. 7.14 shows the general trend over all networks as a function of N. Both stable and unstable points are located by both methods. However, as N increases, both methods tend to find more unstable points than stable ones, and TRAVERSE tends to find as many or more unstable points than the baseline. It is possible that the differences for $N \geq 32$ are not intrinsic properties of directional fibers, but rather due to the fact that TRAVERSE was starting from $\mathbf{0}$ and consistently terminating early on these trials before reaching the outer extremities of the fiber. Although this might be viewed as an artifact, it is also an important practical consideration: If TRAVERSE tends to locate more unstable points more quickly than the baseline, it may be particularly useful in applications where the *non-fixed* dynamics are of primary concern.[8]

[8]After the publication of [71] a methodological flaw was discovered in the stability anaysis. In particular, [71] used the eigenvalues of Df, which is the derivative of the *difference* function, rather than Dm, which is the derivative of the *update mapping*. The former is incorrect: it will sometimes falsely classify unstable points as "stable" and vice versa. The latter is correct; it is used here and in the most recent version of the code repository. Consequently the quantitative results as shown in Figs. 7.13 and 7.14 are slightly different from [71]. However, as it happens, there is no substantial qualitative difference in the results as a whole, and the same conclusions still hold.

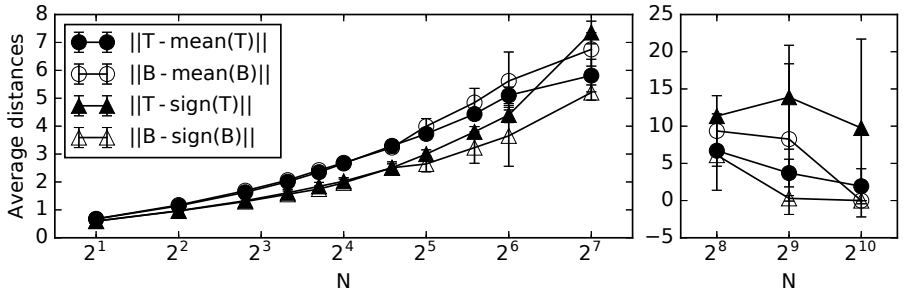

Figure 7.12: Statistics on the spatial distributions of fixed points found by TRAVERSE ("T") and the baseline ("B"). Each datapoint is the average taken over all fixed points found for all networks of a given size. Standard deviations are shown with error bars. Two y-axis scales are used for improved legibility.

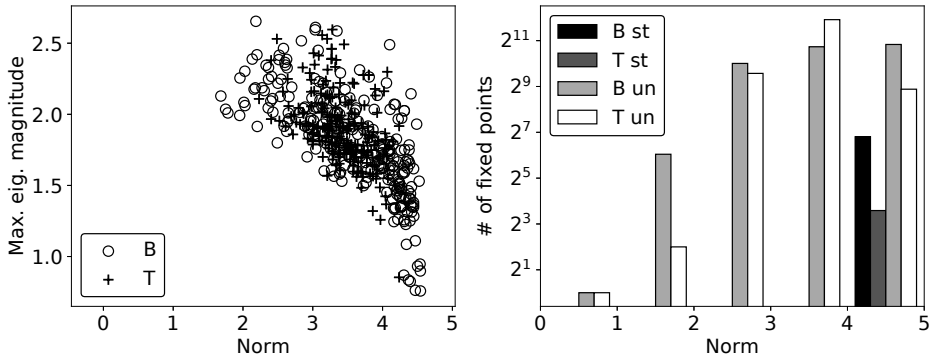

Figure 7.13: Stability of fixed points found in one sample network. **Left**: Each datapoint corresponds to a fixed point found by TRAVERSE ("T") or the baseline ("B"). The maximum eigenvalue magnitude of Dm at that point is plotted vs. the point's norm, where $m(v) = \sigma(Wv)$ is the update mapping. A maximum eigenvalue magnitude less than 1 indicates a stable point. **Right**: A histogram of the same data, showing the distribution of norms within the stable ("st") and unstable ("un") sets.

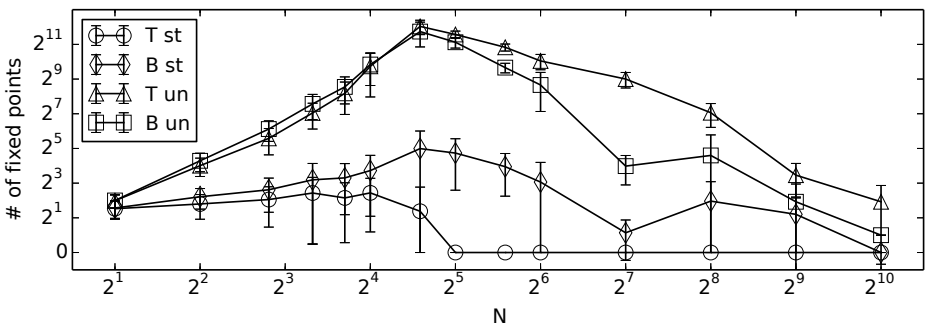

Figure 7.14: Counts of stable ("st") and unstable ("un") fixed points found by TRAVERSE ("T") and the baseline ("B"), averaged over all networks of a given size. Standard deviations are shown with error bars.

7.4.3 Comparison of Different Directional Fibers

The next experiment concerned the problem of choosing c. It was designed to test the hypothesis that for almost any network, there is a *single* choice of c with which TRAVERSE can locate *all* or at least *most* fixed points. For each network tested, many choices of c were generated (as detailed below), denoted $c^1, ..., c^k, ..., c^K$, and TRAVERSE was invoked on each one. Let $T(W, c)$ denote the set of fixed points found when using a direction c on a given set of network weights W. The union $\cup_k T(W, c^k)$ can be viewed as a first approximation to the full set of all fixed points of the network. If the hypothesis is true, we might expect to observe a single c^k whose individual $T(W, c^k)$ contains all or most points in the entire union $\cup_k T(W, c^k)$. Of course this experiment is by no means conclusive, since not every possible c can be checked, and $\cup_k T(W, c^k)$ is only a first approximation to the full set of fixed points. The goal was solely to collect some relevant preliminary results.

The choices of c^k used in this experiment were motivated by Eq. 7.21, which requires that $W_{i,:}c \neq 0$ for each i. Visual inspection of low-dimensional examples, as in Fig. 7.7, suggest that the c satisfying $W_{i,:}c = 0$ may be loosely correlated with the critical directions. As such, the regions of the sphere where all $W_{i,:}c \neq 0$ may be considered as rough proxies for the regular regions. There are 2^N such regions; one for each possible choice of $\text{sign}(Wc) \in \{-1, 1\}^N$. Let $s^k \in \{-1, 1\}^N$ denote the k^{th} possibility. A corresponding c^k can be found by solving the linear system $Wc^k = s^k$. In practice we can add small random noise to ensure that c^k is in general position and hence most likely a regular direction. Each c^k was computed in this way for each W that was tested, and then TRAVERSE was used to compute the corresponding $T(W, c^k)$. To assess the results, we can compute the following statistics for each W, shown in Fig. 7.15: $\min_k |T(W, c^k)|$, $\text{mean}_k |T(W, c^k)|$, $\max_k |T(W, c^k)|$, and $|\cup_k T(W, c^k)|$. In the figure k is suppressed for legibility. For this experiment N was capped at 10, since $N = 10$ already results in 2^{10} possible c^k, each of which must be used for another run of TRAVERSE.

As shown, even the worst choices of c^k (corresponding to $\min_k |T(W, c^k)|$) typically found more fixed points than were known by construction of W, and the single best choice of c^k (corresponding to $\max_k |T(W, c^k)|$) consistently located a large portion of the entire union. Even on average, most individual choices of c^k (corresponding to $\text{mean}_k |T(W, c^k)|$) were able to independently locate a non-negligible portion of the entire union. These results suggest that even choosing c at random can lead to reasonable performance of TRAVERSE, and for almost any network, there *may* be a single nearly optimal choice of c. On the other hand, as yet there is no

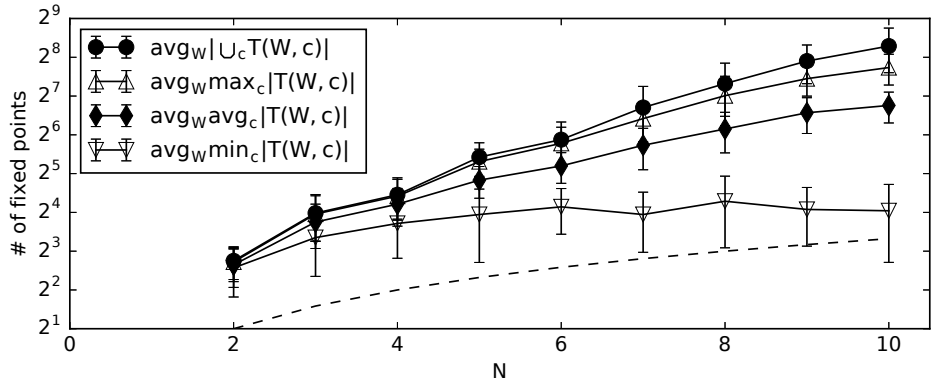

Figure 7.15: Statistics on the number of fixed points found by TRAVERSE, using different choices of c. $T(W, c)$ denotes the set of points found when using direction c on network weights W. Multiple c were tested with each W, and multiple W were tested at each network size N. Statistics on $T(W, c)$ were calculated across all c for a fixed W. Then averages and standard deviations of those statistics were calculated across all W for a fixed N. Those averages are plotted in the figure with standard deviations indicated by error bars. The dashed line indicates the number of fixed points known by construction.

discernible pattern governing *which* choice of s^k produced the best c^k on any given network. So, short of enumerating every s^k (which is infeasible for large N), it remains unknown how to identify the optimal c for any particular weight matrix W.

7.5 Discussion

This chapter has presented a new, general strategy for locating fixed points, based on directional fibers, and has shown how the strategy may be applied to RNNs. Compared with a traditional fixed point solver, using local search from

random seeds, this strategy often finds more fixed points, particularly on larger networks. The points found also tend to have different distributions in phase space, making the approach complementary to the baseline. Furthermore, the method has lower space requirements, returning points that are already guaranteed to be fixed and unique, as opposed to the baseline which may find many duplicated points.

Since TRAVERSE follows an implicitly defined curve, it is an example of numerical path following [6]. However, whereas the standard formulation of numerical path following involves a "predictor" step followed by a "corrector" step at every update, the update in this work is formalized using a single step comprised of the numerical solution of Eq. 7.6. Moreover, in addition to a robust stopping condition, this work provides a recipe for maximizing step size which formally guarantees that the numerical traversal matches the mathematically ideal traversal up to machine precision. Lastly, to my knowledge, the notion of *directional fibers* in particular as the paths to be followed is new. Numerical path following has been used for fixed point location before, but in a rather different way via homotopy continuation [26]. In these methods, a separate path is traced for each fixed point, and each such path originates from a known fixed point in a simpler dynamical system that is continuously transformed into the target dynamical system. In contrast, TRAVERSE uses a *single* path defined *within* the state space of the target dynamical system (i.e., a directional fiber), and which passes through *multiple* fixed points.

The main open questions in this approach are how and when a suitable directional fiber can be identified. However, the computer experiments show that the method locates a substantial number of fixed points even with random choices of

c. The experiments also provide preliminary empirical evidence that for many networks, there may exist a single nearly optimal choice of c that locates most fixed points. However, the question remains of how to *instantiate* this c in practice. This open problem is ripe for further theoretical and empirical investigation, which is the subject of future work. One avenue worth investigating is a *combination* of the baseline and TRAVERSE: since the points found by each are largely complementary, it appears likely that baseline points often lie on disconnected components of the fiber, and can be used to initialize subsequent runs of TRAVERSE along those disconnected components.

In addition to locating many fixed points, other criteria for choosing c should also be considered. Different c in the same regular region may find the same fixed points, but in very different running time, depending on the shapes of the corresponding fibers: If one fiber has sharper "bends", it may require smaller step sizes and hence more steps. Therefore the choice of c can also impact the work complexity of TRAVERSE.

Aside from choosing c, various other issues in the current work should be addressed. To begin with, the open question of whether almost every W satisfies Prop. 1 should be answered. In addition, the application of TRAVERSE to RNNs should be extended to handle networks with external input, and tested using other generative processes for W that are more representative of the networks used in modern machine learning. This includes W that are singular, sparse, larger, and/or trained (as in [120]).

The method also relies on a finite-precision error analysis for accurately count-

ing unique fixed points, which appears effective in practice but lacks full mathematical rigor. Aside from their bisection-based algorithms, existing rigorous global solvers also use provably correct data types that fully account for round-off errors [73,138]. Incorporating these data types into the implementation of TRAVERSE could resolve the issue.

Future work could also explore the applicability of directional fibers in other non-neural dynamical systems, although system-specific starting conditions, stopping conditions, and step sizes would have to be derived. In addition, finding zeros of a gradient can be viewed as finding fixed points of a vector field. So directional fibers may have relevance to (potentially non-convex) numerical optimization. Relatedly, the prospects of TRAVERSE for NP-hard optimization are intriguing but uncertain, and hinge on a better understanding of directional fibers, both in terms of correctness (i.e., connectedness) and complexity (i.e., amenability to fast traversal).

Lastly, future studies can put TRAVERSE to work in order to further deepen our understanding and engineering of fixed and non-fixed network dynamics. For example, although we contrast TRAVERSE with the baseline method as implemented by Sussillo and Barak, it was only a small part of their work [120]. Once the fixed points were located, they used an additional analysis, based on linearization around those fixed points, to form compelling explanations for how the networks behaved and how they accomplished the tasks for which they were trained. Repeating this analysis on fixed points located by TRAVERSE could yield additional insights. These insights would deepen our understanding of neural attractor dynamics, and in

turn could render the GALIS approach more transparent at a technical level [122]. Considering that GALIS is a promising approach for programming neural networks with cognitive-level behavior, this could ultimately lead to greater transparency and trustworthiness in a GALIS-based neural reimplementation of CERIL (and other systems).

Chapter 8: Discussion

8.1 Summary

As autonomous systems steadily become more intelligent and ubiquitous in every day life, it is increasingly important to ensure that these systems are pliable, transparent, and trustworthy for end users. The issue is compounded by wide-spread use of large and complex neural networks for autonomous control, which are very difficult to train and understand for non-experts.

Robotic imitation learning has emerged as an increasingly effective paradigm for pliable autonomy that can be shaped, understood, and trusted by end users. However, much past work on imitation learning has focused on sensorimotor-level behavior. At this level, the robot attempts to closely mimic the motor output of the demonstrator, limiting the robot's ability to generalize. Some past work has modeled various aspects of cognitive-level imitation learning, but cause-effect reasoning is a core aspect of human cognition but is under-represented in these systems.

This dissertation explored the hypothesis that cause-effect reasoning could be an effective platform for pliable and transparent cognitive-level imitation learning, and could facilitate generalization from just a single demonstration, much as people

do. That hypothesis was borne out by CERIL, the imitation learning framework developed in this dissertation. CERIL only requires one demonstration to successfully learn and generalize a skill. The learning mechanism is based on novel causal reasoning algorithms with strong formal guarantees. It has been empirically validated on a physical robot using a suite of assembly and maintenance tasks, including complex manipulations of non-convex assemblies in 3D. Lastly, CERIL can justify planned actions to a human end user. All of these characteristics promote pliability and transparency (P&T).

CERIL is currently implemented with symbolic computation, which facilitates P&T, but has limited capacity for learning and adaptability as compared with neurocomputational techniques. Conversely, because large and complex neural networks remain poorly understood and difficult to train, their benefits are generally accompanied by reduced P&T. As such, it is important to reconcile neural computation with P&T in CERIL and other autonomous systems. Work has already started on exploring whether CERIL can be translated to a purely neurocomputational system. In that context, this dissertation also puts forth a novel mathematical tool for analyzing neural network dynamics and improving their transparency. This is an important step towards the eventual goal of reimplementing CERIL with purely neural techniques while retaining P&T.

8.2 Contributions

The specific contributions of this dissertation are as follows.

- The first contribution is a novel architecture (CERIL) for imitation learning that combines abductive inference with hierarchical planning. As both can be viewed as forms of cause-effect reasoning, which is a quintessential cognitive faculty in humans, CERIL can be considered as a human-like model of cognitive-level imitation learning. This integration of two causal inference frameworks has the synergistic effect of enabling generalization from just a single demonstration, much as people do. CERIL is validated by empirical studies conducted in this work.

- The second contribution is an extension of Parsimonious Covering Theory (PCT) to simultaneously accommodate real-valued spatial information, causal chaining, and temporal ordering constraints. This extension comes with strong formal correctness and complexity guarantees, which are proven by theoretical analyses in this work. This extension extends the reach of PCT from fields such as medical diagnosis, fault localization, and semantic web technology to also apply in the field of imitation learning. In the context of automated planning, this extension amounts to a provably correct inversion of the HTN planning algorithm.

- The third contribution is the introduction of new parsimony criteria for PCT, specifically the "minimum parameters" (MP) criterion, and the first comparison of various parsimony criteria in the context of plan recognition. The results of empirical studies conducted in this work show that, for intention inference during imitation learning, the MP criterion is often more effective

than other criteria in certain regards, and some traditionally favored criteria can be disadvantageous.

- The fourth contribution is an XAI mechanism based on causal knowledge through which autonomous agents can justify their planned actions to a human end user. This mechanism has been packaged in a feature-rich graphical user interface, which can serve as an experimental scaffold for controlled studies of trustworthy autonomy with human participants.

- The final contribution is a step towards a transparent, purely neurocomputational implementation of CERIL. In particular, this work provides a new method for fixed point location in recurrent neural networks (and other dynamical systems), based on directional fiber traversal. Locating these fixed points is important for better understanding neural attractor dynamics, which in turn are relevant to GALIS-based architectures that can be "programmed" with cognitive-level behaviors, such as the behaviors exhibited by CERIL. Some theoretical properties of the method have been established, and empirical computer studies show that the method is competitive with and complementary to existing techniques.

8.3 Limitations and Future Work

Despite its successes, the current implementation of CERIL has a number of significant limitations. The first limitation is the heavy reliance on substantial background knowledge. Before CERIL can begin imitating, a human domain expert

is responsible for encoding its knowledge of the domain. This includes detailed models of the actions that can be performed and their effect on the environment. It also includes the database of causal knowledge relating intentions to the sub-intentions they can cause. This knowledge is represented by CERIL as largely unconstrained data structures and computer code, which affords greater flexibility, but requires significant time and effort on the part of one or more domain authors who are well versed both in the domain itself and in computer programming. Once this step is complete, CERIL is ready to imitate and learn from end users without programming or robotics expertise, but it is still a large capital expenditure.

In future work, this expense should be offset by enabling CERIL to acquire this knowledge from experience. The sensory changes that CERIL observes after executing motor commands can serve as training data for supervised and/or reinforcement learning processes, through which CERIL could learn to predict the effects of its actions on the environment. In addition, the actions observed in demonstrations could serve as training data for *induction* of new intentions to complement the current *abductive* inference process (e.g., using methodology from [56] or [141]). Reimplementation of CERIL with neural computation would also open new avenues for learning new domain knowledge before, during, and after imitation.

Another issue in the current implementation is CERIL is the limited use of *descriptive* machine representations (e.g., precondition and postcondition lists) during planning, as compared to *operational* representations (e.g., computer programs). CERIL's action justification method makes use of descriptive representations, but these are identified after planning is complete and not yet used during the planning

process itself. While operational representations afford more flexibility during planing, they limit CERIL's ability to introspect its own causal knowledge, so that more knowledge remains implicit in the head of the domain author rather than explicitly available to CERIL. In future work, shifting to the Goal-Task Network planning formalism would allow CERIL to plan with descriptive and operational knowledge simultaneously, achieving the best of both worlds.

CERIL's PCT algorithms for intention inference also have several limitations that could be improved in future work. As mentioned in Chapter 5, incorporating causal probabilities would be a significant improvement to expressive power and open up new opportunities for machine learning in CERIL. The causal semantics could also be relaxed to allow partial ordering, optional effects (i.e., not every sub-intention in a child sequence need always occur) and shared causes (i.e., two parent intentions causing the same child within the same covering graph). While CERIL's abductive inference algorithms draw heavily on PCT, they are not strictly compatible with the original formulation, in that problem instances of the original formulation are not subsumed as special cases. This would be remedied by the foregoing improvements. As also mentioned in Chapter 5, a limitation in the experimental methods is that the comparison of parsimony criteria could be sensitive to the domain encoding. Future work should repeat the experiments on more domains written by other authors.

To fully test the hypothesis that CERIL promotes P&T, it is essential in future work to conduct experiments with human participants who rate their experience using the system. These experiments should target the imitation learning system as a whole to gauge how easy it is for non-roboticists to teach the robot new skills.

They should also target CERIL's XAI interface to gauge how convincing or helpful the answers are to human end users. The results of these experiments could inform the continued design of the system and suggest new ways to improve P&T in CERIL.

Lastly, to improve CERIL's adaptability and learning in the messy real world, a promising strategy is to incorporate neural computation in the causal reasoning mechanisms (e.g., building on methodologies from GALIS [122]). Not only could this improve system performance, but it could also provide new insight into the neural basis of cognition and potentially even machine consciousness [106–108]. In order to maintain transparency in a neural re-implementation of CERIL, it is necessary to improve our fundamental understanding of neural networks and neural dynamics. To this end, future work should tackle the lingering open questions surrounding directional fiber traversal as posed in Chapter 7, and should apply the technique to more effectively verify and predict neural network behavior after training. If successful, this work would constitute an important step towards improved transparency in the upcoming generation of neurocomputational autonomous systems.

Appendix A: Appendix

A.1 Dock Maintenance Causal Relations

Below in teletype font are the core causal intention relations defined in the dock maintenance knowledge base, tailored to the Baxter robot. Each causal relationship is written with a "→" symbol. In some cases, the same cause can have multiple possible effect sequences. Three asterisks "***" are used to indicate the "intentions" that are considered directly observable as actions in the SMILE event transcript. The relations are ordered roughly from lowest-level (observable actions) to highest-level (top-level causes).

```
Causal relation:

plan-arm-motion(end-effector-targets) -->
  arm-trajectory(joint-angles)

move-arm-and-grasp(arm, object)*** -->
  plan-arm-motion(end-effector-targets), close-gripper(arm)

move-grasped-object(arm, destination)*** -->
  plan-arm-motion(end-effector-targets)
move-grasped-object(arm, destination)*** -->
  plan-arm-motion(end-effector-targets), insert(arm, destination)

put-down-grasped-object(arm, destination)*** -->
  move-grasped-object(arm, destination), open-gripper(arm)

parallel-hand-off(target-arm) -->
```

```
  plan-arm-motion(end-effector-targets), trade-grippers(target-arm)

perpendicular-hand-off(target-arm) -->
  plan-arm-motion(end-effector-targets), trade-grippers(target-arm)

drive-hand-off(target-arm) -->
  plan-arm-motion(end-effector-targets), trade-grippers(target-arm),
  plan-arm-motion(end-effector-targets), trade-grippers(source-arm),
  plan-arm-motion(end-effector-targets), trade-grippers(target-arm)

hand-off(target-arm) -->
  parallel-hand-off(target-arm)
hand-off(target-arm) -->
  perpendicular-hand-off(target-arm)
hand-off(target-arm) -->
  drive-hand-off(target-arm)

move-unobstructed-object(object, destination-or-arm) -->
  move-arm-and-grasp(arm, object), hand-off(target-arm),
  put-down-grasped-object(target-arm, destination)
move-unobstructed-object(object, destination-or-arm) -->
  move-arm-and-grasp(arm, object), hand-off(target-arm)
move-unobstructed-object(object, destination-or-arm) -->
  move-arm-and-grasp(arm, object),
  put-down-grasped-object(target-arm, destination)
move-unobstructed-object(object, destination-or-arm) -->
  move-arm-and-grasp(arm, object)
move-unobstructed-object(object, destination-or-arm) -->
  hand-off(target-arm),
  put-down-grasped-object(target-arm, destination)
move-unobstructed-object(object, destination-or-arm) -->
  hand-off(target-arm)
move-unobstructed-object(object, destination-or-arm) -->
  put-down-grasped-object(target-arm, destination)

free-gripper(arm, surface) -->
  put-down-grasped-object(arm, surface)
restore-gripper(arm, object) -->
  move-arm-and-grasp(arm, object)

move-object(object, destination) -->
  move-unobstructed-object(object, destination)
move-object(object, destination) -->
  free-gripper(arm, surface),
  move-unobstructed-object(object, destination)
move-object(object, destination) -->
  free-gripper(arm, surface),
```

```
  free-gripper(other-arm, surface),
  move-unobstructed-object(object, destination)

move-object-to-free-spot(object) -->
  move-object(object, free-spot)

discard-object(object) -->
  move-object(object, discard-bin)

open(dock-drawer)*** -->
  plan-arm-motion(end-effector-targets), close-gripper(arm),
  plan-arm-motion(end-effector-targets), open-gripper(arm)

close(dock-drawer)*** -->
  plan-arm-motion(end-effector-targets), close-gripper(arm),
  plan-arm-motion(end-effector-targets), open-gripper(arm)

press-dock-switch(arm,dock-switch,switch-state)*** -->
  plan-arm-motion(end-effector-targets), close-gripper(arm),
  plan-arm-motion(end-effector-targets), open-gripper(arm)

set-dock-switch(dock-switch,switch-state) -->
  press-dock-switch(arm,dock-switch,switch-state)
set-dock-switch(dock-switch,switch-state) -->
  free-gripper(arm,dock-case),
  press-dock-switch(arm,dock-switch,switch-state)
  restore-gripper(arm,object)
```

This list of causal relations is meant to paint a concrete picture of how much knowledge is built in to our system, but the following details are omitted from the list for clarity of presentation. The $\rightarrow$ symbol masks certain non-trivial operations necessitated by physical robot execution, as follows:

The `plan-arm-motion` relation invokes a motion planner to convert end-effector targets in 3D space to joint angle trajectories that avoid obstacle collisions. Grasping and putting down objects must incorporate geometric transformations describing the grasped object pose relative to the end-effector and relative to the destination, and must test for collisions when selecting which grasp pose to use for the manip-

ulated object. Drive hand-offs require three trades between grippers, during which the drive is gripped on its side, since the robot's arms are too thick for both grippers to be simultaneously positioned near the handle.

`move-grasped-object` includes a special branch for inserting drives into slots, since it is a fine motor skill that requires a special motor planning and execution routine. This distinction is not made in SMILE output.

`move-unobstructed-object` moves an object to either another destination object or to one of the arms. It assumes that one or both grippers are free as necessary and may or may not perform hand-offs depending on which arms can reach the source and destination positions.

`move-object` clears any grippers as necessary so that the unobstructed movement can be achieved. This requires identification of a free spot in the environment where currently gripped objects can be placed down so that the grippers are clear, which is accomplished using an evolutionary strategy in which every member of the evolving population is a candidate free spot. Candidates that are near to or overlapping with other objects are less fit.

In sum, parameters to the parents cannot simply be propagated to the children; the full causal relation is complex and non-deterministic. These complex relationships are accounted for in the CAUSES function when it processes a sequence of child intentions. There are also auxiliary causal relations necessary for physical execution but not modeled in CAUSES since they would not factor into intention inference. In particular, several intentions listed above include unshown children for visual processing routines that are interleaved with planning and execution, such as

inspecting the dock slots and LEDs after the drawer is opened and updating the object matching.

A.2 Monroe County Corpus Causal Relations

The causal relations used in the Monroe County Corpus Domain [18] are paraphrased below, with similar notation to Appendix A.1. Parameters are prefixed by '?' and primitive operator names by '!'. Most of these causal relations have preconditions that are not shown: the parent task can only cause its children when certain preconditions are satisfied in the current state. Moreover, several parent tasks have parameters that do not occur in the parameter lists of the children, and can only be inferred by inspecting the accompanying state. This logic is included in this work's implementation of CAUSES for the Monroe Domain, but omitted below for ease of presentation.

```
(set-up-shelter ?loc)-->
   ((get-electricity ?loc)
    (get-to ?leader ?loc)
    (get-to ?food ?loc))

(fix-water-main ?from ?to)-->
   ((shut-off-water ?from ?to)
    (repair-pipe ?from ?to)
    (turn-on-water ?from ?to))

(clear-road-hazard ?from ?to)-->
   ((block-road ?from ?to)
    (clean-up-hazard ?from ?to)
    (unblock-road ?from ?to))

(clear-road-wreck ?from ?to)-->
   ((set-up-cones ?from ?to)
    (clear-wreck ?from ?to)
    (take-down-cones ?from ?to))
```

```
(clear-road-tree ?from ?to)-->
   ((set-up-cones ?from ?to)
    (clear-tree ?tree)
    (take-down-cones ?from ?to))

(plow-road ?from ?to)-->
   ((get-to ?driver ?plowloc)
    (!navegate-snowplow ?driver ?plow ?from)
    (!engage-plow ?driver ?plow)
    (!navegate-snowplow ?driver ?plow ?to)
    (!disengage-plow ?driver ?plow))

(quell-riot ?loc)-->
   ((declare-curfew ?town) (get-to ?p1 ?loc) (get-to ?p2 ?loc)
    (!set-up-barricades ?p1) (!set-up-barricades ?p2)))

(provide-temp-heat ?person)-->
   ((get-to ?person ?shelter))

(provide-temp-heat ?person)-->
   ((generate-temp-electricity ?ploc) (!turn-on-heat ?ploc))

(fix-power-line ?lineloc)-->
   ((get-to ?crew ?lineloc) (get-to ?van ?lineloc)
    (repair-line ?crew ?lineloc))

(provide-medical-attention ?person)-->
   ((get-to ?person ?hosp) (!treat-in-hospital ?person ?hosp))

(provide-medical-attention ?person)-->
   ((emt-treat ?person))

(clean-up-hazard ?from ?to)-->
   ((!call fema))
(clean-up-hazard ?from ?to)-->
   ((get-to ?ht ?from) (!clean-hazard ?ht ?from ?to))

(block-road ?from ?to)-->
   ((set-up-cones ?from ?to) (get-to ?police ?from))
(block-road ?from ?to)-->
   ((get-to ?police ?from) (set-up-cones ?from ?to))

(unblock-road ?from ?to)-->
   ((take-down-cones ?from ?to))

(get-electricity ?loc)-->
```

```
   ((generate-temp-electricity ?loc))

(repair-pipe ?from ?to)-->
   ((get-to ?crew ?from)
    (set-up-cones ?from ?to)
    (open-hole ?from ?to)
    (!replace-pipe ?crew ?from ?to)
    (close-hole ?from ?to)
    (take-down-cones ?from ?to))

(open-hole ?from ?to)-->
   ((get-to ?backhoe ?from)
    (!dig ?backhoe ?from))

(close-hole ?from ?to)-->
   ((get-to ?backhoe ?from)
    (!fill-in ?backhoe ?from))

(set-up-cones ?from ?to)-->
   ((get-to ?crew ?from) (!place-cones ?crew))

(take-down-cones ?from ?to)-->
   ((get-to ?crew ?from) (!pickup-cones ?crew))

(clear-wreck ?from ?to)-->
   ((tow-to ?veh ?dump))

(tow-to ?veh ?to)-->
   ((get-to ?ttruck ?vehloc)
    (!hook-to-tow-truck ?ttruck ?veh)
    (get-to ?ttruck ?to)
    (!unhook-from-tow-truck ?ttruck ?veh))

(clear-tree ?tree)-->
   ((get-to ?tcrew ?treeloc) (!cut-tree ?tcrew ?tree)
    (remove-blockage ?tree))

(remove-blockage ?stuff)-->
   ((get-to ?crew ?loc)
    (!carry-blockage-out-of-way ?crew ?stuff))
(remove-blockage ?stuff)-->
   ((get-to ?stuff ?dump))

(declare-curfew ?town)-->
   ((!call EBS) (!call police-chief))
(declare-curfew ?town)-->
   ((!call police-chief) (!call EBS))
```

```
(generate-temp-electricity ?loc)-->
   ((make-full-fuel ?gen) (get-to ?gen ?loc))
    (!hook-up ?gen ?loc) (!turn-on ?gen))

(make-full-fuel ?gen)-->
   ((get-to ?gc ?ss) (add-fuel ?ss ?gc) (get-to ?gc ?genloc)
    (!pour-into ?gc ?gen))

(make-full-fuel ?gen)-->
   ((get-to ?gen ?ss) (add-fuel ?ss ?gen))

(add-fuel ?ss ?obj)-->
   ((!pay ?ss) (!pump-gas-into ?ss ?obj))
(add-fuel ?ss ?obj)-->
   ((!pump-gas-into ?ss ?obj) (!pay ?ss))

(repair-line ?crew ?lineloc)-->
   ((shut-off-power ?crew ?lineloc)
    (clear-tree ?tree)
    (!remove-wire ?crew ?lineloc)
    (!string-wire ?crew ?lineloc)
    (turn-on-power ?crew ?lineloc))
(repair-line ?crew ?lineloc)-->
   ((shut-off-power ?crew ?lineloc)
    (!remove-wire ?crew ?lineloc)
    (clear-tree ?tree)
    (!string-wire ?crew ?lineloc)
    (turn-on-power ?crew ?lineloc))
(repair-line ?crew ?lineloc)-->
   ((shut-off-power ?crew ?lineloc)
    (!remove-wire ?crew ?lineloc)
    (!string-wire ?crew ?lineloc)
    (turn-on-power ?crew ?lineloc))

(shut-off-power ?crew ?loc)-->
   ((!call ?powerco))

(turn-on-power ?crew ?loc)-->
   ((!call ?powerco))

(shut-off-water ?from ?to)-->
   ((!call ?waterco)))

(turn-on-water ?from ?to)-->
   ((!call ?waterco))
```

```
(emt-treat ?person)-->
   ((get-to ?emt ?personloc) (!treat ?emt ?person))

(stabilize ?person)-->
   ((emt-treat ?person))

(get-to ?person ?place)-->
   ((drive-to ?person ?veh ?place))
(get-to ?veh ?place)-->
   ((drive-to ?person ?veh ?place))
(get-to ?obj ?place)-->
   ((get-to ?veh ?objloc) (get-in ?obj ?veh)
    (get-to ?veh ?place)
    (get-out ?obj ?veh))
(get-to ?obj ?place)-->
   ((get-to ?veh ?objloc) (stabilize ?obj) (get-in ?obj ?veh)
    (get-to ?veh ?place) (get-out ?obj ?veh))

(drive-to ?person ?veh ?loc)-->
   ((!navegate-vehicle ?person ?veh ?loc))

(get-in ?obj ?veh)-->
   ((!climb-in ?obj ?veh))
(get-in ?obj ?veh)-->
   ((get-to ?person ?objloc) (!load ?person ?obj ?veh))

(get-out ?obj ?veh)-->
   ((!climb-out ?obj ?veh))
(get-out ?obj ?veh)-->
   ((get-to ?person ?vehloc) (!unload ?person ?obj ?veh))
```

A.3 State Reconstruction in the Monroe Corpus

As described in Sect. 5.1.2, the Monroe County Corpus includes planning trees of tasks and actions, but does not retain the initial and intermediate states that were visited during the HTN planning process. The following is an example entry taken verbatim from the corpus:

```
((PROVIDE-MEDICAL-ATTENTION PERSON-30029)
   ((GET-TO PERSON-30029 PARK-RIDGE)
```

```
        ((GET-TO DTRUCK1 STRONG)
            ((DRIVE-TO TDRIVER1 DTRUCK1 STRONG)
                (!NAVEGATE-VEHICLE TDRIVER1 DTRUCK1 STRONG)))
        ((GET-IN PERSON-30029 DTRUCK1)
            (!CLIMB-IN PERSON-30029 DTRUCK1))
        ((GET-TO DTRUCK1 PARK-RIDGE)
            ((DRIVE-TO TDRIVER1 DTRUCK1 PARK-RIDGE)
                (!NAVEGATE-VEHICLE TDRIVER1 DTRUCK1 PARK-RIDGE)))
        ((GET-OUT PERSON-30029 DTRUCK1)
            (!CLIMB-OUT PERSON-30029 DTRUCK1)))
   (!TREAT-IN-HOSPITAL PERSON-30029 PARK-RIDGE))
```

The top-level goal in this example is:

```
(PROVIDE-MEDICAL-ATTENTION PERSON-30029).
```

Its immediate child tasks are:

```
(GET-TO PERSON-30029 PARK-RIDGE),
(!TREAT-IN-HOSPITAL PERSON-30029 PARK-RIDGE)
```

and the observable action sequence is:

```
(!NAVEGATE-VEHICLE TDRIVER1 DTRUCK1 STRONG),
(!CLIMB-IN PERSON-30029 DTRUCK1),
(!NAVEGATE-VEHICLE TDRIVER1 DTRUCK1 PARK-RIDGE),
(!CLIMB-OUT PERSON-30029 DTRUCK1),
(!TREAT-IN-HOSPITAL PERSON-30029 PARK-RIDGE)
```

The latter sequence is an example of what is used as input to the EXPLAIN algorithm during its empirical evaluation. Let us refer to the top-level goals in the examples as the "original" or "ground-truth" top-level goals.

As seen in the example above, the initial and intermediate states used when originally generating the HTN plan trees are not retained in the corpus. However, these states often contain important information that is necessary to uniquely determine parent tasks for an observed child sequence. Fortunately, the states can be

partially reconstructed as follows. Each operator in the domain definition includes a list of preconditions, which specifies propositions that must be true in the state immediately before the operator is applied. The actions also include add and delete lists which specify propositions that become true or false in the state immediately after the action is applied. For example, the `(!navegate-vehicle ?person ?veh ?loc)` operator has the following signature:

Operator:

```
(!navegate-vehicle ?person ?veh ?loc)
```

Preconditions:

```
(person ?person) (vehicle ?veh) (atloc ?veh ?vehloc)
(atloc ?person ?vehloc) (can-drive ?person ?veh)
(not (wrecked-car ?veh)
```

Delete list:

```
(atloc ?veh ?vehloc) (atloc ?person ?vehloc)
```

Add list:

```
(atloc ?veh ?loc) (atloc ?person ?loc)
```

The preconditions enforce constraints such as the `?person` parameter actually being a person, and the person and vehicle being collocated. The add and delete lists change the person and vehicle location from the source to the destination. Leveraging these descriptive operators, an automatic procedure can be used to traverse the HTN plan tree in any test example, adding and removing propositions in each state along the way, according to the planning operator definitions. This approach was

used as a pre-processing step to generate sequences of partial states to accompany each testing example in the dataset. For instance, the following partial states were reconstructed for the HTN plan tree example given above:

State 1:

```
(PERSON, TDRIVER1) (VEHICLE, DTRUCK1)
```

Action 1:

```
(!NAVEGATE-VEHICLE TDRIVER1 DTRUCK1 STRONG)
```

State 2:

```
(ATLOC, TDRIVER1, STRONG) (VEHICLE, DTRUCK1)

(ATLOC, PERSON-30029, STRONG)

(PERSON, TDRIVER1) (ATLOC, DTRUCK1, STRONG)
```

Action 2:

```
(!CLIMB-IN PERSON-30029 DTRUCK1)
```

State 3:

```
(ATLOC, PERSON-30029, DTRUCK1) (ATLOC, TDRIVER1, STRONG)

(VEHICLE, DTRUCK1) (PERSON, TDRIVER1) (ATLOC, DTRUCK1, STRONG)
```

Action 3:

```
(!NAVEGATE-VEHICLE TDRIVER1 DTRUCK1 PARK-RIDGE)
```

State 4:

```
(ATLOC, PERSON-30029, DTRUCK1) (ATLOC, DTRUCK1, PARK-RIDGE)

(VEHICLE, DTRUCK1) (PERSON, TDRIVER1) (ATLOC, TDRIVER1, PARK-RIDGE)
```

Action 4:

```
(!CLIMB-OUT PERSON-30029 DTRUCK1),
```

State 5:

```
(ATLOC, PERSON-30029, PARK-RIDGE) (PERSON, TDRIVER1)

(ATLOC, DTRUCK1, PARK-RIDGE) (ATLOC, TDRIVER1, PARK-RIDGE)

(VEHICLE, DTRUCK1)
```

Action 5:

```
(!TREAT-IN-HOSPITAL PERSON-30029 PARK-RIDGE)
```

Each state was paired with its corresponding low-level action, according to the intention formalization described in Chapter 4, before being passed as input to EXPLAIN.

A.4 Anomalies in the Monroe Plan Corpus

There is a small collection of anomalous examples in the Monroe Plan Corpus, where the parameters of child tasks apparently conflict with the parameters of their parents. For example, the 1542nd example is:

```
((CLEAR-ROAD-TREE HENRIETTA-DUMP ROCHESTER-GENERAL)
    ((SET-UP-CONES HENRIETTA-DUMP ROCHESTER-GENERAL)
     (!PLACE-CONES TCREW1))
    ((CLEAR-TREE TREE-10264971) (!CUT-TREE TCREW1 TREE-10264971)
        ((REMOVE-BLOCKAGE TREE-10264971)
         (!CARRY-BLOCKAGE-OUT-OF-WAY TCREW1 TREE-10264971)))
    ((TAKE-DOWN-CONES HENRIETTA-DUMP ROCHESTER-GENERAL)
        ((GET-TO CCREW1 HENRIETTA-DUMP)
            ((GET-TO TTRUCK1 PITTSFORD-PLAZA)
               ((DRIVE-TO TTDRIVER1 TTRUCK1 PITTSFORD-PLAZA)
                (!NAVEGATE-VEHICLE TTDRIVER1 TTRUCK1 PITTSFORD-PLAZA)))
            ((GET-IN CCREW1 TTRUCK1) (!CLIMB-IN CCREW1 TTRUCK1))
            ((GET-OUT CCREW1 TTRUCK1) (!CLIMB-OUT CCREW1 TTRUCK1)))
        (!PICKUP-CONES CCREW1)))
```

The first GET-TO seeks to move CCREW1 to HENRIETTA-DUMP, but its sub-tree apparently gets CCREW1 to PITTSFORD-PLAZA. The plaza is in Pittsford, not Henrietta; and it's not in Rochester either, where ROCHESTER-GENERAL is. So these actions do not seem to accomplish the top-level goal, and the propagation of parameters from parent to children does not seem to match the method schema in the domain definition. At least two other examples are similar: 1298 and 2114. They seem to involve a common pattern of

```
(GET-TO
  (GET-TO
  GET-IN
  GET-OUT))
```

where a crew simply gets in and out of a vehicle without navigating anywhere else.

A.5 RNN-specific Numerical Update Scheme

As described in Chapter 7, at a given point $x^{(0)} \in \Gamma^{(c)}$, let z denote the tangent vector to $\Gamma^{(c)}$. As shown in that chapter, if the Jacobian $DF^{(c)}(x^{(0)})$ is full rank, then z is the unique (up to sign) unit vector satisfying

$$DF^{(c)}(x^{(0)})z = \mathbf{0}. \tag{A.1}$$

The numerical step advances $x^{(0)}$ by a distance of θ^* in the direction of z, resulting in a new point $x^{(\theta^*)} \in \Gamma^{(c)}$. The update scheme accomplishes this by using Newton's method to solve

$$G(x^{(\theta^*)}) = \begin{bmatrix} \mathbf{0} \\ \theta^* \end{bmatrix} \tag{A.2}$$

for $x^{(\theta^*)}$, seeded with $x^{(0)}$, where $G : \mathbb{R}^{N+1} \to \mathbb{R}^{N+1}$ is defined by

$$G(x) \stackrel{\text{def}}{=} \begin{bmatrix} F^{(c)}(x) \\ z^T(x - x^{(0)}) \end{bmatrix}. \tag{A.3}$$

Eq. A.2 simultaneously maintains $F^{(c)}(x^{(\theta^*)}) = \mathbf{0}$, which keeps $x^{(\theta^*)}$ in $\Gamma^{(c)}$, and enforces $z^T(x^{(\theta^*)} - x^{(0)}) = \theta^*$, which moves the traversal forward by a distance of θ^* in the tangent direction. This update is illustrated in Fig. 7.5.

As long as W is invertible, the step-size θ^* can be determined rigorously with strong formal guarantees. In particular, this section shows how to compute a θ^* for which the numerical update is *guaranteed to converge to the same point that would have resulted from the mathematically ideal traversal*: that is, the traversal in which $x^{(0)}$ flows continuously along $\Gamma^{(c)}$, by a distance of θ^*, in the direction of z. The conditions that θ^* must satisfy for this to hold are provided by Theorem 6 below. For greater notational ease in the statement and proof of this theorem, several auxiliary functions and quantities are defined as follows, some of which were shown in Fig. 7.5.

First let λ denote the smallest singular value of $DG(x^{(0)})\tilde{W}^{-1}$, where DG is the Jacobian of G and $\tilde{W}$ abbreviates

$$\begin{bmatrix} W & \mathbf{0} \\ \mathbf{0}^T & 1 \end{bmatrix}. \tag{A.4}$$

Next, given any $\varepsilon > 0$, define $\delta_i(\varepsilon) > 0$ to be the largest δ such that for $i \leq N$ and any $x \in \mathbb{R}^{N+1}$, if

$$|\tilde{W}_{i,:}(x - x^{(0)})| < \delta, \tag{A.5}$$

then

$$|\sigma'(\tilde{W}_{i,:}x) - \sigma'(\tilde{W}_{i,:}x^{(0)})| < \varepsilon. \tag{A.6}$$

$\delta_i(\varepsilon)$ is used to determine a neighborhood around $x^{(0)}$ in which $DG(x)$ remains close to $DG(x^{(0)})$, where "closeness" is measured by ε. Its computation is explained after the statement of the theorem and illustrated in Fig. A.1.

Based on $\delta_i(\varepsilon)$, we can define several intermediate bounds used by the theorem:

- $\Delta_i(\varepsilon) \stackrel{\text{def}}{=} \{x : |\tilde{W}_{i,:}(x - x^{(0)})| < \delta_i(\varepsilon)\}$,
- $\underline{\delta}(\varepsilon) \stackrel{\text{def}}{=} \min_i \delta_i(\varepsilon)$,
- $\underline{\Delta}(\varepsilon) \stackrel{\text{def}}{=} \{x : ||\tilde{W}(x - x^{(0)})|| < \underline{\delta}(\varepsilon)\}$,
- $\mu(\varepsilon) \stackrel{\text{def}}{=} \max_i \max_{x \in \Delta_i(\varepsilon)} \frac{1}{2}|\sigma''(\tilde{W}_{i,:}x)|$,
- and $\rho(\varepsilon) \stackrel{\text{def}}{=} \mu(\varepsilon)/(\lambda - \varepsilon)$.

Note that $\delta_i, \underline{\delta}, \mu$, and ρ are all positive for $\varepsilon \in (0, \lambda)$.

Finally, define

$$\Theta(\varepsilon) \stackrel{\text{def}}{=} \frac{1}{||\tilde{W}z||} \cdot \frac{\underline{\delta}(\varepsilon)}{1 + \rho(\varepsilon)\underline{\delta}(\varepsilon)}, \tag{A.7}$$

let

$$\varepsilon^* = \operatorname*{argmax}_{\varepsilon \in (0,\lambda)} \Theta(\varepsilon), \tag{A.8}$$

and let $\theta^* = \Theta(\varepsilon^*)$.

Theorem 6. *Given fixed c and invertible W, let $x^{(0)}$ be any point in $\Gamma^{(c)}$. Suppose $DF^{(c)}(x^{(0)})$ is full rank and z is the tangent vector spanning its null space. Then for each $\theta \in [0, \theta^*]$, there is a unique $x^{(\theta)} \in \underline{\Delta}(\varepsilon^*)$ satisfying*

$$G(x^{(\theta)}) = \begin{bmatrix} \mathbf{0} \\ \theta \end{bmatrix}, \tag{A.9}$$

and Newton's method, when seeded with $x^{(0)}$ and used to solve Eq. (A.9), will converge to $x^{(\theta)}$. Moreover, the resulting bijection $\theta \mapsto x^{(\theta)}$ is continuous on $[0, \theta^*]$.

In Theorem 6, each $x^{(\theta)}$ solving Eq. (A.9) is a point in $\Gamma^{(c)}$ that lies a distance of θ from $x^{(0)}$ in the direction of the tangent z. The fact that the map $\theta \mapsto x^{(\theta)}$ is a continuous bijection for all $\theta \in [0, \theta^*]$ guarantees that the same $x^{(\theta)}$ would result from the mathematically ideal traversal where x flows continuously along $\Gamma^{(c)}$ starting from $x^{(0)}$. The proof follows a common strategy of proving the IVT, based on Newton's method (e.g., [125, 140]). However, additional care is taken to keep an explicit bound on the region of convergence as large as possible, capitalizing on the specific characteristics of the network model studied here. In this proof the n^{th} iterate of Newton's method is denoted $x^{(n)}$. Whereas we solve for $x^{(\theta^*)}$ during fiber traversal since it is the largest step-size with a formal guarantee, in this proof we solve for $x^{(\theta)}$, for an arbitrary $\theta \in [0, \theta^*]$, to establish the guarantee.

Proof of Theorem 6. Let $\rho^*, \mu^*, \underline{\delta}^*, \underline{\Delta}^*$ abbreviate $\rho(\varepsilon^*), \mu(\varepsilon^*), \underline{\delta}(\varepsilon^*), \underline{\Delta}(\varepsilon^*)$. By rearranging Eq. (A.7), we get both

$$\rho^* ||\tilde{W} z|| \theta^* = \frac{\underline{\delta}^*}{\underline{\delta}^* + 1/\rho^*} < 1 \tag{A.10}$$

and

$$\frac{\rho^* ||\tilde{W} z||\theta^*}{1 - \rho^* ||\tilde{W} z||\theta^*} = \rho^* \underline{\delta}^*. \tag{A.11}$$

Now consider any $\theta \in [0, \theta^*]$. Given Eq. (A.10), the left-hand side of Eq. (A.11) is the closed form for a geometric series with ratio $\rho^* ||\tilde{W} z||\theta^*$. Combining with $\theta \leq \theta^*$, we have

$$\frac{1}{\rho^*} \sum_{k=1}^{\infty} (\rho^* ||\tilde{W} z||\theta)^k \leq \underline{\delta}^*. \tag{A.12}$$

Since $r^{2^k} \leq r^{k+1}$ for any positive r less than 1 and integer $k \geq 0$, Eq. (A.12) implies

$$\frac{1}{\rho^*} \sum_{k=0}^{\infty} (\rho^* ||\tilde{W} z||\theta)^{2^k} \leq \underline{\delta}^*. \tag{A.13}$$

We will bound the Newton iterates within $\underline{\Delta}^*$ using Eq. (A.13) as well as the following bound on the derivatives of G. Let x be any point in $\underline{\Delta}^*$. Explicitly differentiating G, we have

$$DG(x) = \begin{bmatrix} \Sigma'(x) W - I, & -c \\ z^T & \end{bmatrix}, \tag{A.14}$$

where $\Sigma'(x)$ abbreviates $\text{diag}_{i \leq N}(\sigma'(\tilde{W}_{i,:} x))$. By adding and subtracting $DG(x^{(0)}) \tilde{W}^{-1}$, we have

$$DG(x)\tilde{W}^{-1} = DG(x^{(0)})\tilde{W}^{-1} + \begin{bmatrix} (\Sigma'(x) - \Sigma'(x^{(0)})) & \mathbf{0} \\ \mathbf{0}^T & 0 \end{bmatrix}. \tag{A.15}$$

Since $x \in \underline{\Delta}^*$, we have

$$|\tilde{W}_{i,:}(x - x^{(0)})| \leq ||\tilde{W}(x - x^{(0)})|| \leq \underline{\delta}^* \leq \delta_i(\varepsilon^*) \tag{A.16}$$

for all $i \leq N$, which implies

$$\max_{i \leq N} |\sigma'(\tilde{W}_{i,:}x) - \sigma'(\tilde{W}_{i,:}x^{(0)})| \leq \varepsilon^* < \lambda \tag{A.17}$$

by the definition of δ_i and the constraint in Eq. (A.8) that $\varepsilon^* \in (0, \lambda)$. Therefore

$$\left|\left|\begin{bmatrix} (\Sigma'(x) - \Sigma'(x^{(0)})) & \mathbf{0} \\ \mathbf{0}^T & 0 \end{bmatrix}\right|\right| < \varepsilon^* < \lambda, \tag{A.18}$$

and combining with Eq. (A.15), we get

$$s^* > \lambda - \varepsilon^*, \tag{A.19}$$

where s^* is the minimal singular value of $DG(x)\tilde{W}^{-1}$.

We are now prepared to show that the Newton iterates converge. We will prove by induction that

$$||\tilde{W}(x^{(n+1)} - x^{(n)})|| \leq (\rho^*||\tilde{W}z||\theta)^{2^n}/\rho^* \tag{A.20}$$

for all iterates $x^{(n)}$. The induction relies on the formula for Newton iterations, which can be expressed as

$$\begin{bmatrix} \mathbf{0} \\ \theta \end{bmatrix} - G(x^{(n)}) = DG(x^{(n)})(x^{(n+1)} - x^{(n)}). \tag{A.21}$$

Eq. (A.21) is solved for $x^{(n+1)}$ on each iteration.

In the base case $n = 0$, writing Eq. (A.21) more explicitly, we have:

$$\begin{bmatrix} \mathbf{0} \\ \theta \end{bmatrix} = \begin{bmatrix} DF^{(c)}(x^{(0)}) \\ z^T \end{bmatrix} (x^{(1)} - x^{(0)}), \tag{A.22}$$

which is solved by $x^{(1)} - x^{(0)} = z\theta$, since z is a unit vector spanning the null space of $DF^{(c)}(x^{(0)})$. Therefore

$$||\tilde{W}(x^{(1)} - x^{(0)})|| = ||\tilde{W}z||\theta = (\rho^*||\tilde{W}z||\theta)^{2^0}/\rho^*, \tag{A.23}$$

and Eq. (A.20) is true with equality.

For the inductive case, suppose Eq. (A.20) is true for $k \leq n$. Then we have

$$||\tilde{W}(x^{(k)} - x^{(0)})|| \leq \sum_{j=0}^{k-1} ||\tilde{W}(x^{(j+1)} - x^{(j)})|| \tag{A.24}$$

$$\leq \sum_{j=0}^{k-1} (\rho^*||\tilde{W}z||\theta)^{2^j}/\rho^* \tag{A.25}$$

$$\leq \underline{\delta}^* \tag{A.26}$$

where Eqs. (A.24-A.26) follow by the triangle inequality, the inductive hypothesis, and Eq. (A.13), respectively. This shows that $x^{(n)}$ and $x^{(n-1)}$ are both in $\underline{\Delta}^*$.

Using $x^{(n)}, x^{(n-1)} \in \underline{\Delta}^*$ we derive a recursive relation on the iterates as follows. Recapitulating Eq. (A.21), the n^{th} and $(n+1)^{th}$ Newton iterates are computed according to

$$\begin{bmatrix} \mathbf{0} \\ \theta \end{bmatrix} - G(x^{(n-1)}) = DG(x^{(n-1)})(x^{(n)} - x^{(n-1)}) \tag{A.27}$$

$$\begin{bmatrix} \mathbf{0} \\ \theta \end{bmatrix} - G(x^{(n)}) = DG(x^{(n)})(x^{(n+1)} - x^{(n)}). \tag{A.28}$$

Subtracting (A.28) from (A.27) gives

$$G(x^{(n)}) - G(x^{(n-1)}) = DG(x^{(n-1)})(x^{(n)} - x^{(n-1)}) - DG(x^{(n)})(x^{(n+1)} - x^{(n)}). \tag{A.29}$$

By Taylor's theorem [44], $G(x^{(n)}) - G(x^{(n-1)})$ also satisfies

$$G(x^{(n)}) - G(x^{(n-1)}) = DG(x^{(n-1)})(x^{(n)} - x^{(n-1)}) + R^{(n-1)}, \tag{A.30}$$

with second-order remainder term $R^{(n-1)}$. Substituting (A.29) into (A.30) and canceling terms leaves

$$-DG(x^{(n)})(x^{(n+1)} - x^{(n)}) = R^{(n-1)}, \tag{A.31}$$

and explicitly differentiating DG shows that for $i \leq N$,

$$R_i^{(n-1)} = \frac{1}{2}\sigma''(\tilde{W}_{i,:}\tilde{x}^{(i,n)})(\tilde{W}_{i,:}(x^{(n)} - x^{(n-1)}))^2, \tag{A.32}$$

where each $\tilde{x}^{(i,n)}$ is a weighted average of $x^{(n)}$ and $x^{(n-1)}$, and hence also in $\underline{\Delta}^*$. As for $i = N+1$, differentiation shows that $R_{N+1}^{(n-1)} = 0$.

Inserting the product $\tilde{W}^{-1}\tilde{W} = I$ in the left-hand side of Eq. (A.31) and taking the norm of both sides, we have

$$||DG(x^{(n)})\tilde{W}^{-1}\tilde{W}(x^{(n+1)} - x^{(n)})|| = ||R^{(n-1)}||. \tag{A.33}$$

From Eq. (A.19), this implies

$$(\lambda - \varepsilon^*)||\tilde{W}(x^{(n+1)} - x^{(n)})|| \leq ||R^{(n-1)}||. \tag{A.34}$$

To bound $||R^{(n-1)}||$, we first note that since each $\tilde{x}^{(i,n)} \in \underline{\Delta}^*$, we have $\max_i \frac{1}{2}\sigma''(\tilde{W}_{i,:}\tilde{x}^{(i,n)}) \leq \mu^*$. Moreover, for any vector a, we have $||a||^2 \geq ||a^2||$, where the exponent inside the norm is taken coordinate-wise. This is true because

$$(||a||^2)^2 = \left(\sum_i a_i^2\right)^2 = \sum_i a_i^4 + \sum_{i \neq j} a_i^2 a_j^2 \geq \sum_i a_i^4 = ||a^2||^2. \tag{A.35}$$

Finally, $||R^{(n-1)}|| = ||R^{(n-1)}_{1:N}||$ since $R^{(n-1)}_{N+1} = 0$. Therefore from Eqs. (A.32), (A.34), and (A.35), we get

$$||\tilde{W}(x^{(n+1)} - x^{(n)})|| \leq \frac{\mu^*}{\lambda - \varepsilon^*}||\tilde{W}(x^{(n)} - x^{(n-1)})||^2 = \rho^*||\tilde{W}(x^{(n)} - x^{(n-1)})||^2. \tag{A.36}$$

Substituting from the inductive hypothesis on the right-hand side of Eq. (A.36), we have

$$||\tilde{W}(x^{(n+1)} - x^{(n)})|| \leq \rho^* \left((\rho^*||\tilde{W}z||\theta)^{2^{n-1}})/\rho^*\right)^2 = (\rho^*||\tilde{W}z||\theta)^{2^n}/\rho^*. \tag{A.37}$$

Hence the induction goes through for all n. The consequence is that $x^{(n)}$ is a Cauchy sequence, and therefore converges to a limit. It remains to show that the limit of $x^{(n)}$ is in fact a solution $x^{(\theta)}$ of

$$G(x^{(\theta)}) = \begin{bmatrix} \mathbf{0} \\ \theta \end{bmatrix} \tag{A.38}$$

that is unique in $\underline{\Delta}^*$, and that the associated map $\theta \mapsto x^{(\theta)}$ is continuous.

To show that $\lim_{n\to\infty} x^{(n)}$ is a solution of Eq. (A.38), we again take norms in the Newton iteration formula, which gives

$$\left|\left|\begin{bmatrix} \mathbf{0} \\ \theta \end{bmatrix} - G(x^{(n)})\right|\right| \leq ||DG(x^{(n)})\tilde{W}^{-1}|| \cdot ||\tilde{W}(x^{(n+1)} - x^{(n)})||. \tag{A.39}$$

Since $||DG(x^{(n)})\tilde{W}^{-1}|| > \lambda - \varepsilon^* > 0$, whereas $||\tilde{W}(x^{(n+1)} - x^{(n)})||$ approaches 0, it must be that

$$\left|\left|\begin{bmatrix} \mathbf{0} \\ \theta \end{bmatrix} - G(x^{(n)})\right|\right| \tag{A.40}$$

also approaches 0 (and hence $x^{(n)}$ approaches a solution $x^{(\theta)}$).

For uniqueness, we take norms using the first-order Taylor theorem, which shows for any x that

$$||G(x^{(\theta)}) - G(x)|| = ||DG(\tilde{x})\tilde{W}^{-1}\tilde{W}(x^{(\theta)} - x)|| \tag{A.41}$$

$$\geq (\lambda - \varepsilon^*)||\tilde{W}(x^{(\theta)} - x)||, \tag{A.42}$$

where $\tilde{x}$ is a weighted average of $x^{(\theta)}$ and x and hence in $\underline{\Delta}^*$. Therefore if $||G(x^{(\theta)}) - G(x)|| = 0$, then it must be that $||\tilde{W}(x^{(\theta)} - x)|| = 0$. In other words, if $G(x^{(\theta)}) = G(x)$, then $x^{(\theta)} = x$.

Lastly, take any $e > 0$. To show continuity, we must find some $d > 0$, such that for any $\hat{\theta} \in [0, \theta^*]$,

$$|\theta - \hat{\theta}| < d \text{ implies } ||W(x^{(\theta)} - x^{(\hat{\theta})})|| < e. \tag{A.43}$$

Taking $x = x^{(\hat{\theta})}$ in Eq. (A.42), we obtain

$$||G(x^{(\theta)}) - G(x^{(\hat{\theta})})|| \geq (\lambda - \varepsilon^*)||\tilde{W}(x^{(\theta)} - x^{(\hat{\theta})}||. \tag{A.44}$$

Noting that $||G(x^{(\theta)}) - G(x^{(\hat{\theta})})|| = |\theta - \hat{\theta}|$, we find that setting $d = e(\lambda - \varepsilon^*)$ is sufficient. □

The quantities $\delta_i(\varepsilon)$, $\mu(\varepsilon)$, and $\rho(\varepsilon)$ can all be computed for any given ε with elementary, albeit cumbersome, operations, based on the properties of σ. Since $\sigma'(r) = 1 - \sigma^2(r)$ for any $r \in \mathbb{R}$, σ' can be inverted as follows:

$$r = (\sigma')^{-1}(\sigma'(r)) = \pm\sigma^{-1}\left(\sqrt{1 - \sigma'(r)}\right). \tag{A.45}$$

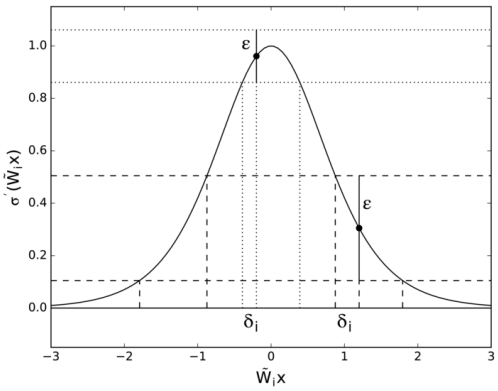

Figure A.1: Two examples of computing δ_i from ε, one in dashed lines and one in dotted lines. First, $\sigma'(\tilde{W}_i x^{(0)}) \pm \varepsilon$ is calculated (horizontal lines), representing endpoints of the range in which we want to bound σ'. Then, the calculated endpoints are passed through $(\sigma')^{-1}$ (vertical lines), to obtain the endpoints of the corresponding range in which we should bound $\tilde{W}_i x$. These endpoints are subtracted from $\tilde{W}_i x^{(0)}$ to obtain δ_i.

Using Eq. (A.45), $\delta_i(\varepsilon)$ can be computed as

$$\delta_i(\varepsilon) = \min\left\{\left|\pm\sigma^{-1}\left(\sqrt{1-(\sigma'(\tilde{W}_i x^{(0)}) \pm \varepsilon)}\right) - \tilde{W}_i x^{(0)}\right|, \infty\right\}, \tag{A.46}$$

where the minimum is taken over all choices of $\pm$ that produce real-valued results (e.g., the horizontal lines in Fig. A.1 that intersect the graph of σ'.). If none of the choices do, then $\delta_i(\varepsilon) = \infty$ signifies that any δ_i, no matter how large, satisfies the definition of $\delta_i(\varepsilon)$ in Theorem 6. Two examples of this computation are illustrated in Fig. A.1.

Differentiation shows that we can compute $\sigma''(r)$ directly as $\sigma''(r) = 2\sigma'(r)\sigma(r) = 2(1-\sigma^2(r))\sigma(r)$. Moreover, the maximum of $|\sigma''(r)|$ over any interval either occurs at one of the endpoints, or else is the global maximum of $|\sigma''(r)|$, namely $\sqrt{16/27}$,

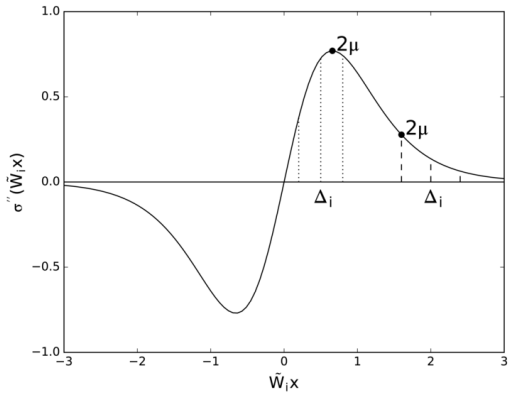

Figure A.2: Two examples of computing μ from $\delta_i(\varepsilon)$, one in dashed lines and one in dotted lines. First, $\tilde{W}_i x^{(0)} \pm \delta_i$ is calculated to obtain the endpoints of Δ_i. Then each endpoint is passed through σ'' to determine μ (vertical lines). 2μ is either the greater of the two endpoints, or the global maximum of σ'' if it is included in Δ_i.

which occurs at $r = \sigma^{-1}(\sqrt{1/3})$. So $\mu(\varepsilon)$ can be computed as

$$\mu(\varepsilon) = \frac{1}{2} \begin{cases} \sqrt{16/27} & \text{if } \sigma^{-1}(\sqrt{1/3}) \in \Delta_i(\varepsilon) \\ \max_i \left| \sigma''(\tilde{W}_i x^{(0)} \pm \delta_i(\varepsilon)) \right| & \text{otherwise,} \end{cases} \tag{A.47}$$

where the $\max_i$ is taken over each choice of sign for each i. This computation is illustrated in Fig. A.2.

Once each δ_i and μ are computed, ρ can be computed directly from its definition. As for ε^*, it can be approximated reasonably well by evaluating Eq. (A.8) at a modest number (16 in this work) of regularly spaced values of $\varepsilon \in (0, \lambda)$, thereby efficiently computing a step-size θ reasonably close to θ^*.

A corollary of Theorem 6 is that, while confined to $\underline{\Delta}(\varepsilon^*)$, the directional fiber cannot "double back" in the direction of $-z$ (otherwise, there would be two distinct $x^{(\theta)} \in \underline{\Delta}(\varepsilon^*)$ for the same θ, contradicting the theorem). In other words, the new

tangent vector after the step should have a positive dot product with the previous tangent vector before the step. This allows us to ensure that the numerical traversal never inadvertently reverses direction from one step to the next. Specifically, we can compute the new tangent vector after the step, denoted $\hat{z}$, by solving the linear system

$$\begin{bmatrix} DF(x^{(\theta)}) \\ z^T \end{bmatrix} \hat{z} = \begin{bmatrix} \mathbf{0} \\ 1 \end{bmatrix} \tag{A.48}$$

for $\hat{z}$ and then normalizing $\hat{z}$ to unit magnitude. This ensures both that $\hat{z}$ spans the null space of $DF(x^{(\theta)})$, so that it is tangent to $\Gamma^{(c)}$ at $x^{(\theta)}$, and also that $z^T\hat{z} > 0$, so that traversal continues in the right direction.

A.6 Counting Unique Fixed Points in Finite Precision

As described in Sect. 7.4.1.2, in order to accurately assess the performance of TRAVERSE, it is important to accurately count the number of unique fixed points found. Determining whether a point should be considered fixed, and whether two fixed points should be considered identical or distinct, are non-trivial problems in finite-precision arithmetic. The computed value of $f(v)$ at "fixed points" was generally a few multiples of machine precision and rarely identically $\mathbf{0}$. Similarly, any pair of "identical" fixed points were generally a few multiples of machine precision apart, and rarely identically equal.

To decide whether a point should be considered fixed, a forward error analysis

of f yields the following upper bound:

$$|\overline{f(\overline{v})} - f(v)| \leq \mathcal{E}(\overline{v}), \tag{A.49}$$

where

$$\mathcal{E}(\overline{v}) \stackrel{\text{def}}{=} \overline{|W|\epsilon(\overline{v})} + N\epsilon(\overline{|W|\overline{|v|}}) + 5\epsilon(\overline{\sigma(\overline{Wv})}) + \epsilon(\overline{v}) + \max(\epsilon(\overline{\sigma(\overline{Wv})}), \epsilon(\overline{v})), \tag{A.50}$$

and where all operations (except matrix multiplication) are applied coordinate-wise, and the inequality is true in every coordinate. The overbars denote the closest finite-precision approximation to an infinite-precision value, and $\epsilon(\cdot)$ denotes machine precision at a given finite-precision value. The coefficient of 5 bounds the relative error of σ. Rather than inspecting the machine implementation of hyperbolic tangent, this coefficient was estimated empirically based on the evaluation of $\sigma(\overline{x})$ at 2^{16} values of $\overline{x}$ uniformly sampled from $[0, 1]$. At a true fixed point v, $f(v) = \mathbf{0}$, and $|\overline{f(\overline{v})} - f(v)| = |\overline{f(\overline{v})} - \mathbf{0}| = |\overline{f(\overline{v})}|$, so any finite-precision point $\overline{v}$ satisfying $|\overline{f(\overline{v})}| > \mathcal{E}(v)$ can be rejected as certainly not fixed.

As a sanity check, histograms were computed of the relative errors at points accepted and rejected as fixed according to this test. Specifically, define the relative error $\mathcal{RE}$ as

$$\mathcal{RE}(\overline{v}) \stackrel{\text{def}}{=} \max_i |\overline{f_i(\overline{v})}|/\mathcal{E}_i(\overline{v}), \tag{A.51}$$

where the index i ranges over the coordinates of f (from 1 to N). The test rejects $\overline{v}$ as certainly not within machine precision of a true fixed point if $\mathcal{RE}(\overline{v}) > 1$. Otherwise it accepts $\overline{v}$ as potentially within machine precision of a true fixed point.

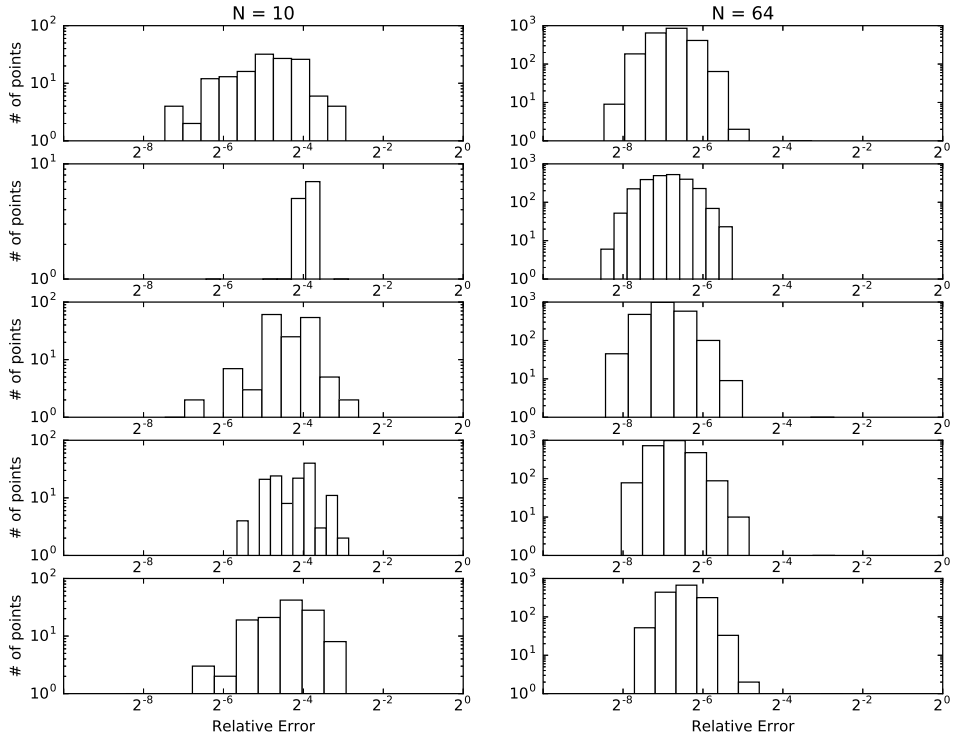

Figure A.3: Histograms of relative errors $\mathcal{RE}(\overline{v})$ at points found by fiber traversal. Each and every fixed point was accepted as fixed by a large margin.

$\mathcal{RE}$ can be extremely large since $\mathcal{E}(\overline{v})$ is generally near machine precision, but when v is not fixed $f(v)$ can be much larger than machine precision.

Each panel in Fig. A.3 shows the relative errors at "fixed" points found by a single fiber traversal on a single network. The left column contains panels for five networks with $N = 10$ and the right column contains panels for five networks with $N = 64$. The histograms show that each and every point identified by fiber traversal was accepted as fixed, by a wide margin, demonstrating that the theoretical results and error analysis are highly consistent.

Each panel in Fig. A.4 shows the results for the baseline solver described in

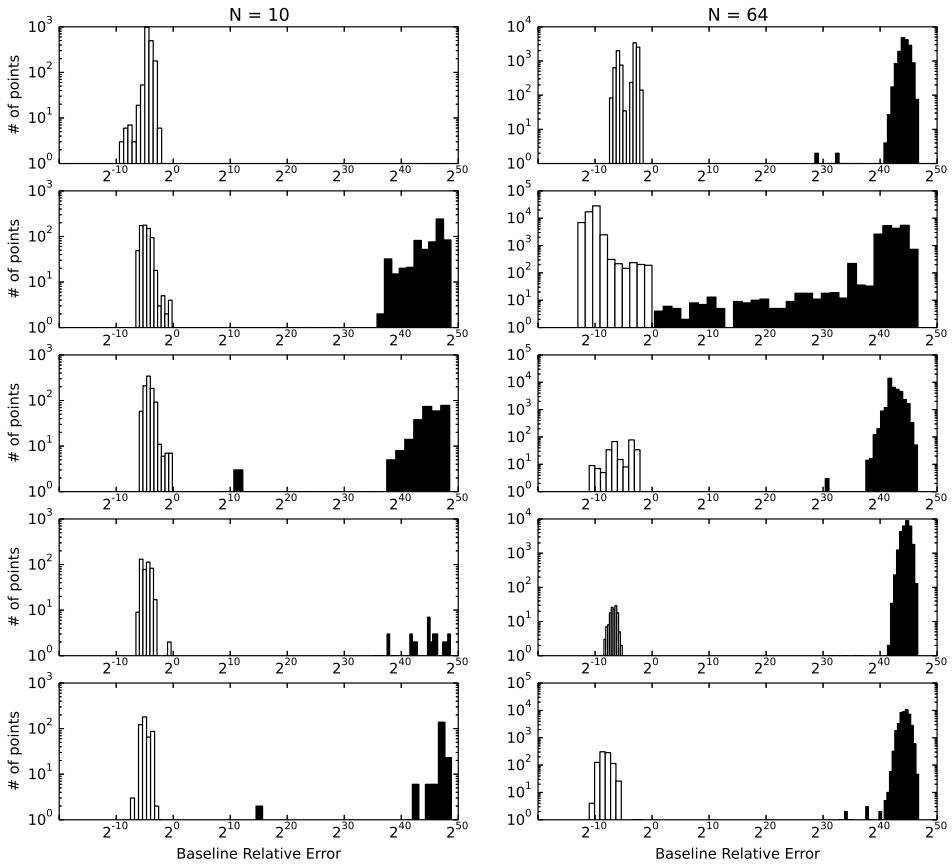

Figure A.4: Histograms of relative errors $\mathcal{RE}(\overline{v})$ at points found by the baseline fixed point solver, colored according to whether they were accepted as fixed (white bars) or rejected as not fixed (black bars).

Chapter 7 on a single network. As in Fig. A.3, the left column shows five networks with $N = 10$ and the right shows five networks with $N = 64$. The baseline solver can locate either fixed points or so-called "slow" points that are not fixed but are local minima of $||f(v)||$. Points accepted as fixed are shown in white and points rejected as not fixed are shown in black. Although the distinction was typically clear cut, there were some edge cases which call the fidelity of the error analysis into question

(middle panel on left, second panel from top on right). However, it is important to note that these histograms are shown with a log-scale on the y-axis. When shown normally, the edge cases are mostly invisible.

Nevertheless, some of the results in Sect. 7.4 rely on an accurate comparison of fiber traversal with the baseline solver. In particular, the results therein rely on the metrics $|T-B|$ and $|B-T|$, where T is the set of fixed points found by traversal and B is the set of points found by the baseline. $|T-B|$ measures the number of points that were found by the former but not the latter, and vice versa for $|B-T|$. The edge cases in these histograms raise the question of whether allegedly larger values of $|T-B|$ than $|B-T|$ are in fact artifacts of a flawed error analysis.

To dispel these concerns, the results were quantitatively inspected for each questionable histogram. For example, consider the second histogram from top in the right column of Fig. A.4. On this network, $|T-B|$ and $|B-T|$ were measured to be 694 and 476, respectively. Let us assume an inordinate worst case and suppose that every point in the histogram bins ranging all the way from $\mathcal{RE} = 2^0$ to 2^{20} was actually fixed and incorrectly classified as "not fixed" by the rejection test. Additionally let us even suppose that each point in these bins was a distinct fixed point with no duplicates. Even then, these bins contain only 86 points, which cannot account for even half of the difference $|T-B|-|B-T|$. The same check was performed on every network with size $N \in \{24, 32, 48, 64\}$ (where it was claimed that $|T-B|$ was significantly larger than $|B-T|$), again using the generous cap of 2^{20}. On average, the number of points in the questionable bins was only 27.7% of $|T-B|-|B-T|$. So we can be quite confident that the results reported in Sect.

7.4 are essentially correct.

As justified empirically in Sect. 7.4.1.2, given two points $\overline{v}^{(1)}$ and $\overline{v}^{(2)}$ that had both been classified as fixed, they were considered duplicates only if

$$\max_i |\overline{v}_i^{(1)} - \overline{v}_i^{(2)}| < 2^{-21}. \tag{A.52}$$

Based on this test, unique fixed points are extracted from a set with duplicates as follows. First, an adjacency graph is formed, where two points are adjacent if they were detected as duplicates. Next, the connected components of the graph are identified. Finally, one representative unique fixed point is chosen from each connected graph component.

Bibliography

[1] Pieter Abbeel, Adam Coates, and Andrew Y Ng. Autonomous helicopter aerobatics through apprenticeship learning. *The Intl. Jrnl. of Robotics Research*, 29(13):1608–1639, 2010.

[2] M. F. Afzal and A. A. Minai. Reliable storage and recall of aperiodic spatiotemporal activity patterns using scaffolded attractors. In *2016 Intl. Joint Conf. on Neural Networks*, pages 2047–2054, July 2016.

[3] Sungjin Ahn, Heeyoul Choi, Tanel Pärnamaa, and Yoshua Bengio. A neural knowledge language model. *arXiv preprint arXiv:1608.00318*, 2016.

[4] Sreeram VB Aiyer, Mahesan Niranjan, and Frank Fallside. A theoretical investigation into the performance of the Hopfield model. *IEEE Trans. on Neural Networks*, 1(2):204–215, 1990.

[5] Ron Alford, Vikas Shivashankar, Mark Roberts, Jeremy Frank, and David W Aha. Hierarchical planning: Relating task and goal decomposition with task sharing. In *IJCAI*, pages 3022–3029, 2016.

[6] Eugene L Allgower and Kurt Georg. Numerical path following. *Handbook of Numerical Analysis*, 5(3):207, 1997.

[7] Daniel J Amit. *Modeling brain function: The world of attractor neural networks*. Cambridge University Press, 1992.

[8] John R Anderson, Daniel Bothell, Michael D Byrne, Scott Douglass, Christian Lebiere, and Yulin Qin. An integrated theory of the mind. *Psychological Review*, 111(4):1036, 2004.

[9] Marcin Andrychowicz and Karol Kurach. Learning efficient algorithms with hierarchical attentive memory. In *1st Workshop on Neural Abstract Machines & Program Induction (NAMPI), Neural Information Processing Systems*, 2016.

[10] Brenna Argall, Brett Browning, and Manuela Veloso. Learning mobile robot motion control from demonstrated primitives and human feedback. In *Robotics Research*, pages 417–432. Springer, 2011.

[11] D. Baldwin and J. Baird. Discerning intentions in dynamic human action. *Trends in Cog. Sciences*, 5(4):171–178, 2001.

[12] João José Oliveira Barros, Vítor Manuel Ferreira dos Santos, and Filipe Miguel Teixeira Pereira da Silva. Bimanual haptics for humanoid robot teleoperation using ROS and V-REP. In *Intl. Conf. on Auton. Robot Sys. and Competitions*, pages 174–179. IEEE, 2015.

[13] Mathias Bauer and Gabriele Paul. Logic-based plan recognition for intelligent help systems. In *Current Trends in AI Planning: EWSP*, pages 60–73, 1993.

[14] R Ben-Yishai, R Lev Bar-Or, and H Sompolinsky. Theory of orientation tuning in visual cortex. *Proceedings of the National Academy of Sciences*, 92(9):3844–3848, 1995.

[15] Aude Billard, Sylvain Calinon, Ruediger Dillmann, and Stefan Schaal. Robot programming by demonstration. In *Springer handbook of robotics*, pages 1371–1394. Springer, 2008.

[16] Aude Billard and Daniel Grollman. Imitation learning in robots. In *Encyclopedia of the Sciences of Learning*, pages 1494–1496. Springer, 2012.

[17] Aude G Billard, Sylvain Calinon, and Rüdiger Dillmann. Learning from humans. In *Springer Handbook of Robotics*, pages 1995–2014. Springer, 2016.

[18] N. Blaylock and J. Allen. Generating artificial corpora for plan recognition. In *Intl. Conf. on User Modeling*, pages 179–188. Springer, 2005.

[19] N. Blaylock and J. Allen. Hierarchical instantiated goal recognition. In *AAAI Workshop on Modeling Others from Observations*, 2006.

[20] Jehoshua Bruck and Vwani P Roychowdhury. On the number of spurious memories in the Hopfield model. *IEEE Trans. on Information Theory*, 36(2):393–397, 1990.

[21] Sandra Carberry. Techniques for plan recognition. *User Modeling and User-Adapted Interaction*, 11(1-2):31–48, 2001.

[22] Eugene Charniak and Robert P Goldman. A bayesian model of plan recognition. *Artificial Intelligence*, 64(1):53–79, 1993.

[23] Eugene Charniak and Solomon Eyal Shimony. Cost-based abduction and map explanation. *Artificial Intelligence*, 66(2):345–374, 1994.

[24] Antonio Chella, Haris Dindo, and Ignazio Infantino. A cognitive framework for imitation learning. *Robotics and Autonomous Systems*, 54(5):403–408, 2006.

[25] Antonio Chella, Haris Dindo, and Ignazio Infantino. Learning high-level tasks through imitation. In *Intl. Conf. on Intelligent Robots and Sys.*, pages 3648–3654. IEEE, 2006.

[26] Shui Nee Chow, John Mallet-Paret, and James A Yorke. Finding zeroes of maps: Homotopy methods that are constructive with probability one. *Math. of Comp.*, 32(143):887–899, 1978.

[27] Michael Jae-Yoon Chung, Marcellus Forbes, Maya Cakmak, and Rajesh PN Rao. Accelerating imitation learning through crowdsourcing. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4777–4784. IEEE, 2014.

[28] Michael Jae-Yoon Chung, Abram L Friesen, Dieter Fox, Andrew N Meltzoff, and Rajesh PN Rao. A Bayesian developmental approach to robotic goal-based imitation learning. *PloS one*, 10(11):e0141965, 2015.

[29] Gil Citro, Gordon Banks, and Gregory Cooper. Inkblot: a neurological diagnostic decision support system integrating causal and anatomical knowledge. *Artificial Intelligence in Medicine*, 10(3):257–267, 1997.

[30] Helen Couclelis. The abduction of geographic information science: transporting spatial reasoning to the realm of purpose and design. In *Spatial Information Theory*, pages 342–356. Springer, 2009.

[31] Daniel Crevier. *AI: The tumultuous history of the search for artificial intelligence*. Basic Books, 1993.

[32] Stanislas Dehaene and Jean-Pierre Changeux. A hierarchical neuronal network for planning behavior. *Proceedings of the National Academy of Sciences*, 94(24):13293–13298, 1997.

[33] Gerald DeJong and Raymond Mooney. Explanation-based learning: An alternative view. *Machine learning*, 1(2):145–176, 1986.

[34] Travis DeVautl, Seth Forrest, Ian Tanimoto, Terence Soule, and Robert Heckendorn. Learning from demonstration for distributed, encapsulated evolution of auton. outdoor robots. In *Proc. of the Comp. Pub. of the Annl. Conf. on Genetic and Evolutionary Computation*, pages 1381–1382. ACM, 2015.

[35] Haris Dindo, Antonio Chella, Giuseppe La Tona, Monica Vitali, Eric Nivel, and Kristinn R Thórisson. Learning problem solving skills from demonstration. In *Intl. Conf. on AGI*, pages 194–203. Springer, 2011.

[36] D. Doran, S. Schulz, and T. R. Besold. What Does Explainable AI Really Mean? A New Conceptualization of Perspectives. *arXiv preprint arXiv:1710.00794*, 2017.

[37] Kutluhan Erol, Dana S Nau, and Venkatramana S Subrahmanian. Complexity, decidability and undecidability results for domain-independent planning. *Artificial Intelligence*, 76(1):75–88, 1995.

[38] Jerome Feldman. The neural binding problem (s). *Cognitive neurodynamics*, 7(1):1–11, 2013.

[39] Richard E Fikes and Nils J Nilsson. Strips: A new approach to the application of theorem proving to problem solving. *Artificial intelligence*, 2(3):189–208, 1972.

[40] Chelsea Finn, Ian Goodfellow, and Sergey Levine. Unsupervised learning for physical interaction through video prediction. In *Advances in Neural Information Processing Systems*, pages 64–72, 2016.

[41] Tesca Fitzgerald, Ashok K Goel, and Andrea L Thomaz. Representing skill demonstrations for adaptation and transfer. In *AAAI Symposium on Knowledge, Skill, and Behavior Transfer in Autonomous Robots*, 2014.

[42] Robert W Floyd. Algorithm 97: shortest path. *Communications of the ACM*, 5(6):345, 1962.

[43] L. Fogassi, P.F. Ferrari, B. Gesierich, S. Rozzi, F. Chersi, and G. Rizzolatti. Parietal lobe: from action organization to intention understanding. *Science*, 308:662–667, 2005.

[44] Gerald B Folland. *Advanced Calculus*. Prentice Hall, 2002.

[45] Abram L Friesen and Rajesh PN Rao. Imitation learning with hierarchical actions. In *Intl. Conf. on Devel. and Learning*, pages 263–268. IEEE, 2010.

[46] Christopher W Geib and Robert P Goldman. A probabilistic plan recognition algorithm based on plan tree grammars. *AI*, 173(11):1101–1132, 2009.

[47] R. J. Gentili, H. Oh, D.-W. Huang, G. E. Katz, R. H. Miller, and J. A. Reggia. Towards a multi-level neural architecture that unifies self-intended and imitated arm reaching performance. In *Proc. of the* 36^{th} *Annu. Intl. Conf. of the IEEE Engineering in Medicine and Biol. Society*, pages 2537–2540. IEEE, 2014.

[48] Rodolphe J Gentili, Hyuk Oh, Javier Molina, James A Reggia, and José L Contreras-Vidal. Cortex inspired model for inverse kinematics computation for a humanoid robotic finger. In *2012 Annual International Conference of the IEEE Engineering in Medicine and Biology Society.*, volume 2012, page 3052. NIH Public Access, 2012.

[49] Malik Ghallab, Dana Nau, and Paolo Traverso. *Automated Planning: Theory & Practice*. Elsevier, 2004.

[50] Malik Ghallab, Dana Nau, and Paolo Traverso. *Automated Planning and Acting*. Cambridge University Press, 2016.

[51] Alex Graves, Greg Wayne, and Ivo Danihelka. Neural Turing machines. *arXiv preprint arXiv:1410.5401*, 2014.

[52] Alex Graves, Greg Wayne, Malcolm Reynolds, Tim Harley, Ivo Danihelka, Agnieszka Grabska-Barwińska, Sergio Gómez Colmenarejo, Edward Grefenstette, Tiago Ramalho, John Agapiou, et al. Hybrid computing using a neural network with dynamic external memory. *Nature*, 538(7626):471–476, 2016.

[53] P. Haikonen. *The Cognitive Approach to Conscious Machines*. Imprint Academic, 2003.

[54] Alex Hern. Stephen Hawking: AI will be 'either best or worst thing' for humanity. *The Guardian*, October 2016.

[55] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[56] Chad Hogg, Ugur Kuter, and Hector Munoz-Avila. Learning methods to generate good plans. In *AAAI*, 2010.

[57] Jun Hong. Goal recognition through goal graph analysis. *Journal of Artificial Intelligence Research*, 15:1–30, 2001.

[58] John J Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proc. of the Natl. Acad. of Sci.*, 79(8):2554–2558, 1982.

[59] John J Hopfield and David W Tank. Neural comp. of decisions in optimization problems. *Biol. Cybernetics*, 52(3):141–152, 1985.

[60] Osamu Hoshino, Noriaki Usuba, Yoshiki Kashimori, and Takeshi Kambara. Role of itinerancy among attractors as dynamical map in distributed coding scheme. *Neural Networks*, 10(8):1375–1390, 1997.

[61] D.-W. Huang, G. E. Katz, J. D. Langsfeld, R. J. Gentili, and J. A. Reggia. A virtual demonstrator environment for robot imitation learning. In *IEEE Intl. Conf. on Technologies for Practical Robot Applications (TePRA)*, pages 1–6. IEEE, 2015.

[62] D.-W. Huang, G. E. Katz, J. D. Langsfeld, H. Oh, R. J. Gentili, and J. A. Reggia. An object-centric paradigm for robot programming by demonstration. In *Schmorrow, D. D., Fidopiastis, M. C. (eds.) Foundations of Augmented Cognition 2015 LNCS*, volume 9183, pages 745–756. Springer Intl. Publishing, 2015.

[63] Di-Wei Huang, Rodolphe J Gentili, Garrett E Katz, and James A Reggia. A limit-cycle self-organizing map architecture for stable arm control. *Neural Networks*, 85:165–181, 2017.

[64] Di-Wei Huang, Garrett E Katz, Joshua D Langsfeld, Hyuk Oh, Rodolphe J Gentili, and James A Reggia. An object-centric paradigm for robot programming by demonstration. In *Intl. Conf. on Augmented Cog.*, pages 745–756. Springer, 2015.

[65] M. Iacoboni, I. Molnar-Szakacs, V. Gallese, G. Buccino, J.C. Mazziotta, and G. Rizzolatti. Grasping the intentions of others with ones own mirror neuron system. *PLoS Biol.*, 3(e79), 2005.

[66] Bart Jansen and Tony Belpaeme. A computational model of intention reading in imitation. *Robotics and Auton. Sys.*, 54(5):394–402, 2006.

[67] Garrett E Katz, Di-Wei Huang, Rodolphe Gentili, and James Reggia. Imitation learning as cause-effect reasoning. In *Intl. Conf. on AGI*, pages 64–73. Springer, 2016.

[68] Garrett E Katz, Di-Wei Huang, Rodolphe Gentili, and James Reggia. An empirical characterization of parsimonious intention inference for cognitive-level imitation learning. In *Proc. of the Intl. Conf. on A. I.*, pages 83–89. CSREA Press, 2017.

[69] Garrett E Katz, Di-Wei Huang, Theresa Huage, Rodolphe Gentili, and James Reggia. A novel parsimonious cause-effect reasoning algorithm for robot imitation and plan recognition. *IEEE Trans. on Cog. and Devel. Sys.*, 2017.

[70] Garrett E Katz and James A Reggia. Identifying fixed points in recurrent neural networks using directional fibers: Supplemental material on theoretical results and practical aspects of numerical traversal. Technical Report CS-TR-5051, Univ. of MD, College Park, December 2016.

[71] Garrett E Katz and James A Reggia. Using directional fibers to locate fixed points of recurrent neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 2017. (accepted).

[72] Henry A Kautz and James F Allen. Generalized plan recognition. *AAAI*, 86(3237):5, 1986.

[73] R Baker Kearfott. *Rigorous global search: continuous problems*, volume 13. Springer Science & Business Media, 2013.

[74] Steven G Krantz and Harold R Parks. *The implicit function theorem: history, theory, and applications*. Springer Sci. & Business Media, 2012.

[75] John E Laird. Extending the SOAR cognitive architecture. *Frontiers in Artificial Intelligence and Applications*, 171:224, 2008.

[76] Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. *arXiv preprint arXiv:1504.00702*, 2015.

[77] Sergey Levine, Peter Pastor, Alex Krizhevsky, Julian Ibarz, and Deirdre Quillen. Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection. *The International Journal of Robotics Research*, 2016.

[78] Nan Li, Subbarao Kambhampati, and Sungwook Yoon. Learning probabilistic hierarchical task networks to capture user preferences. In *International Joint Conference on Artificial Intelligence*, 2009.

[79] James MacGlashan and Michael L Littman. Between imitation and intention learning. In *IJCAI*, pages 3692–3698, 2015.

[80] Edwin Mandfield. The diffusion of industrial robots in Japan and the United States. *Research Policy*, 18(4):183–192, 1989.

[81] Valerio Mante, David Sussillo, Krishna V Shenoy, and William T Newsome. Context-dependent computation by recurrent dynamics in prefrontal cortex. *Nature*, 503(7474):78–84, 2013.

[82] Ben Leon Meadows, Pat Langley, and Miranda Jane Emery. Seeing beyond shadows: Incremental abductive reasoning for plan understanding. In *AAAI Workshop: Plan, Activity, and Intent Recognition*, volume 13, page 13, 2013.

[83] A. Meltzoff and A. Moore. Imitation of facial and manual gestures by human neonates. *Science*, 198(4312):75–78, 1977.

[84] Tom M Mitchell, Richard M Keller, and Smadar T Kedar-Cabelli. Explanation-based generalization: A unifying view. *Machine learning*, 1(1):47–80, 1986.

[85] Anahita Mohseni-Kabir, Charles Rich, Sonia Chernova, Candace L Sidner, and Daniel Miller. Interactive hierarchical task learning from a single demonstration. In *Proceedings of the Tenth Annual ACM/IEEE International Conference on Human-Robot Interaction*, pages 205–212. ACM, 2015.

[86] David Z. Morris. Elon Musk says artificial intelligence is the 'greatest risk we face as a civilization'. *Fortune*, July 2017.

[87] Igor Mozetič. Hierarchical model-based diagnosis. *International Journal of Man-Machine Studies*, 35(3):329–362, 1991.

[88] Dana Nau, T-C Au, Okhtay Ilghami, Ugur Kuter, Dan Wu, Fusun Yaman, Héctor Muñoz-Avila, and J William Murdock. Applications of SHOP and SHOP2. *IEEE Intelligent Systems*, 20(2):34–41, 2005.

[89] Dana S Nau, Malik Ghallab, and Paolo Traverso. Blended planning and acting: Preliminary approach, research challenges. In *Proceedings Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015.

[90] Arvind Neelakantan, Quoc V Le, and Ilya Sutskever. Neural programmer: Inducing latent programs with gradient descent. *arXiv preprint arXiv:1511.04834*, 2015.

[91] João Pedro Neto, Hava T Siegelmann, and J Félix Costa. Symbolic processing in neural networks. *Journal of the Brazilian Computer Society*, 8(3):58–70, 2003.

[92] H. Oh. A multiple representations model of the human mirror neuron system for learned action imitation (doctoral dissertation). 2015. University of Maryland, College Park.

[93] H. Oh, R. J. Gentili, J. A. Reggia, and J. L. Contreras-Vidal. Modeling of visuospatial perspectives processing and modulation of the fronto-parietal network activity during action imitation. In *2012 Annu. Intl. Conf. of the IEEE Engineering in Medicine and Biol. Society*, pages 2551–2554. IEEE, 2012.

[94] Erhan Oztop, Daniel Wolpert, and Mitsuo Kawato. Mental state inference using visual control parameters. *Cog. Brain Research*, 22(2):129–151, 2005.

[95] Charles Sanders Peirce. *The Essential Peirce: Selected Philosophical Writings*, volume 2. Indiana University Press, 1998.

[96] Yun Peng and James A Reggia. Diagnostic problem-solving with causal chaining. *International Journal of Intelligent Systems*, 2(3):265–302, 1987.

[97] Yun Peng and James A Reggia. A probabilistic causal model for diagnostic problem solving - part i: Integrating symbolic causal inference with numeric probabilistic inference. *IEEE Transactions on Systems, Man and Cybernetics*, 17(2):146–162, 1987.

[98] Yun Peng, James A Reggia, et al. *Abductive Inference Models for Diagnostic Problem-Solving*. Springer-Verlag New York, 1990.

[99] Gualtiero Piccinini. Some neural networks compute, others dont. *Neural networks*, 21(2):311–321, 2008.

[100] David Poole. Explanation and prediction: an architecture for default and abductive reasoning. *Computational Intelligence*, 5(2):97–110, 1989.

[101] Robert F Port and Timothy Van Gelder. *Mind as Motion: Explorations in the Dynamics of Cognition*. MIT press, 1995.

[102] C Puente, MD López, J Rodrigo, and JA Olivas. Weighted graphs to model causality. In *Proc. of the Intl. Conf. on A. I.*, pages 297–301. CSREA Press, 2017.

[103] Mikhail I Rabinovich, Ramón Huerta, Pablo Varona, and Valentin S Afraimovich. Transient cognitive dynamics, metastability, and decision making. *PLoS Comput. Biol.*, 4(5):e1000072, 2008.

[104] S. Raghavan and R. J. Mooney. Bayesian abductive logic programs. In *Proc. of the* 10^{th} *AAAI Workshop on Statistical Relational A. I.*, pages 82–87, 2010.

[105] Scott Reed and Nando de Freitas. Neural programmer-interpreters. In *4th International Conference on Learning Representations (ICLR)*, 2016.

[106] James A Reggia, Di-Wei Huang, and Garrett E Katz. Exploring the computational explanatory gap. *Philosophies*, 2(1):5, 2017.

[107] James A Reggia, Garrett E Katz, and Di-Wei Huang. What are the computational correlates of consciousness? *Biologically Inspired Cognitive Architectures*, 17:101–113, 2016.

[108] James A Reggia, Derek Monner, and Jared Sylvester. The computational explanatory gap. *Journal of Consciousness Studies*, 21(9-10):153–178, 2014.

[109] Raymond Reiter. The frame problem in the situation calculus: A simple solution (sometimes) and a completeness result for goal regression. *Artificial intelligence and mathematical theory of computation: papers in honor of John McCarthy*, 27:359–380, 1991.

[110] Mohammad Taghi Saffar, Mircea Nicolescu, Monica Nicolescu, and Banafsheh Rekabdar. Intent understanding using an activation spreading architecture. *Robotics*, 4(3):284–315, 2015.

[111] Arthur Sard et al. The measure of the critical values of differentiable maps. *Bull. Amer. Math. Soc*, 48(12):883–890, 1942.

[112] Paulo Shakarian and VS Subrahmanian. *Geospatial Abduction: Principles and Practice*. Springer, 2011.

[113] Murray Shanahan. Robotics and the common sense informatic situation. In *ECAI*, pages 684–688. PITMAN, 1996.

[114] Vikas Shivashankar, Ron Alford, Ugur Kuter, and Dana Nau. The GoDeL planning system: a more perfect union of domain-independent and hierarchical planning. In *Proceedings of the Twenty-Third international joint conference on Artificial Intelligence*, pages 2380–2386. AAAI Press, 2013.

[115] Vikas Shivashankar, Krishnanand N Kaipa, Dana S Nau, and Satyandra K Gupta. Towards integrating hierarchical goal networks and motion planners to support planning for human-robot teams. 2014. [Online; `http://www.cs.umd.edu/~nau/papers/shivashankar2014towards.pdf`; accessed 8-July-2014].

[116] Vikas Shivashankar, Ugur Kuter, Dana Nau, and Ron Alford. A hierarchical goal-based formalism and algorithm for single-agent planning. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems*, volume 2, pages 981–988. International Foundation for Autonomous Agents and Multiagent Systems, 2012.

[117] Patrick Simen, MK van Vugt, Fuat Balci, David Freestone, and Thad Polk. Toward an analog neural substrate for production systems. In *Proceedings of the International Conference in Cognitive Modeling*. Citeseer, 2010.

[118] Parag Singla and Raymond J Mooney. Abductive markov logic for plan recognition. In *AAAI*, 2011.

[119] Daniel Soudry and Yair Carmon. No bad local minima: Data independent training error guarantees for multilayer neural networks. *arXiv preprint arXiv:1605.08361*, 2016.

[120] David Sussillo and Omri Barak. Opening the black box: low-dimensional dynamics in high-dimensional recurrent neural networks. *Neural computation*, 25(3):626–649, 2013.

[121] Jared Sylvester. Neurocomputational methods for autonomous cognitive control (doctoral dissertation). 2014. University of Maryland, College Park.

[122] Jared Sylvester and James Reggia. Engineering neural systems for high-level problem solving. *Neural Networks*, 79:37–52, 2016.

[123] J.C. Sylvester, J.A. Reggia, S.A. Weems, and M.F. Bunting. Controlling working memory with learned instructions. *Neural Networks*, 41(0):23–38, 2013. Special Issue on Autonomous Learning.

[124] Austin Tate. Generating project networks. In *Proceedings of the 5th international joint conference on Artificial intelligence-Volume 2*, pages 888–893. Morgan Kaufmann Publishers Inc., 1977.

[125] Michael Taylor. The inverse function theorem via Newton's method. `http://www.unc.edu/math/Faculty/met/invfn.pdf`. Accessed: 2016-11-21.

[126] D. Tecuci and B. Porter. Memory based goal schema recognition. In *Proc. of the* 22^{nd} *Florida A.I. Research Society Conf.*, 2009.

[127] David S Touretzky. Boltzcons: Dynamic symbol structures in a connectionist network. *Artificial Intelligence*, 46(1):5–46, 1990.

[128] David S Touretzky and Geoffrey E Hinton. A distributed connectionist production system. *Cognitive Science*, 12(3):423–466, 1988.

[129] Reiko Tsuneto, Kutluhan Erol, James Hendler, and Dana Nau. Commitment strategies in hierarchical task network planning. In *NCAI*, pages 536–542, 1996.

[130] Stanley Tuhrim, James Reggia, and Sharon Goodall. An experimental study of criteria for hypothesis plausibility. *Jrnl. of Experimental & Theoretical AI*, 3(2):129–144, 1991.

[131] Michael Van Lent, William Fisher, and Michael Mancuso. An explainable artificial intelligence system for small-unit tactical behavior. In *Proceedings of the National Conference on Artificial Intelligence*, pages 900–907. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2004.

[132] Pablo Varona and Mikhail I Rabinovich. Hierarchical dynamics of informational patterns and decision-making. *Proc. R. Soc. B*, 283(1832), 2016.

[133] Deepak Verma and Rajesh PN Rao. Imitation learning using graphical models. In *European Conf. on Machine Learning*, pages 757–764. Springer, 2007.

[134] Marc B Vilain. Getting serious about parsing plans. In *AAAI*, pages 190–197, 1990.

[135] Jacques Wainer and Alexandre de Melo Rezende. A temporal extension to the parsimonious covering theory. *Artificial Intelligence in Medicine*, 10(3):235–255, 1997.

[136] H. Wang, Q. Li, J. Yoo, and Y. Choe. Dynamical analysis of recurrent neural circuits in articulated limb controllers for tool use. In *2016 Intl. Joint Conf. on Neural Networks*, pages 4339–4345, July 2016.

[137] Terry Winograd. Procedures as a representation for data in a computer program for understanding natural language. Technical report, Massachusetts Institute of Technology, 1971.

[138] Alexander N Wittig. *Rigorous High-Precision Enclosures of Fixed Points and Their Invariant Manifolds*. PhD thesis, Michigan State University, 2012.

[139] Yan Wu, Yanyu Su, and Yiannis Demiris. A morphable template framework for robot learning by demonstration. *RAS*, 62(10):1517–1530, 2014.

[140] Wang Xinghua. Convergence of Newton's method and inverse function theorem in Banach space. *Mathematics of Computation of the American Mathematical Society*, 68(225):169–186, 1999.

[141] Qiang Yang, Rong Pan, and Sinno Jialin Pan. Learning recursive HTN-method structures for planning. In *Proc. of the ICAPS Workshop on AI Planning and Learning*, 2007.

[142] Yezhou Yang, Anupam Guha, Cornelia Fermüller, and Yiannis Aloimonos. A cognitive system for understanding human manipulation actions. *Advances in Cognitive Sysytems*, 3:67–86, 2014.

[143] Yezhou Yang, Yi Li, Cornelia Fermüller, and Yiannis Aloimonos. Robot learning manipulation action plans by "watching" unconstrained videos from the world wide web. In *AAAI*, pages 3686–3693, 2015.

[144] Konstantinos Zampogiannis, Yezhou Yang, Cornelia Fermüller, and Yiannis Aloimonos. Learning the spatial semantics of manipulation actions through preposition grounding. In *Robotics and Automation (ICRA), 2015 IEEE International Conference on*, pages 1389–1396. IEEE, 2015.

[145] Zhigang Zeng and Jun Wang. Multiperiodicity of discrete-time delayed neural networks evoked by periodic external inputs. *IEEE Trans. on Neural Networks*, 17(5):1141–1151, 2006.

[146] Huaguang Zhang, Zhanshan Wang, and Derong Liu. A comprehensive review of stability analysis of continuous-time recurrent neural networks. *IEEE Trans. on Neural Networks and Learning Sys.*, 25(7):1229–1262, 2014.