UC San Diego

UC San Diego Electronic Theses and Dissertations

Title

Derivation and Analysis of Distributed Computing Algorithms in Biological Systems

Permalink

https://escholarship.org/uc/item/03v7w5sf

Author

Chandrasekhar, Arjun

Publication Date

2020

Peer reviewed|Thesis/dissertation

eScholarship.org Powered by the California Digital Library
University of California

UNIVERSTITY OF CALIFORNIA SAN DIEGO

Derivation and Analysis of Distributed Computing Algorithms in Biological Systems

A dissertation submitted in partial satisfaction of the requirements for the degree Doctor of Philosophy

in

Bioinformatics and Systems Biology

by

Arjun Chandrasekhar

Committee in Charge

Professor Saket Navlakha, Chair
Professor Deborah Gordon, Co-Chair
Professor Glenn Tesler, Co-Chair
Professor David Kleinfeld
Professor Jason Schweinsberg

2020

The dissertation of Arjun Chandrasekhar is approved, and it is acceptable in quality and form for publication on microfilm and electronically:

Co-Chair

Co-Chair

Chair

University of California San Diego

2020

EPIGRAPH

You get three ants together, they can't do dick. You get 300 million of them, they can build a cathedral.

- Annie Savoy, *Bull Durham*

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

I would like to acknowledge my mother, my father, my brother, and my girlfriend for their unwavering love and support throughout graduate school. This document would not exist without them.

I would like to acknowledge my advisor and committee chair, Dr. Saket Navlakha for his support and mentorship. His clear and patient guidance has proven invaluable to my development as a student and a person. I would also like to acknowledge the rest of the Navlakha lab for their support and feedback.

I would also like to acknowledge the other four members of my committee. In particular, I would like to acknowledge Dr. Deborah Gordon for mentoring me, both in the lab and in the field, as if I were one of her own students. Her involvement in my development has consistently gone above and beyond what is typical for a collaborator.

My work is funded by an NIH training grant, as well as Chapman Foundations Management, LLC.

Chapter 1, in full, is a reprint of material as it appears Scientific Reports in 2018. Chandrasekhar, Arjun; Gordon, Deborah M; Navlakha, Saket, Nature Research, 2018. The dissertation author was the primary investigator and author of this paper.

Chapter 2, in full, is currently in review. Chandrasekhar, Arjun; Marshall, James; Austin, Cortnea; Navlakha, Saket; Gordon, Deborah M, bioRxiv doi: 10.1101/714410. The dissertation author was the primary investigator and author of this paper.

Chapter 3, in full, is a reprint of material as it appears in Proceedings of the Royal Society B. Chandrasekhar, Arjun; Navlakha, Saket, The Royal Society, 2019. The dissertation author was the primary investigator and author of this paper.

VITA

2012-14	Teaching Assistant, Caltech
2014	Bachelor of Science, Caltech
2014-19	Research Assistant, University of California San Diego
2015-16	Teaching Assistant, University of California San Diego
2019	Adjunct Lecturer, University of San Diego
2020	Doctor of Philosophy, University of California San Diego

PUBLICATIONS

Chandrasekhar, A., Gordon, D. M., and Navlakha, S. (2018). *A distributed algorithm to maintain and repair the trail networks of arboreal ants*. Scientific reports, 8(1), 9297.

Chandrasekhar, A., and Navlakha, S. (2019). *Neural arbors are Pareto optimal*. Proceedings of the Royal Society B, 286(1902), 20182727.

Chandrasekhar, A., Marshall, J., Austin, C., Navlakha, S, and Gordon, D.M. (2019). *Better tired than lost: turtle ant trail networks favor coherence over shortest paths*. bioRxiv doi: 10.1101/714410

Conn, A., Chandrasekhar, A., Rongen, M. V., Leyser, O., Chory, J., and Navlakha, S. (2019). *Network trade-offs and homeostasis in Arabidopsis shoot architectures*. PLoS computational biology, 15(9), e1007325.

Chandrasekhar, A. *Derivation and Analysis of Distributed Computing Algorithms in Biological Systems* (2020) (Dissertation).

ABSTRACT OF THE DISSERTATION

Derivation and Analysis of Distributed Computing Algorithms in Biological Systems

by

Arjun Chandrasekhar

Doctor of Philosophy in Bioinformatics and Systems Biology

University of California San Diego, 2020

Professor Saket Navlakha, Chair
Professor Deborah Gordon, Co-Chair
Professor Glenn Tesler, Co-Chair

There are numerous examples of biological systems outperforming human-designed algorithms at engineering tasks. Rather than relying on one central controller, these systems rely on distributed agents following simple local interaction rules that yield complex collective behavior. My research studies these biological systems systems in order to reverse-engineer optimization algorithms, with a focus on designing robust, efficient routing networks.

Repairing broken links is a fundamental problem for human-designed routing networks, as well as trail networks used by ants to travel between nests and food sources. Many ant species have evolved methods to solve the repair problem under more restrictive constraints than those faced by human engineers. Arboreal turtle ants are especially interesting, because their movements are constrained by the branches in the vegetation. I defined a mathematical model for the unique repair problem faced by turtle ants. I demonstrated techniques for using turtle ant behavioral data to derive a distributed repair algorithm and infer

critical features of turtle ant behavior. The algorithm is biologically realistic, scalable, and robust to a variety of topologies. I then analyzed data on trail networks formed by turtle ants over several days to conclude that turtle ant trail networks eschew shorter branches in favor of nodes whose physical orientations allow for quicker pheromone reinforcement.

Human-designed routing networks often seek to maximize efficiency while minimizing cost. These objectives often conflict, which necessitates manage trade-offs. Neural arbors face a similar challenge of conducting signals quickly while conserving material. I defined a mathematical framework for the multi-objective optimization problem that neural arbors face. I showed that neural arbors are significantly better at optimizing trade-offs between material cost and signal delay than would be expected by chance. I showed how this framework can be used to predict an arbor's structure based on its function. My work confirms the value of using neural arbors as inspiration for network design algorithms, and identifies promising new avenues for neuroanatomy research.

My research highlights the symbiotic relationships between biology and computer science: collecting and analyzing behavioral data can inform algorithm design, while algorithmic thinking can elucidating novel behavioral patterns.

Introduction

Algorithms in nature

There are numerous examples of natural systems that must overcome challenges faced by engineering systems [1]. Ant colonies must navigate between nests and food sources [2, 3, 4]; bee colonies must find fruitful foraging sites [5]; birds flocks must organize into advantageous formations [6]; neural arbors must grow in search of synaptic connections [7]. These systems must solve problems under restrictive circumstances. The individual agents in these systems are unlikely to have enough memory to scan, remember, and process all of the available information. It is also unrealistic for any one agent to direct the behavior of the rest of the group.

Many well-known combinatorial optimization problems seek to optimize objectives such as maximal payoff or minimal cost [8], or some combination thereof [9]. In this way, engineered and biological systems both benefit from increased efficiency. Both human-designed wireless communication networks [10] and ant-designed foraging networks [4] benefit from taking less time to transport as many resources as possible. Both immune systems [11] and swarm robots [12] benefit from taking less to scour in search of a target. Both Google image search [13] and the fly olfactory system [14] benefit from identifying images or odors with high accuracy. Thus, selective pressure often acts in favor of these objectives. However, these systems often differ in how strongly they prioritize robustness. Biological systems cannot know what changes will occur to the problem they are trying to solve, the environment in which they solve it, or the adversarial forces that try to impede them. Compared to humans, ecological and molecular systems do not have the same ability to manipulate these variables. Moreover, biological systems are liable to lose components of the system at any time, e.g., ants can get lost, bees can die, and T-cells can dysfunction. All of this means biological systems require flexible methods, perhaps at the cost of finding optimal solutions [15, 16]. While engineering systems often include robustness protocols, such as fault-tolerance [17], this is usually a secondary goal.

What is perhaps most fascinating about biological systems is the fact that they consistently match,

and sometimes exceed, the performance of human-designed systems. For example, long before humans created TCP/IP protocol to manage internet traffic, harvester ants evolved a similar algorithm to modulate the rate of outgoing foragers to match the availability of food [18]. Argentine ants create trail networks that are remarkably similar to Steiner trees [19]. The olfactory circuit in flies classifies odors based on previously encountered odors, and it does so more efficiently than human algorithms designed for similar problems [14]. Studies show that natural immune systems achieve scale-invariant responses to infections; that is, the time to neutralize a pathogen does not systematically increase with body size [20].

Notably, many biological systems remain astoundingly robust to disruptions [15, 16]. Bacteria adapt colony growth patterns to different media [21], and the immune response is legendary for its antibody diversity in response to invasive bacteria [11]; Argentine ants are equal-opportunity invaders, adapting their foraging behavior to a variety of environments [22]; fish schools show strong abilities to escape from traps or other uncomfortable spaces [23].

The reason why natural systems solve problems so well is that natural selection is ruthless, and necessity is the mother of invention. Species go extinct if they fail to evolve a method to solve challenges within the existing constraints. Moreover, unlike engineered systems, biological systems have evolved and refined the methods they use on evolutionary timescales. This combination of selective pressure and time often yields an effective and practical method. As a result, if we could design a method identical to the methods evolved by species fighting for their survival, we would end up reverse-engineering useful, and highly robust, methods for the engineering problems that humans face. We may also understand biological systems better by using engineering principles to explain their behavior.

This insight was put to use by inventor Georges de Mestral when he went for a walk in 1941. Initially frustrated by the burrs that clung to his dog and his legs, he soon became curious about whether their adhesive properties could be harnessed in a useful way. He soon created two straps, one with hooks and one with loops, that held firm when pressed together. So was born two things: the product of velcro [24], and the research field of biologically inspired materials [25]. Eventually, computer scientists began following in the footsteps of Georges de Mestral by looking towards nature for solutions. Rather than trying to reverse engineer materials, computer scientists seek to reverse engineer algorithms used by biological species to solve computational tasks; these algorithms are sometimes called *algorithms in nature* [26, 27, 28].

Distributed computation

Reasoning about the computation performed by biological systems requires carefully defining the concept of computation. Many classical computer science algorithms rely on *centralized* computing: a

single central processor accesses the full input, keeps track of all intermediate computations, and produces an output. Centralized computing is popular in part because it is intuitive to reason about the capabilities of centralized algorithms; there is a vast array of literature on decidability and tractability results for centralized models of computation. The other dominant paradigm in computer science is *distributed* computing. In the distributed paradigm, several agents work collectively to solve a problem. Each agent may only access a portion of the input at a given time, and may perform limited local communication with nearby agents. In principle each agent may carry out any computable function, but in practice each agent typically carries out a simple algorithm. The system produces an output in the form of agents achieving some desired configuration of states. The distributed paradigm of computing is often considered desirable for several reasons. First, distributed algorithms often have low overhead cost because each agent requires less computational power. Second, distributed algorithms are robust to changes to the input; because no agent needs access to the full input, the same algorithm that can generate a solution can easily repair a solution through simple local interactions involving only a fraction of the agents. Third, distributed algorithms are robust to the failures of a single agent, because the system is not dependent on a single controller. Fourth, distributed algorithms are easily scalable – it is often easier to add more agents than to update a single central controller.

The distributed paradigm is a more realistic model for how biological systems solve problems algorithmically. Each individual only needs to be responsible for performing a simple algorithm based on local information and communication with nearby individuals. Despite relying on local simplicity, these systems produce complex collective behavior and elegant emergent properties that allows them to overcome the challenges they face: ants successfully form trail networks, bees find enough nectar to support the colony, birds assemble in a V-formation, and neurons form networks of arbors to carry out all the necessary nervous system functions. The goal is to design algorithms that mimic this behavior.

In the centralized paradigm, defining an algorithm is usually straightforward: following the Turing machine model, the machine starts with the full input encoded as a string on the tape; the tape head performs the read/write operations prescribed by the algorithm, and leaves the output as a string on the tape. Defining a distributed algorithm, especially one that attempts to mimic biology, involves defining the following components of the computation:

1. An appropriate model of computation, such as a finite automata or a Turing machine, which determines the operations each agent can and cannot perform.

2. An appropriate model of computation, such as message passing or shared memory, for how agents are allowed to communicate.

3. The computational problem the agents must solve, including precise definitions of the inputs and out-

puts: what information do agents receive, and what information does the system emit as output?

4. An algorithm that is feasible within the chosen model of computation, which each agent performs independently.

Significant work has gone into the first three components. Some popular biological distributed computing models include the following [29]: beeping [30], in which an agent may either emit a unary 'beep' signal, or determine if at least one of its neighbors beeped (but not both); the stone age model [31], in which agents are finite state machines that may sends and receives characters from some finite alphabet, but can only count up to a fixed number when counting how many times it has received each character; population protocols [32], in which pairs of interactions agents may freely send messages, but interactions are scheduled either randomly or by an adversary; and shared memory [33], in which agents read, write, and erase information from a shared memory source rather than communicating directly. After defining a suitable model of computation, the challenge is to design the right local interaction rules so that the agents collectively maneuver to solve the problem.

Nature-inspired metaheuristics

Drawing inspiration from nature has allowed computer scientists to develop several different *metaheuristics*, i.e., classes of algorithms that follow the same general framework for how agents receive input, make decisions, and communicate. Each metaheuristic observes a particularly innovative behavior used by some natural system, and adapts this to help distributed agents communicate and process input more efficiently; from here, different implementation choices, such as the computational power of individual agents or the local interaction rules used by each agent, lead to different variants. These metaheuristics demonstrate the ways that biological principles can improve optimization algorithms, but they often sacrifice biological realism to do so. This indicates that it is possible to design biological metaheuristics can be improved to use even fewer resources than they already do without sacrificing performance. This will lead to more scalable algorithms, and also make the algorithms more useful as tools for generating biological hypotheses and predictions.

Ant colony optimization

One of the pioneering entries into the field of natural computing was *ant colony optimization (ACO)* [34]. This algorithm takes inspiration from the pheromone-laying behavior of ants to solve combinatorial optimization problems. Initially, ACO was created to the travelling salesman problem (TSP). In this

problem, the input is a complete, weighted graph in which nodes represent cities, and edge weights represent the distance or time required to travel between a pair of cities. A TSP tour is a path that touches every city exactly once; the goal is to find the TSP tour that minimizes the sum of the lengths of all edges traversed. The distributed agents are ants that continuously perform a random walk on the graph. At each time step, ants move from one city to the next. Ants may lay pheromone on an edge, to signal to other ants to use that edge; pheromone levels decay over time. When an ant chooses its next edge, it may take into account the weight of each edge, as well as the current amount of pheromone on each edge. In principle, the ants may use any Turing-computable function to stochastically choose the next edge based on the edge weights and pheromone levels. The goal is to output an optimal TSP tour in the form of a graph that has high pheromone levels on edges that are part of the optimal tour, and low pheromone levels on all other edges.

In the first ACO algorithm, called *Ant System*, each ant uses a simple formula to compute the probability of taking each of the available edges [34]. Over time ants will deposit more pheromone on edges that are part of better tours, which in turn makes them more likely to take and reinforce better tours, thus creating a positive feedback cycle that leads to an optimal solution. In practice, Ant System is highly competitive with traditional heuristics on inputs up to certain sizes [35]. In subsequent years, several improvements have been made to ACO that involve allowing ants to use local search to improve their initial pheromone-based solutions [36]. Over time, ACO has been been generalized to solve other combinatorial optimization problems, by using pheromone to reinforce generalized solution components rather than edges in a graph [37].

Subsequent metaheuristics

ACO illustrates much of the power and appeal of using distributed computation, and of taking inspiration from nature. By borrowing the concept of trail pheromone from ants, ACO manages to greatly improve on traditional TSP heuristics. It is an elegant example of emergent properties: each ant performs a very simple algorithm, but collectively the colony approximates an NP-complete problem. The algorithm is scalable, because there is little overhead cost to adding more ants. Perhaps most exciting is that ACO also captures the robustness that is so appealing in biological systems. The algorithm can be run continuously, and respond to changes to the input in a robust manner: if an edge breaks, or a new city gets added, the algorithm can let the ants continue foraging and explore for a new and/or better path. The success of ACO has inspired similar heuristics inspired by social animals: Artificial Bee Colony, in which simulated bees 'forage' for potential solutions, and use a waggle dance to recruit other bees [38]; Particle Swarm Optimization in which birds in a flock attempt to find a solution by 'flying' towards global (or local) minima on a hypersur-

face [39]; and krill heard, in which krill choose between moving towards food and following the movement of nearby krill [40].

In addition to social insects, the human body has inspired metaheuristics for optimization. Two notable examples include artificial immune systems [41], and artificial neural networks [42, 43]. Neural networks have become particularly popular in recent years for their demonstrated effectiveness in machine learning tasks such as natural language processing and image recognition [44]. The metaheuristic borrows from neuroscience by mimicking the process by which the brain sends and receives electrical signals through a network of interconnected neurons. In a typical neural network, there are several 'layers'of neurons. Each layer receives signals from neurons in the previous layer, does some computation with these inputs, and then 'fires' outputs to the next layer. The first layer receives the input, and the final layer produces a final output. For machine learning problems, the network is trained to extract patterns from a dataset of known examples, so that it can classify new examples as either having or not having a certain property (e.g., predicting whether a Netflix user will like a new show based on his/her past ratings).

Based on the known examples, the nodes in the network adjust the values of the parameters they use when processing input signals. The network is parameterized so that the final layer classifies as many known examples correctly as possible. This is done using the *backpropagation algorithm* [45]. The algorithm starts with random weights between layers, and iteratively updates the weights to reduce the difference between the correct outputs and network's actual outputs. This is done going through all input/output pairs, comparing the expected output to the network's actual output, and recursively propagating backwards through the layers to update weights based on the error in the network's output. At each layer, the network recursively computes the *gradient* of each weight from that layer, i.e., how output of the subsequent layers changes in response to changes to that weight. Based on the gradient, the network updates the weight to drive the network's output closer to the desired output. After propagating through all layers of the network, the algorithm moves on to the rest of the training examples and continues tuning weights.

The hope is that the weight parameters that best allow the network to classify known data will also work well for new data. In practice, neural networks tend to outperform other methods on well-known machine learning problems [46, 47], further confirming the power of nature-inspired algorithms.

Limits of nature-inspired metaheuristics

The algorithms described above are useful for solving optimization problems, and elegant examples of using biology to improve computer science. However, they fail to explain *how* the biological systems solve the problems that they do. They rely on computational sophistication that may not be biologically realistic. In

particular, while these algorithms use biologically realistic communication protocols (i.e., pheromones and local electrical signals) and individual decision-making rules, the models of computation these algorithms afford each agent may be too powerful. For example, ACO provides each ant with theoretically unbounded memory, even though it is generally assumed that social organisms such as ants and bees possess restricted memory. The ants in ACO can remember and retrace entire TSP tours; it is unlikely, or at least unknown, that ant species can do this. ACO also allows ants to adjust how much pheromone they lay based on how optimal a particular TSP tour is; ants are not known to dynamically modulate how much pheromone they deposit, and they certainly do not do so based on the utility of a particular path. Many neural network algorithms also operate in a way that, while algorithmically useful, is not likely to be representative of how actual neurons operate.

The fact that nature-inspired metaheuristics rely on overly powerful models of computation means it may be possible be reduce their computational overhead costs without sacrificing performance. By reverse-engineering biological algorithms in a way that is as faithful to the biology as possible, we can design algorithms that succeed with astonishingly few resources [27].

In addition to the potential for discovering efficient algorithms, there are additional reasons for emphasizing biologically accurate algorithms going forward. First, in the past it may have been difficult to determine what algorithmic techniques were and were not biologically realistic. However, recent hardware and software advances in areas such as next-generation sequencing [48], image processing [49], numerical computing [50], and animal tracking [51] allow us to more accurately quantify how natural systems can actually behave. Second, algorithmic thinking can inform biology. Casting a biological system as an algorithm provides a framework for understanding the section pressures affecting the system, and the algorithmic strategies the system evolved as a result. It also provides signposts for future biological research. The nature of an algorithmic model is such that it provides concrete, mathematically precise predictions to be tested. For example, if one were to discover a biologically realistic improvement to the biologically-inspired algorithm, that would also lead to a precise hypothesis about what properties may exist in the biological system. An elegant example of the value of algorithmic thinking for biologists is in the study of Argentine ants. The mathematical theory of Steiner trees provides ant biologists with precise language for describing the behavior and evolution of Argentine ant trail networks [19]. Modelling Argentine ant colonies as a distributed Steiner ant algorithm creates testable hypotheses, such as whether Argentine ants have evolved specific techniques, whether Argentine ants optimize other precisely defined design objectives, and whether trail networks resemble Steiner trees in other ant species.

A biologically faithful distributed algorithm

One exciting example of the power of biologically faithful algorithms involves recent research into connections between image recognition and the fly olfactory circuit. When flies encounter a new odor, they must classify the odor based on its similarity to previously encountered odors. This is similar to the engineering problem encountered in machine learning: given an image, classify it based on its similarity to images with known labels. Both problems can be classified as a version of *similarity search*: we start with a database of known images/odors (encoded as real-valued vectors), and we encounter a new image/odor. We seek to find the known image/odor which is most similar to the new vector (e.g., smallest vector distance). It is assumed that a simple one-by-one comparison is infeasible due to the number of known odors/images, as well as the sizes of the encodings of each odor/image. The traditional technique used in image search is locality sensitive hashing: compress the encodings in such a way that if two odors/images were originally similar, their compressed encodings are likely to be similar. This allows for significantly more efficient linear similarity search to take place [52].

The fly olfactory system takes the opposite approach: it *expands* the size of the odor encoding [14]. Each glomureli neuron receives the input and randomly fires a signal to a small number of Kenyon cell neurons. The Kenyon cells sum up the signal they receive, and all but the highest signals are inhibited. This creates a larger, but sparser (i.e., more 0's) encoding of the odor. This expanded encoding is then compared against known odors.

It may seem counter-intuitive that the fly would expand the size of the input. However, in practice the fly algorithm tends to outperform algorithms based on locality-sensitive hashing [14]. Upon further inspection, this is because by expanding the input, the fly algorithm is able to compare extremely sparse vectors, which can be done more efficiently than with small but dense vectors. Intuitively, expanding and sparsifying both the input vector and known vectors amplifies differences between odors, and more starkly clusters similar odors.

This work demonstrates the value of developing an algorithm based on faithfully and rigorously following the biology. Not only does an algorithm based on the fly olfactory system outperform human algorithms at a fundamental engineering problem; it inspires outside-the-box techniques, and illuminates new insights into the nature of the problem. Other recent examples of this include: a fly-inspired algorithm for constructing a maximal independent set [30], and an immune-system inspired algorithm to search a space for adversarial particles [53]. This work also highlights the value of algorithmic thinking to biology: it provides an explanation for the evolutionary path of the fly olfactory system, and allows biologists to ask precise questions about how olfaction may to work in flies and other animals.

Contributions of this dissertation

My research contributes to the field of algorithms in nature by studying how biological systems overcome challenges related to designing and maintaining routing networks. To do this, I apply algorithmic thinking to two biological systems that are particularly interesting from a computational perspective: turtle ants, and neural arbors. I demonstrate techniques for analyzing turtle ant data to identify useful network design principles and derive new algorithmic methods for repairing routing networks. I propose a mathematical model for the design trade-offs faced by neural arbors, and demonstrate that neural arbors optimize these trade-offs exceptionally well. I then use this model to identify key differences in how certain types of neurons prioritize different design design objectives.

Overall, my work illuminates the value in using biological systems as inspiration for new mathematics and computer science results. In doing so, my work suggests that common engineering problems can be solved using very restrictive models of computation, thus raising several open questions in the field of distributed computing algorithms. My work also draws fascinating connections between turtle ant behavior and random walk theory, suggesting that probability theorists and ant biologists may benefit from collaboration. Finally, I also show how algorithmic models of biological systems naturally produce precise, testable hypotheses about how biological systems behave. My work uses algorithmic thinking to produce specific hypotheses about turtle ant pheromone regulation and structure-function relationships in neural arbors.

Foraging algorithms in turtle ant colonies

My first two chapters describe research into an algorithm used by the genus *Cephalotes* (colloquially known as *turtle ants*), to solve the fundamental engineering problem of constructing, maintaining, and repairing a routing network. Most ant species, are terrestrial – that is, they forage on dry land, a continuous space. Species such as Argentine ants [19], pharaoh ants [54], and harvester ants [18, 55] have been the source of biologically-inspired foraging algorithms. Unlike these species, turtle ants are arboreal – they nest and forage in trees. The vegetation in which they forage can be represented as a discrete graph: branches correspond to edges, and overlapping branches form a node. This makes their foraging behavior exciting from a computer science perspective, because unlike for terrestrial species, turtle ant foraging is analogous to traversing a path within a graph. Turtle ants must form a foraging network connecting several nests and food sources [56, 3]. The vegetation in which the turtle ants forage is volatile. Connections between branches and vines can be fragile, and can be broken by disruptions such as intruders or inclement weather. Turtle ant foraging behavior must be robust to these situations. My first chapter shows how studying turtle ants allows us to derive an effective algorithm for repairing broken links in routing networks. I provide evidence

that turtle ants evolved an algorithm that is robust to a wide variety of situations, at the possible expense of being optimal for any one situation. I identify critical features for the success of the our ant-inspired algorithm, leading to hypotheses about whether turtle ants have evolved similar algorithmically advantageous behaviors. Overall, I demonstrate the potential for ant biology to lead us to new results in graph theory, distributed computing, and (perhaps more surprisingly), probability theory, while demonstrating the potential for computer science to lead us to testable hypotheses in ant biology.

Trade-offs in turtle ant trail networks

My second chapter begins to tackle the problem of understanding how turtle ants initially form the trail networks that they later repair. Traditionally, artificial routing networks are designed to manage trade-offs between efficiency and robustness [17]. Turtle ant trail networks must account for additional challenges, such as variation in the physical properties of branches, and the possibility of ants getting lost. I hypothesize three design principles that guide the structure of the trail networks used by turtle ants. I tested these three hypotheses by analyzing data on trail networks formed by three different turtle ant colonies over the course of several days. From the data, I concluded that turtle ants take advantage of physical variation in the vegetation to ensure that ants stay attached to the trail. This research further demonstrates the power of studying turtle ants to discover new insights into graph algorithms and distributed computing. Turtle ants use physical variation to enhance coordination among the colony, and human-designed systems such as robot swarms may benefit from doing the same. Furthermore, this chapter provides a clear example of how algorithmic thinking can inform biology. Viewing a turtle ant colony as a distributed routing network algorithm gives us precise language with which to make predictions about what behavior we expect to observe.

Pareto-optimal trade-offs in neural arbors

My third chapter continues my research into biologically generated routing networks. Neural arbors consist of a cell body (soma), and several wires called axons and dendrites that form synaptic connections with other neurons. The process by which neural arbors grow in search of synaptic connections can be viewed as a transport network design problem. One one hand, it is advantageous to design a network in which electrical signals can travel quickly between the soma and each synapse. However, the structure that minimizes travel time may be very expensive in terms of material cost. Neural arbors must manage this trade off. In most cases, there does not exist a tree that simultaneously optimizes both of these objectives; the best the arbor can hope to do is construct a *Pareto-optimal* arbor.

In multi-objective optimization, a solution is Pareto-optimal it is impossible to improve one aspect of

the solution without making another aspect worse. The solution is optimal in the sense that no other solution is superior for *all* of the objectives. The concept of Pareto-optimality was originally invented to analyze optimal solutions in economics [57] and engineering [58]. Recent research identifies how several biological species, such as rodents and finches, arrive at Pareto-optimal phenotypes through years of evolution [59].

I tested and validated the hypothesis that neural arbors have also evolved Pareto-optimal arbor morphology phenotypes. My work demonstrates that neural arbors manage to solve a fundamental engineering problem nearly as well as a centralized computing algorithm, likely without the computational sophistication necessary to perform centralized computation. This suggests that by studying growth rules for developing neurons, we may eventually derive a distributed algorithm to solve a network optimization problem with extremely limited computational resources. I use this model to propose hypotheses on possible structure-function relationships in different types of neurons, as well as similarities between plant and neural arbors, further confirming the value of using algorithmic models for biological systems.

Chapter 1

A Distributed Algorithm to Maintain and Repair the Trail Networks of Arboreal Ants

In this chapter, I explore connections between ant biology and network routing algorithms. I show how field observations from arboreal turtle ants can be used to reverse-engineer an algorithm for a fundamental routing network problem, and I show how an algorithmic model of turtle ants can identify crucial aspects of their foraging behavior that may be hard to detect otherwise.

Abstract

We study how the arboreal turtle ant (*Cephalotes goniodontus*) solves a fundamental computing problem: maintaining a trail network and finding alternative paths to route around broken links in the network. Turtle ants form a routing backbone of foraging trails linking several nests and temporary food sources. This species travels only in the trees, so their foraging trails are constrained to lie on a natural graph formed by overlapping branches and vines in the tangled canopy. Links between branches, however, can be ephemeral, easily destroyed by wind, rain, or animal movements. Here we report a biologically feasible distributed algorithm, parameterized using field data, that can plausibly describe how turtle ants maintain the routing backbone and find alternative paths to circumvent broken links in the backbone. We validate the ability of this probabilistic algorithm to circumvent simulated breaks in synthetic and real-world networks, and we derive an analytic explanation for why certain features are crucial to improve the algorithm's success. Our proposed algorithm uses fewer computational resources than common distributed graph search algorithms, and thus may be useful in other domains, such as for swarm computing or for coordinating molecular robots.

1.1. Introduction

Distributed algorithms allow a collection of agents to efficiently solve computational problems without centralized control [1]. Recent research has uncovered such algorithms implemented by many biological systems, including slime molds during foraging [60] and neural circuits during development [61]. Ants are a diverse taxon of more than 14,000 species that have also evolved distributed algorithms to establish trail networks [19]. Investigating the algorithms used by biological systems can reveal novel solutions to engineering problems [62, 61].

Here we present the first computational analysis, parameterized using data from field observations, of trail networks of an arboreal ant species. The arboreal turtle ant *C. goniodontus* nests and forages in the trees in the tropical dry forest of western Mexico [56]. Because the ants never leave the trees, their foraging trails are constrained by a natural graph: branches and vines form the edges in the graph, and junctions at overlapping branches form the nodes (Figure 1.1A–C). Each colony has several nests, located in dead tree branches, that are connected to each other in a circuit or network routing backbone [19, 63, 64]. Moving on the trails along this backbone, the ants distribute resources among the juveniles, workers, and reproductives in all of the nests, while additional temporary trails split from the backbone and lead to food sources. The backbone trail network can be large, often extending over 50 meters in circumference,

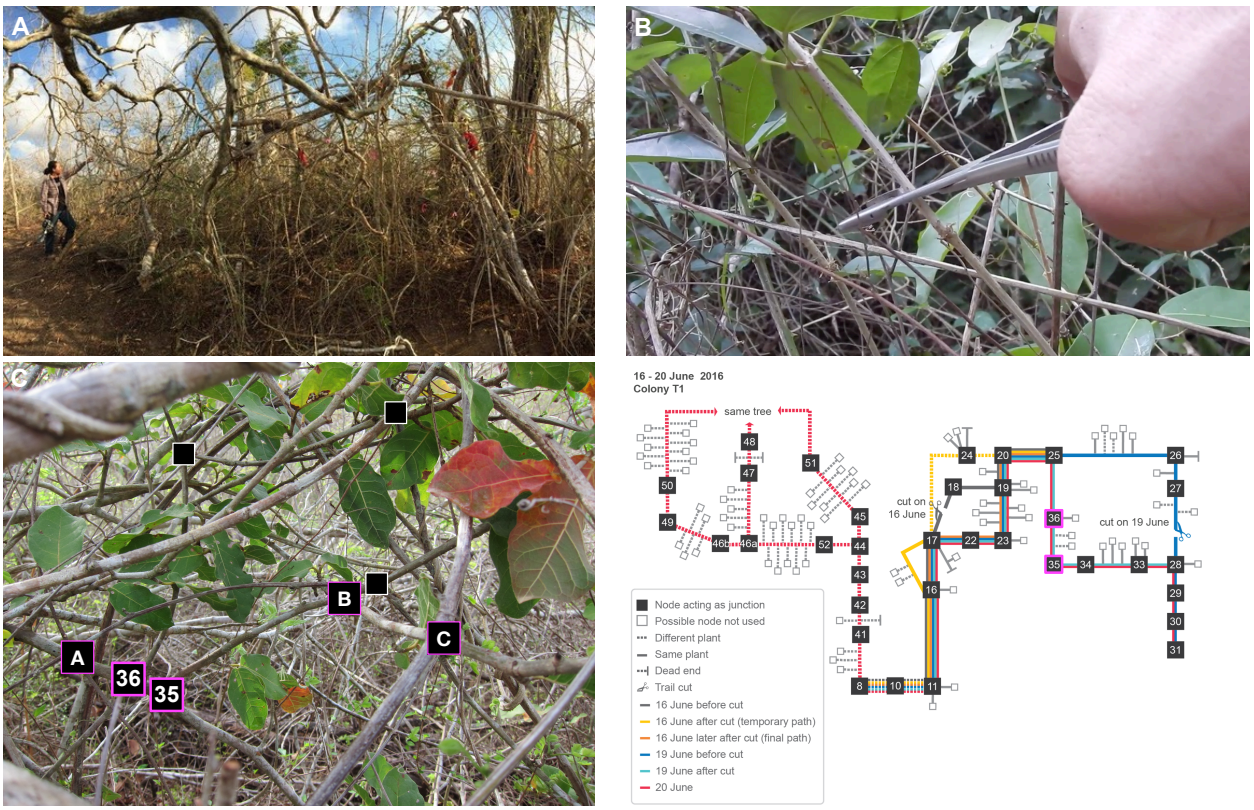


Figure 1.1: **Turtle ant habitat and trail network.** A) The photograph shows the highly tangled forest canopy in which turtle ants forage. B) Experiments were performed in which an edge in the path was cut, to observe how the ants respond and repair the break [3]. C) Modeling the trail network as a graph, with junctions as nodes and connecting branches and twigs as edges. The diagram on the right from [3] shows a detailed depiction of a large portion of the trail network. Each day's path is shown in a different color (see legend), and additional repair paths are shown in a distinct color. Solid lines connect two nodes that are on the same plant (e.g. node 36 and node A are on the same plant). Dashed lines connect two nodes that are on a different plant (e.g. nodes B and C are on different plants).

and encompassing numerous trees [56]. The ants use many junctions in dense vegetation, so trails can be tortuous; each meter of linear distance typically requires ants to traverse approximately 2–5 meters of vegetation [56]. The colony thus chooses paths in the network from a myriad of potential routes, dictated by the graph structure of the vegetation. Ants lay trail pheromone as they move along the edges, and ants use pheromone when choosing edges.

We present a distributed algorithm that can plausibly describe how turtle ants maintain and repair breaks to their routing backbone. Links between branches or vines can be ephemeral, often disrupted by wind, rain, or the movement of an animal through the vegetation. To re-establish connectivity of the routing backbone after a break, the ants must establish a new path that reconnects the two sides of the broken trail. This is an important problem in many network applications [65] and can be solved efficiently using numerous graph algorithms, such as Dijkstra's algorithm or the Bellman-Ford algorithm [66]. However, these classic

algorithms require significantly more computation and memory than is likely available to simple biological agents such as turtle ants, who regulate their behavior using local interactions rather than central control [67].

Repairing breaks requires overcoming three challenges. First, the ants must succeed in finding an alternative path by exploring new edges that currently have no pheromone and avoiding dead-ends in the network. One hypothesis for how this could be achieved is to first generate many candidate alternative paths and then converge to one or a few of them over time — a process we call "pruning". Such a strategy, also employed by slime molds [60] and neural circuits [61], has been shown to help quickly discover new paths in distributed settings, in which no agent is aware of the topology of the entire network. Second, all ants must converge to the same new path in order to optimally coordinate resource transport. Turtle ants travel in coherent trails that link nests and food sources [3]. After the vegetation supporting the trail is ruptured, the ants explore outside the previous path, and eventually commit again to a single path. Such convergence prevents ants from getting lost or separated from the rest of the colony. This is also an important goal in computer routing networks, where convergence to a single path ensures in-order delivery of data packets [68]. Third, it may be important to minimize the length of the new trail, which is also a standard measure of efficiency used when evaluating transport network design. However, data from field studies [56, 3] suggest that turtle ant paths are often not the shortest globally. It appears that the second objective, successful convergence, is more important than minimizing trail length, presumably because ants getting lost or separated has a higher cost than the energy spent in walking [3]. A common strategy to increase robustness to edge failures in a graph is to include loops in the path. Prior work [3] showed that loops do form in turtle ant trail networks; however, loops tend to get pruned over time, perhaps reducing the number of foragers needed to maintain the path.

The distributed algorithm used to maintain and repair trail networks must be robust across varying planar network topologies. The forest canopy is highly complex and dynamic, and it is unlikely that turtle ants use different algorithms to accommodate different network structures. Thus, we seek an algorithm that, while likely not "optimal" for any single planar topology, performs well across different planar topologies. The algorithm must also use very limited memory of individual agents, as ants are not capable of remembering many of their steps along the graph structure.

Our work seeks to uncover a biologically plausible distributed algorithm that corresponds with field observations of turtle ant behavior in response to experimentally-induced edge breaks (Figure 1.1B) [3]. We ask:

1. What model is most likely to explain how turtle ants at a node select which edge to traverse next?

2. How well can the algorithm repair broken trails in simulated breaks in synthetic and real-world network

topologies when parameterized by the most biologically realistic parameter values?

(a) Does the algorithm consistently converge to a single consensus path?

(b) Does the algorithm find short paths?

(c) Does bi-directional search, using ants from both sides of the broken path concurrently, improve the performance of the algorithm relative to uni-directional search?

(d) How does allowing an ant to avoid going back to the node it previously visited (backtracking), improve algorithm performance relative to performance when ants are not prevented from backtracking?

(e) How does the algorithm perform in the presence of multiple breaks?

(f) Can we provide any theoretical insights into why certain model features are necessary for any plausible turtle ant algorithm?

3. Can the same algorithm used to repair breaks also be used to keep the established routing backbone intact in the absence of a break?

4. Do turtle ants form multiple alternative paths and then prune some of them over time, as also observed in field studies?

A model that performs well on all of these criteria can be considered a plausible model of turtle ant behavior. Our main contribution is to identify several plausible non-linear models; we also show why one common linear model is likely implausible despite succeeding on some of the criteria listed above.

1.2. Related work

To our knowledge, this is the first computational analysis of trail networks of an arboreal ant species, whose movements are constrained to a discrete graph structure rather than continuous space. Compared to previous work, we attempt to solve the network repair problem using different constraints and fewer assumptions about the computational abilities of individual ants.

1.2.1 Species-specific modeling of ant behavior.

Previous studies of ant trail networks have largely examined species that forage on a continuous 2D surface [69], including Pharaoh's ants [54], Argentine ants [19, 70, 71], leaf-cutter ants [72], army ants [73], and red wood ants [74]. These species can define nodes and edges at any location on the surface, and form

trails using techniques such as random amplification [75, 76, 73], or using their own bodies to form living bridges [77]. Experimental work on these species sometimes uses discrete mazes or Y-junctions to impose a graph structure; however, these species have evolved to create graph structures in continuous space, not to solve problems on a fixed graph structure, as turtle ants have evolved to do. Turtle ant movements are entirely constrained by the vegetation in which they travel. They cannot form trails with nodes and edges at arbitrary locations; instead, they can use only the nodes and edges that are available to them.

Further, to provide the simplest possible algorithm that is biologically realistic, we assume that turtle ants use only one type of pheromone. There are more than 14,000 species of ants, and they differ in their use of chemical cues. For example, *Monomorium pharoensis* uses several different trail pheromones [78, 79, 80, 81, 82]. There is, however, no evidence that turtle ants lay more than one type of trail pheromone.

1.2.2 Ant colony optimization.

Models of ant colony optimization (ACO), first proposed in 1991, loosely mimic ant behavior to solve combinatorial optimization problems, such as the traveling salesman problem [34, 83, 37]. In ACO, individual ants each use a heuristic to construct candidate solutions, and then use pheromone to lead other ants towards higher quality solutions. Recent advances improve ACO through techniques such as local search [84], cunning ants [36], and iterated ants [85]. ACO, however, provides simulated ants more computational power than turtle actually ants possess; in particular, ACO-simulated ants have sufficient memory to remember, retrace, and reinforce entire paths or solutions, and they can choose how much pheromone to lay in retrospect, based on the optimality of the solution.

Prior work inspired by ants provides solutions to graph search problems [86, 87], such as the Hamiltonian path problem [88] or the Ants Nearby Treasure Search (ANTS) problem. The latter investigates how simulated ants collaboratively search the integer plane for a treasure source. These models afford the simulated ants various computational abilities, including searching exhaustively around a fixed radius [89], sending constant sized messages [90], or laying pheromone to mark an edge as explored [91]. Our work involves a similar model of distributed computation, but our problem requires not only that the ants find an alternative path to a nest (a “treasure”), but also that all the ants commit to using the same alternative path. This requires a fundamentally different strategy from that required for just one ant to find a treasure.

1.2.3 Graph algorithms and reinforced random walks.

Common algorithms used to solve the general network search and repair problem, including Dijkstra’s algorithm, breadth-first search, depth-first search, and A* search [66], all require substantial commu-

nication or memory complexity. For example, agents must maintain a large routing table, store and query a list of all previously visited nodes, or pre-compute a topology-dependent heuristic to compute node-to-node distances [92]. These abilities are all unlikely for turtle ants.

Distributed graph algorithms, in which nodes are treated as fixed agents capable of passing messages to neighbors, have also been proposed to find shortest paths in a graph [93, 94], to construct minimum spanning trees [95, 96], and to approximate various NP-hard problems [97, 30]. In contrast, our work uses a more restrictive model of distributed computation, where agents communicate only through pheromone which does not have a specific targeted recipient.

Finally, the limited assumptions about the memory of turtle ants invite comparison to a Markov process. Edge-reinforced random walks [98], first introduced by Diaconis and others [99, 100], proceed as follows: an agent, or random walker, traverses a graph by choosing amongst adjacent edges with a probability proportional to their edge weight; then the agent augments the weight (or pheromone) of each edge chosen. Our model expands edge-reinforced random walks in two ways: first, we allow many agents to walk the graph concurrently, and second, we decrease edge weights over time. Our work is similar to previous models of the gliding behavior of myxobacteria [101] that consider synchronous, node (rather than edge)-reinforced, random walks with decay. These models seek to determine when bacteria aggregate on adjacent points or instead walk freely on the grid. By contrast, here we ask whether the random walkers converge to a single consensus path between two points on the grid that are not necessarily adjacent.

1.3. Results

Our goals are to find an algorithm that can simultaneously explain the movement patterns of turtle ants on a trail network and that can effectively solve the network repair problem. First, we describe a computational framework for evaluating the collective response of turtle ants to edge ruptures. We evaluate the response according to three objectives: the likelihood of finding an alternative path to repair the trail, how well the ants converge to the same new trail, and the capacity to minimize the length of the trail. Second, we derive multiple candidate distributed algorithms for network repair. We parameterize each algorithm using data from field experiments to determine how the model would predict which edge a turtle ant would choose to traverse next from a node, given only local information about adjacent edges and their edge weights. Third, we analyze via simulation how our algorithms perform on different planar network topologies, including simulated breaks on a European road transport network.

1.3.1 A graph-theoretic framework for modeling network repair by turtle ants

We start with a weighted, undirected graph $G = (V, E, W)$, where V is the node set, E is the edge set, and W are the edge weights, as well as two nest nodes $u, v \in V$, and a path $P = (u, \ldots, v)$ from u to v with pheromone along each edge in P. Edges are undirected since turtle ants can walk in both directions over edges. Edge weights correspond to the amount of pheromone on the edges, which can change over time. We mimic a break in the path by removing some edge in P[1]. The challenge for the ants is to find an alternative path that reconnects u and v. The alternative path may build off the existing path, so that the initial and final path may share some edges.

Communication among simulated ants is limited to chemical signals, analogous to pheromone, left on edges traversed. Field observations are consistent with the assumption that, like Argentine ants [102], turtle ants lay trail pheromone continuously as they walk [56, 3]. Though this has not been observed directly, we hypothesize that there are certain exceptional situations in which turtle ants discontinue laying pheromone (Methods). Each ant at a node senses the pheromone levels on adjacent edges to inform its next movement. Observations suggest that a turtle ant tends to keep moving in the same direction, indicating that an ant is able to avoid the previous node it visited. We thus assume that simulated ants have one time-step of memory, used to avoid going back and forth along the same edge. As is characteristic of many species of ants [103, 104, 2], simulated ants have no unique identifiers and can use only local information.

1.3.1.1 Parameters.

Our algorithm uses three biological parameters: $q_{\text{add}}, q_{\text{decay}}$, and q_{explore}. The first parameter (q_{add}) determines how much pheromone is added when an ant traverses an edge. After each time step, each edge (v_1, v_2) traversed increases its edge weight as:

$$w(v_1, v_2) \leftarrow w(v_1, v_2) + q_{\text{add}}. \tag{1.1}$$

Without loss of generality, we fix $q_{\text{add}} = 1$, representing a unit of pheromone that an ant deposits on each edge traversed.

The second parameter (q_{decay}) specifies how much pheromone evaporates on each edge in each time step due to natural decay. We model pheromone decay as an exponential decrease in edge weight [105,

[1] In principle, if a branch or vine is cracked but not removed, it could create two new dead end nodes. Here we focus on the scenario in which a link is removed in its entirety.

106]; thus $q_{\text{decay}} \in (0, 1)$, and at each time step, for each edge (v_1, v_2), its weight is updated as:

$$w(v_1, v_2) \leftarrow w(v_1, v_2) \times (1 - q_{\text{decay}}). \tag{1.2}$$

Larger values of q_{decay} correspond to more rapid decay of pheromone on the edge.

The third parameter (q_{explore}) specifies the probability that an ant takes an "explore step". The definition of an "explore step" is algorithm-specific (see below), but intuitively, it involves choosing an edge with relatively less or no pheromone. Such deviation is clearly required by any network repair algorithm, since after the routing backbone is ruptured, edges not part of the existing path must be traversed to repair the break. Field observations show that even in the absence of a break, turtle ants explore edges off the main trail. This allows them to discover new food sources and incorporate them into the trail network [3].

1.3.1.2 Performance metrics.

After T time steps, we evaluate the outcome of the algorithm using the following measures (averaged over 50 repeat simulations):

1. Success rate: The probability that the simulated ants succeeded in forming a new path from u to v that does not use the broken edge. In this new path, ants are not required to traverse edges of relatively low weight (Methods). Higher values are better; for example, a success rate of 70% means that in 70% of the simulations, the ants successfully formed an alternative path.

2. Path entropy: An information-theoretic measure of how well the ants converge to a single consensus path, rather than creating multiple, potentially overlapping, $u \to v$ paths with pheromone. Lower values are better, indicating that subsequent ants using the same algorithm on the resulting network will all follow a common path, rather than dispersing along many different paths. This measure is computed only in the simulations in which an alternative path was successfully found. Field observations show that turtle ants consistently converge to a consensus path, and loops in the network are often pruned away over time [3]. This reduces the numbers of lost ants and the numbers of ants traveling in circles.

3. Path length: The length of the new path. Although turtle ants do not always find the globally shortest path [3], we include this measure because it is commonly used to evaluate routing algorithms. Lower values are better, indicating shorter paths. This measure is computed only in the simulations in which an alternative path was successfully found.

1.3.1.3 A model of computation for individual ants.

We assume that all ants are identical and have the following computational abilities:

- Each ant can avoid the node it immediately previously visited. It cannot, however, remember its entire path from the nest up to its current point. The ant may also keep track of a binary state variable that determines whether it is combing back from a dead end and should discontinue laying pheromone.
- In field observations, ants appear to pause at nodes and inspect more than one edge before choosing an edge to take [3]. Thus, each ant can access all adjacent edge weights to decide which node to visit next. To choose its next edge, we allow ants to perform any Turing-computable computation, although we show that a simple, albeit non-linear, function will suffice.

See Methods for full technical details of the model and performance metrics.

1.3.2 Candidate distributed algorithms

Below we introduce several biologically plausible algorithms that attempt to describe how a turtle ant at a node s chooses which edge to traverse next among possible neighboring edges $t_1, t_2, ...t_n$. These algorithms build upon previous linear and non-linear models used to analyze ant trail formation in other species, such as Argentine ants [107, 108, 109] and pharaoh ants [110]. Let $w(s, t_i)$ be the current weight on edge (s, t_i), and let uniform$()$ be a random value drawn uniformly from $[0, 1]$.

In the *Weighted* random walk (Algorithm 1), each ant chooses the next edge to traverse with probability proportional to the amount of pheromone on that edge: the more pheromone on an edge, the more likely an ant is to traverse that edge. However, with probability q_{explore}, the ant takes an edge that has zero pheromone.

Algorithm 1 *Weighted* random walk

$X \leftarrow \{t_i : w(s, t_i) > 0\}$ # Explored edges
$Y \leftarrow \{t_i : w(s, t_i) = 0\}$ # Unexplored edges
if uniform$() < q_{\text{explore}}$ **then**
  return $t_i \in Y$ with probability $1/|Y|$
else
  return $t_i \in X$ with probability $w(s, t_i)/\sum_{j \in X} w(s, t_j)$
end if

Note: The algorithm excludes the previously visited node from the sets of candidate edges, unless it is at a dead end. If all neighboring edges have weight 0, the ant chooses a zero-weight edge with probability 1 rather than probability q_{explore}. If none of the neighboring edges have weight 0, then the ant chooses an edge with nonzero-weight with probability 1 rather than probability $1 - q_{\text{explore}}$.

In the *RankEdge* random walk (Algorithm 2), with probability $1 - q_{\text{explore}}$, the ant chooses an edge

with the highest weight (ties are broken at random). With probability q_{explore}, it bypasses the highest weighted edges and considers edges with the second highest weight. With probability $q_{\text{explore}}(1 - q_{\text{explore}})$, it chooses an edge with the second highest weight. With probability q_{explore}^2, it bypasses both the highest and second highest weighted edges and considers edges with the third highest weight, and so on.

Algorithm 2 *RankEdge* random walk

$W \leftarrow [w_1, w_2, \ldots, w_k]$ # Sorted unique edge weights, in decreasing order
for $i = 1 \ldots k$ **do**
  $X \leftarrow \{t : w(s,t) = W[i]\}$
  if uniform() $< (1 - q_{\text{explore}})$ **then**
    return $t_i \in X$ with probability $1/|X|$
  end if
end for

Note: The algorithm excludes the previously visited node from the set of candidate edges. If all neighboring edges are tied for the highest weight, then a maximally-weighted edge is chosen with probability 1 rather than probability $1 - q_{\text{explore}}$. If an ant keeps exploring until it gets to the lowest weight, it takes one of the edges tied for the lowest weight with probability 1 rather than probability $1 - q_{\text{explore}}$.

Each algorithm contains additional details inspired by field observations, including a queueing system so ants traverse edges one at a time, the ability to traverse and return from an edge on an explore step in one time-step, and the ability to discontinue laying pheromone on the way back from a dead-end. See Methods for full details.

1.3.2.1 Comparison to additional non-linear algorithms

The *RankEdge* algorithm requires an ant arriving at a junction to rank-order all outgoing edges by weight before making a decision. Thus, we test *RankEdge* against five other, arguably simpler algorithms. We describe these algorithms here, and present results on all of them in tables in the main text. We also compared to a null model, called the *Unweighted* random walk. The null model uses no parameters; instead, the ants ignore edge weights and choose amongst candidate edges with equal probability. Overall, we find that *RankEdge* does the best job of simultaneously explaining field data and performing well in simulations, and so we primarily focus on the *RankEdge* algorithm in our figures.

- *MaxEdgeA*: With probability $1 - q_{\text{explore}}$, choose randomly among edges tied for the highest weight. With probability q_{explore}, choose randomly among all other edges (i.e., excluding the edge with the highest weight). This algorithm only needs to distinguish between edges with the highest weight and edges with any other weight.
- *MaxEdgeB*: With probability $1 - q_{\text{explore}}$, choose randomly among edges tied for the highest weight.

With probability q_{explore}, choose randomly among all edges (including the edge with the highest weight). This algorithm only needs to distinguish between edges with the highest weight and edges with any other weight.

- *MaxEdgeC*: With probability $1 - q_{\text{explore}}$, choose randomly among edges tied for the highest or second highest weight. With probability q_{explore}, choose randomly among edges with less than the second highest weight. This algorithm only needs distinguish between edges with the highest or second highest weight and edges with any other weight.
- *MaxWeighted*: With probability $1 - q_{\text{explore}}$, choose randomly among edges tied for the highest weight. With probability q_{explore}, choose randomly among all edges with probability proportional to the weight of the edge. This is a hybrid algorithm, using *MaxEdgeA* when not exploring, and using *Weighted* otherwise.
- *Deneubourg* [111]: From node u, choose edge (u, v) with probability:

$$\frac{(a + w(u,v))^{1/q_{\text{explore}}}}{\sum_{(u,v') \in E(G)} (a + w(u,v'))^{1/q_{\text{explore}}}},$$

where we set $a = 1$ to constrain this model to two parameters (as in the other models). Low values of q_{explore} more strongly biases towards picking the highest weighted edge, and vice-versa. We find that the maximum likelihood value of q_{explore} is $0.78 < 1$, meaning the exponent is strictly greater than 1. Thus, the probability an edge gets chosen is not a linear function of the weight. Note that under this model, as q_{explore} goes to 0, the probability of taking the highest weighted edge goes to 1.

1.3.2.2 Summary of conclusions.

Overall, we find that non-linear models perform the best at simultaneously explaining field observations and providing a mechanism by which turtle ants could solve the network repair problem. While the linear (Weighted) algorithm does perform well at explaining some aspects of field observations, and it repairs breaks with high probability, it also produces a very high path entropy, with poor convergence to a consensus path. This departs strongly from field observations [3] that show that when repairing broken paths, the ants quickly converge to a single path.

In particular, we find that: (A) *RankEdge* outperforms all other non-linear algorithms, except for *MaxEdgeA*, in the likelihood of explaining observed edge choices by turtle ants (Table 1.1). The log-likelihoods of *RankEdge* and *MaxEdgeA* were nearly identical. (B) When parameterized by field data, *RankEdge* outperforms all other non-linear algorithms in success rate (Table 1.2); and (C) *RankEdge* is

Table 1.1: **Maximum likelihood estimates for each algorithm.** For each algorithm, we show the values of q_{explore} and q_{decay} that maximize the likelihood of producing the observed choices made by turtle ants in the field. Standard deviation is computed across 13 junctions, each corresponding to field observations of one junction on a given day. All models are significantly more likely to explain the data than the null model (*Unweighted*).

Algorithm	q_{explore}	q_{decay}	log-likelihood
Unweighted	N/A	N/A	-744.25
Weighted	0.05 ± 0.06	0.01 ± 0.10	-345.05
RankEdge	0.20 ± 0.04	0.02 ± 0.08	-405.42
MaxEdgeA	0.20 ± 0.04	0.02 ± 0.08	-402.63
MaxEdgeB	0.42 ± 0.04	0.01 ± 0.09	-424.61
MaxEdgeC	0.23 ± 0.01	0.02 ± 0.09	-586.87
MaxWeighted	0.22 ± 0.01	0.01 ± 0.09	-568.70
Deneubourg	0.78 ± 0.10	0.01 ± 0.03	-518.18

Table 1.2: **Success rates for each algorithm.** For each algorithm, we show the success rate on each simulated network. The last column summarizes the robustness of each method across all networks. Of the non-linear algorithms, *RankEdge* performs the best.

Algorithm	Minimal	Simple	Medium	Full	Spanning Grid	Europe	Robustness
Unweighted	1.00	1.00	1.00	1.00	1.00	1.00	1.00
Weighted	1.00	1.00	1.00	1.00	1.00	1.00	1.00
RankEdge	1.00	0.98	0.84	0.70	0.54	0.70	0.78
MaxEdgeA	1.00	0.78	0.74	0.48	0.36	0.62	0.63
MaxEdgeB	0.76	0.60	0.64	0.64	0.44	0.70	0.62
MaxEdgeC	1.00	1.00	0.84	0.48	0.44	0.48	0.66
MaxWeighted	1.00	0.88	0.72	0.63	0.38	0.68	0.68
Deneubourg	1.00	1.00	0.94	0.44	0.67	0.52	0.72

equivalent to all other non-linear algorithms in path entropy (Table 1.3). Compared to the linear *Weighted* algorithm, *RankEdge* has a lower likelihood of explaining the observed edge choices and a lower success rate. However, *RankEdge* performs much better in path entropy, path length, and maintaining the trail in the absence of a break. We emphasize that the strong success rate of *Weighted* is because pheromone is left essentially on every edge in the graph. This guarantees high success but very poor convergence to a single path. Field experiments show that turtle ants converge strongly to a single path.

1.3.3 Q1. Field observations to determine the best algorithm and parameter values

We first determined what parameter values best allow each algorithm to match the data from field observations. We then used these parameter values to test algorithm performance for the network repair problem.

The performance of each candidate algorithm is sensitive to the values chosen for the two free parameters, q_{explore} and q_{decay} (as previously mentioned, we set $q_{\text{add}} = 1$). For example, with low values

Table 1.3: **Path entropy for each algorithm.** For each algorithm, we show the path entropy on each simulated network. Lower values indicate convergence to fewer paths, and 0 indicates convergence to a single unique path. All non-linear algorithms achieve the optimal path entropy. The standard deviation of the entropy for all non-linear models is 0. We do not report an interval for *Unweighted* because it does not depend on pheromone amount and thus has the same limiting behavior in all cases.

Algorithm	Minimal	Simple	Medium	Full	Spanning Grid
Unweighted	0.00	0.69	1.79	13.75	7.24
Weighted	0.00 ± 0.00	0.23 ± 0.25	1.52 ± 0.20	12.38 ± 0.22	5.94 ± 0.15
RankEdge	0.00	0.00	0.00	0.00	0.00
MaxEdgeA	0.00	0.00	0.00	0.00	0.00
MaxEdgeB	0.00	0.00	0.00	0.00	0.00
MaxEdgeC	0.00	0.00	0.00	0.00	0.00
MaxWeighted	0.00	0.00	0.00	0.00	0.00
Deneubourg	0.00	0.00	0.00	0.00	0.00

of q_{explore}, the ants may take a long time to explore enough new edges to find an alternative path; on the other hand, for high values of q_{explore}, the ants will scatter throughout the network and may not converge to a single path. Similarly, for high values of q_{decay} (pheromone decays rapidly), it may be difficult to build and reinforce a single path; for low values of q_{decay}, it may be hard for the colony to eliminate unnecessary edges and commit to one path. These two parameters also affect each other; for example, the higher the decay rate, the fewer edges with pheromone, and thus the more possible edges to explore.

We used data from observations made in the field to evaluate the match between the choices of edges made by turtle ants and the choices predicted by a candidate algorithm (with parameters $q_{\text{explore}}, q_{\text{decay}}$). Observations were made at La Estacion Biologica de Chamela in Jalisco, Mexico [56, 3]. Ants were observed traversing a junction (node) along a foraging trail. We recorded the time at which an ant moved to or from that junction node, and the edge it chose to traverse (Figure 1.2A–B). Observations were made of six different colonies, with an average of 2.16 junctions per colony, over three days in June 2015 and one day in June 2016. We observed 13 different junctions for time periods ranging from 7 to 24 minutes (mean of 12.3 minutes per observation at a given colony on one day), for a total of 773 edge choices made by turtle ants.

Maximum likelihood estimation: We determined which algorithm and parameter values best explained the observed edge choices made by turtle ants using maximum likelihood estimation (MLE). The data were used to determine the likelihood that a given algorithm, with a given pair of parameter values, would have produced the observed set of edge choices. Figure 1.2A–B shows an example likelihood calculation, and Figure 1.2C–D illustrates the results of the MLE for each algorithm over all pairs of parameter values.

Overall, for *RankEdge*, the maximum likelihood parameter values that best explained the observed

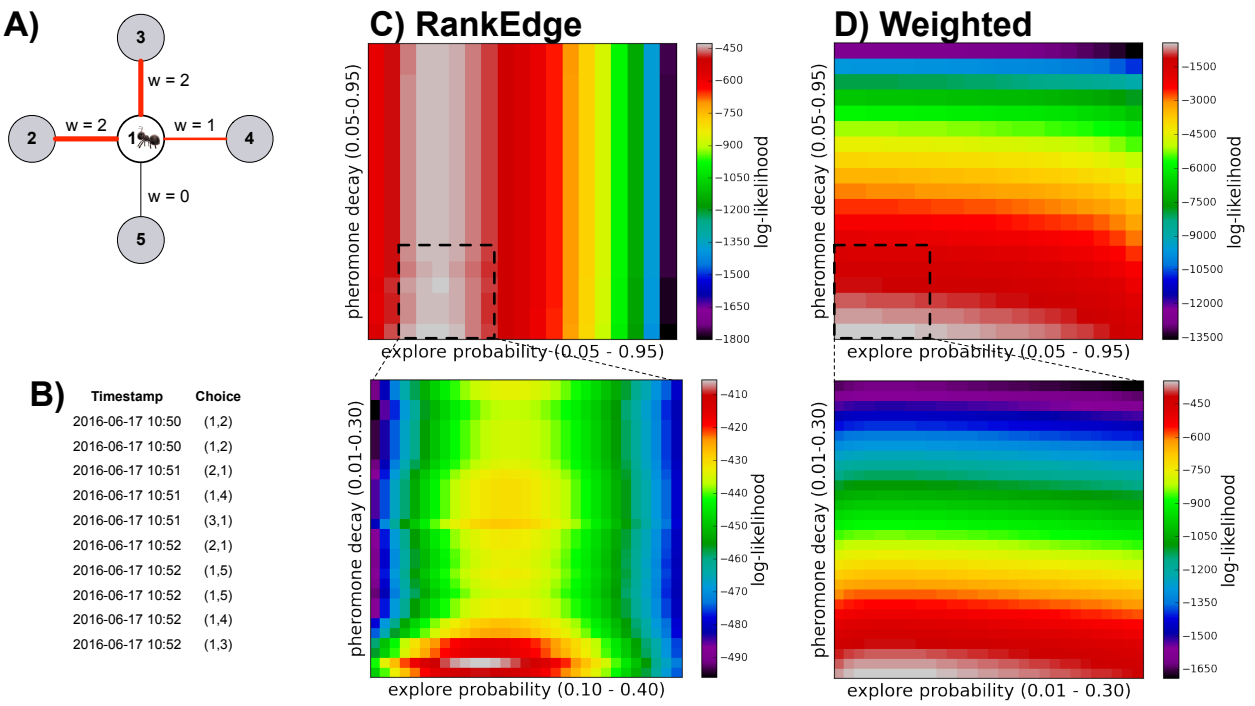


Figure 1.2: **Maximum likelihood computation.** A–B) Example node junction and edge choices for turtle ants. All ants arrive at node 1 from a different node that is not shown. In the example, we assume pheromone has been deposited at previous time-points, and we now compute the likelihood of the next ant choice. Under the *RankEdge* algorithm, the likelihood of choosing edges $1 \to 3$ or $1 \to 2$ is $(1 - q_{\text{explore}})(1/2)$; the likelihood of edge $1 \to 4$ is $q_{\text{explore}}(1 - q_{\text{explore}})$; and the likelihood of $1 \to 5$ is q_{explore}^2. Under the *Weighted* algorithm, the likelihood of choosing edge $1 \to 5$ is q_{explore}; the likelihood of edge $1 \to 4$ is $(1 - q_{\text{explore}})(1/(1+2+2))$; and the likelihood of edges $1 \to 2$ or $1 \to 3$ is $(1 - q_{\text{explore}})(2/(1+2+2))$. Under the *Unweighted* algorithm, the edge weights are disregarded, and the likelihood of taking any one of the four edges is $(1/4)$. C–D) For each combination of q_{explore} (x-axis) and q_{decay} (y-axis) values, we determined the pair's likelihood of producing the choices observed in turtle ants. Each heatmap shows the likelihood for each algorithm with a zoom-in below around the highest likelihood region. The optimal parameter values for each algorithm, depicted in white, are shown in Table 1.1.

turtle ant behavior were: $q_{\text{explore}} = 0.20$, and $q_{\text{decay}} = 0.02$ (Table 1.1). For *Weighted*, the maximum likelihood parameter values were $q_{\text{explore}} = 0.05$ and $q_{\text{decay}} = 0.01$. Both candidate algorithms were more likely to explain the data than the null model (Table 1.1).

1.3.3.1 Variance of maximum likelihood estimations

The maximum likelihood heatmap (Figure 1.2) showed the cumulative likelihoods over all 13 turtle ant junctions. Here we confirm that the likelihood estimation produced consistent results in each of the 13 junctions.

We found the values for q_{explore} and q_{decay} that maximized the likelihood of producing the edge choices in each of the 13 junctions individually (Figure 1.3). We varied $q_{\text{explore}} \in [0.10, 0.40]$ and $q_{\text{decay}} \in$

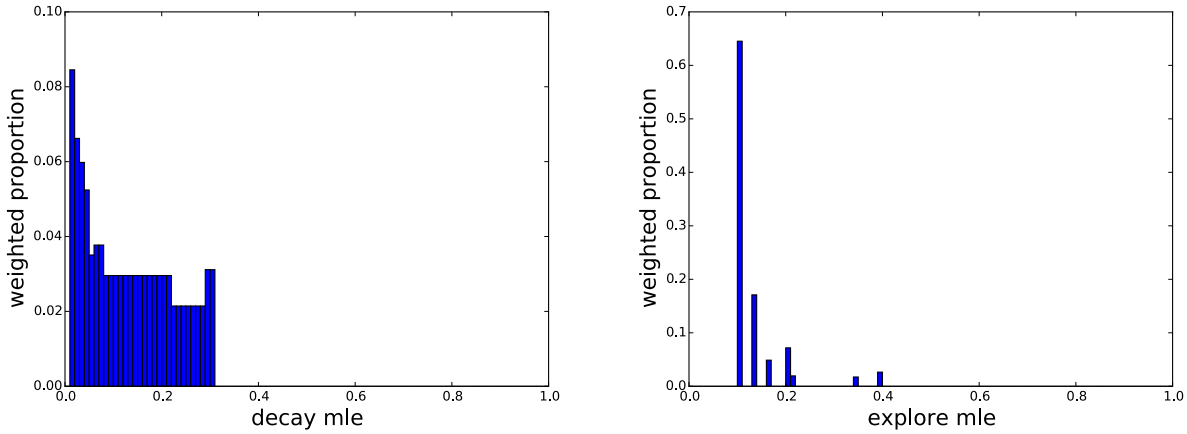


Figure 1.3: **Variance of maximum likelihood estimation.** Maximum likelihood analysis of the *RankEdge* algorithm for q_{decay} (left) and q_{explore} (right).

$[0.01, 0.30]$ in increments of 0.01 (the same range used for the maximum likelihood analysis in Figure 1.2), and recorded the maximum likelihood parameters for each junction. We then plotted a histogram of the distribution of the maximum likelihood parameters across the different junctions. Each maximum likelihood estimate is weighted by sample size; i.e., its contribution is weighted in proportion to the number of edge choices in that junction.

We observe that the distribution of most likely parameter values is tight, especially the values for q_{explore}, for which over 60% of the (weighted) junctions had nearly the same q_{explore} value (Figure 1.3). There is some variation in q_{decay}, possibly due to differences among days in environmental conditions that could affect pheromone evaporation rates. Overall, these results suggest that similar algorithm parameters apply for different colonies and days.

1.3.4 Q2. Algorithm performance on synthetic and real-world planar networks

Our goal here is to test how well each algorithm solves the network repair problem on simulated and real-world networks. We were particularly interested in how well each algorithm performed when its parameters were set to the maximum likelihood values derived from observations of turtle ants. Our main result is that the maximum likelihood parameters for the *RankEdge* performed well in simulations for network repair across six networks (Figure 1.4; the black rectangle in both panels shows that the parameter values that best explain the turtle ants' behavior also perform best for solving the network repair problem.) The latter result is substantiated below.

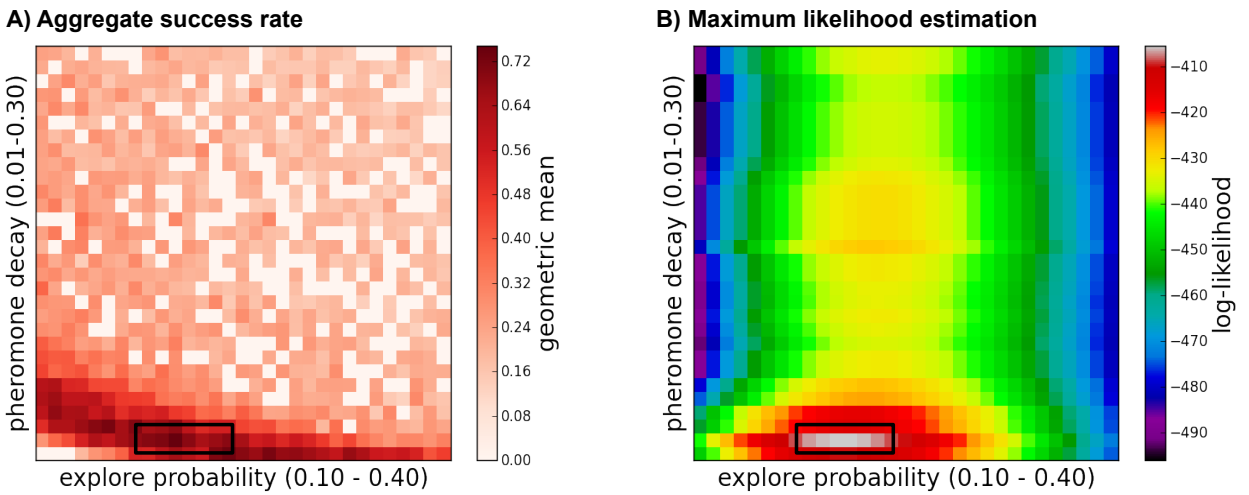


Figure 1.4: **The maximum likelihood parameters closely match the best simulation parameters.** A) The color of each square in the heatmap corresponds to the robustness (Methods) of the success rates for the *RankEdge* algorithm for each combination of q_{explore} (x-axis) and q_{decay} (y-axis) values. Results are aggregated over the six simulated and real-world networks presented in Figures 1.5 and 1.6. B) The maximum likelihood parameter estimates for *RankEdge* from observations of turtle ants. The black rectangle in both panels shows that the parameter values that best explain the turtle ants' behavior also perform best for solving the network repair problem.

1.3.4.1 Simulation setup

For all simulations, we ran each algorithm for $T = 1000$ steps using $N = 100$ ants, and repeated each simulation 50 times. To initialize each simulation, we placed each of the N ants at a random node in the original path. This means that at the start of the simulation there were likely ants at nodes on both sides of the rupture in the path. No ants were placed at nodes not part of the original path. Each ant was randomly assigned to walk in search of one of the two nests. All edges that were part of the initial path were initialized with 10 units of pheromone. All other edges were initialized to 0 units of pheromone. When an ant reached its destination nest, it attempted to return to the other nest, and repeated this, going back and forth between nests, for T time-steps. The ants walk synchronously for T time-steps; this is a common assumption in distributed computing problems.

1.3.4.2 Simulation Networks

Our first performance metric, called the *success rate*, measures how well the ants succeed in finding an alternative path to repair the break. We simulated breaks under six planar network structures, which have an increasingly complex topology with varying numbers of possible paths. In each evaluation below (Figure 1.5), we show three panels: the initial network with a break, the final network at the end of the simulation, which is generated using the MLE parameter values, and a heatmap showing the success rate

for pairs of parameter values ($q_{\text{explore}}, q_{\text{decay}}$) close to the MLE range. In each synthetic network, a only single link is broken (shown as the 'X' mark in Figure 1.5); in section 1.3.4.7, we describe cases where multiple links are broken.

We analyzed all algorithms but focus on results for *RankEdge* because *Weighted* rarely converged onto a single path, thus it did not satisfy our second performance metric. It also did not maintain trails in the absence of a break. These results are described in detail later. Also, see section 1.3.2.1 and Table 1.2 for analysis of the additional non-linear algorithms.

Minimal graph (Figure 1.5A): Here we find that *RankEdge* can solve a basic repair problem in a minimal working example, in which the break causes the existing path to lead to a dead end that should be avoided in favor of a single alternative path to the nest. To favor the alternative path, the simulated ants must largely eliminate the pheromone on the edge leading to the dead end, a process which we call 'pruning'. To favor the alternative path instead of the existing path, the ants should put more pheromone on the edge leading upwards to the alternative route, even though this edge initially had no pheromone.

We find that the *RankEdge* algorithm succeeds in this task 100% of the time, as long as the ants do not leave pheromone on the way back returning from the dead-end (Methods and Q4).

Simple graph (Figure 1.5B): Here we increased the complexity of the graph to offer two alternative paths, instead of one in the Minimal graph. We found that *RankEdge* not only prunes the dead-end, but it can find and commit to one of the two alternatives with a 98% success rate.

Medium graph (Figure 1.5C): Here we further increased the complexity of the Minimal graph to offer six alternative paths and found that *RankEdge* not only prunes the dead-end, but can find and commit to one of the six alternatives with a 84% success rate.

Full grid (Figure 1.5D): The Full grid presents a different computational challenge: there is no dead-end to prune and the shortest alternative path requires only 3 additional edges. However, the total number of possible new paths is extremely large, which makes it difficult to find and commit to a single path. The Full grid is also a standard benchmark used in the ANTS problem (Related work), in which ants search the integer plane [89, 90, 91].

We found that the highest success rate (70%) occurred for low values of q_{decay}, which closely matches the observed best decay value estimated using maximum likelihood. This highlights an inherent trade-off in the turtle ant algorithm. Low decay rates help preserve the initial path and bias the turtle ants toward finding an alternative route that re-uses as much of the previous path as possible; that is, with low decay rates, repair starts as close to the break as possible. However, low decay rates also limit the capacity to search for other paths that may be shorter even though they re-use less of the previous path. An alternative would be to use higher values of q_{explore} to search for other paths that do not re-use the initial

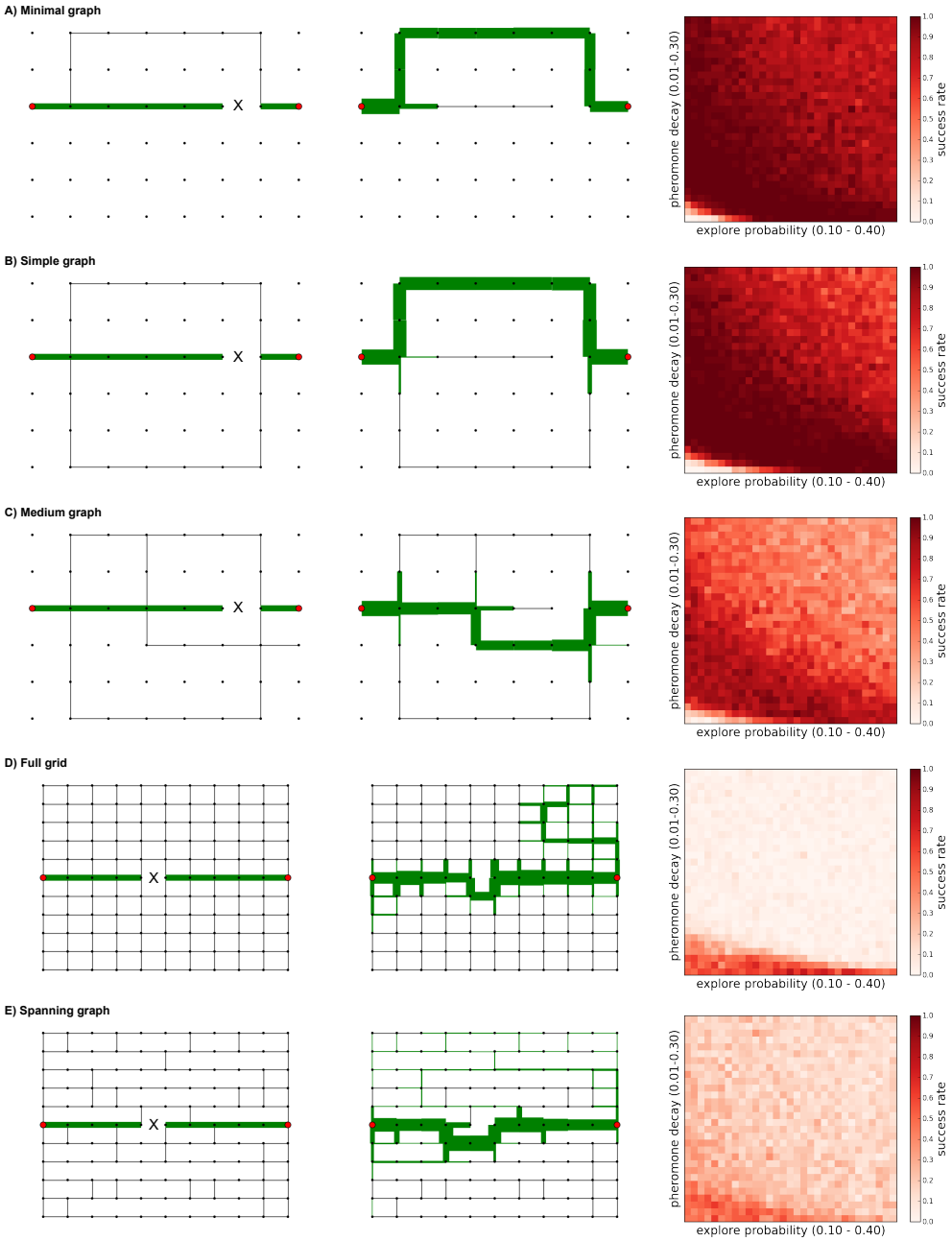


Figure 1.5: **Success rates for each network.** A–E) For each network we show the initial graph (left), an example of the final graph after running the *RankEdge* algorithm using the maximum likelihood parameters (middle), and the algorithm's success rate for each parameter combination (right). In each panel, black dots indicate nodes in the network, and solid lines indicate edges that may be traversed. If two adjacent nodes are not connected by an edge, there is a space between them. In the initial graphs, the 'X' marks the edge that is broken. The x-axis of the heatmap (right column) shows q_{explore}, and the y-axis shows q_{decay} under the range close to the MLE parameters. Darker shades of red are indicate success rates closer to 1, and thus are better.

path, but this would make it more difficult for the ants to converge to a single new path.

Spanning grid (Figure 1.5E): In contrast to the Full grid, the Spanning grid is sparser and requires

that the ants go back at least one node from the break to find an alternative path.

We found that the maximum likelihood parameters produced a moderate success rate (54%). As above, the highest success rate occurred for low values of q_{decay} and moderate values of q_{explore}. These values achieve a good trade-off between searching sufficiently far from the break to find an alternative path, and largely preserving the previous path. The performance on the Spanning grid demonstrates that the algorithm is flexible enough to search locally around a break point for new paths, while still maintaining most of the old path.

The results from the Full grid and the Spanning grid together suggest that the algorithm performs best when it preserves as much of the previous path as possible, even if it can not re-use all of the original path. This is consistent with field observations that showed that turtle ants sought alternative paths in a "greedy" manner, by going back up to 1 or 2 nodes from the break point, even though going back more nodes may have resulted in a path with fewer nodes overall [3].

European road transportation network (Figure 1.6): To demonstrate the utility of this algorithm in a real-world scenario, we applied the *RankEdge* algorithm to repair networks in a human-designed transport network. We downloaded the network depicting the major roads (edges) connecting intersections (nodes) in the international E-Road in Europe [112] (Methods). We removed an edge from an existing path between two nodes and ran the *RankEdge* algorithm to repair the simulated closure. The *RankEdge* algorithm achieved a success rate of 70%, indicating that the turtle ant algorithm can also repair breaks in real-world topologies. This shows how distributed solutions may be useful for new application domains, such as for swarm robotics or molecular robots [113, 12, 114, 115] in remote environments, when centralized or global positioning systems may not be as effective.

Performance on additional random graphs: Our analysis focused on planar graphs, since the turtle ant environment is a physical network. Here, we further test performance on two additional, well-studied random graph models: Watts-Strogatz [116] small-world networks ($n = 121, k = 4, p = 0.05$), and Erdos-Renyi (ER) random networks ($n = 121, p = 4/120$). The number of nodes n is equal to the number of nodes in the Full grid, and the other parameters were selected such that each node has an expected degree of 4 (also like the Full grid). We find that *RankEdge* has a success rate of 54% on ER networks and 46% on WS networks (geometric mean of 0.498), and a path entropy of 0 for both networks. By comparison, the *Deneubourg* algorithm achieved lower success rates of 46% on ER networks and 28% on WS networks (geometric mean of 0.359), also with path entropy of 0. *Weighted*, as expected, achieved 100% success but showed a very high path entropy: 6.69 on ER networks and 5.63 on WS networks.

The turtle ant algorithm evolved to operate on networks with physical, geometric constraints, whereas the ER and WS random network models are not constrained as such. For this reason we did not emphasize

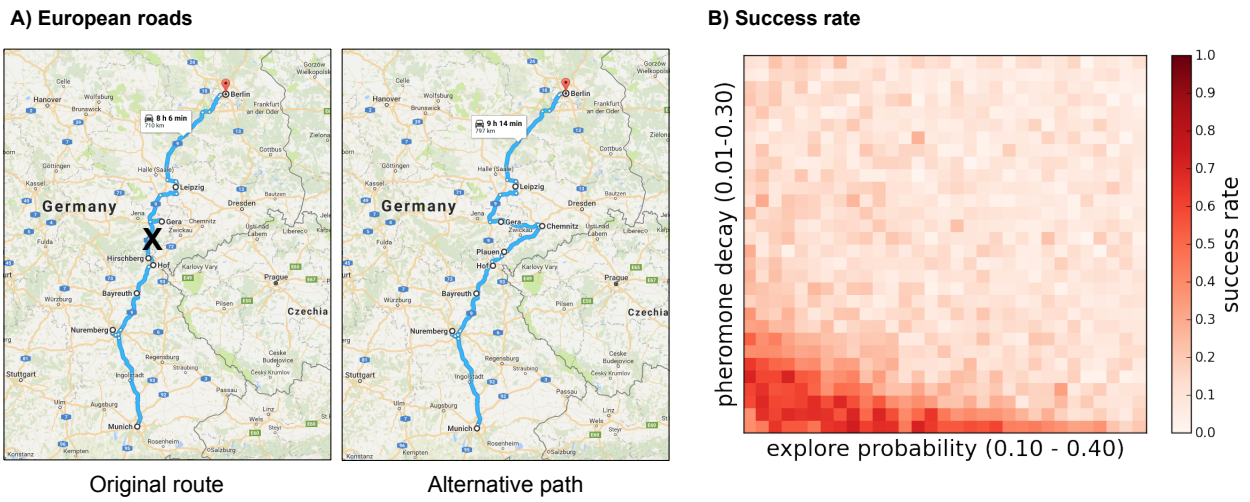


Figure 1.6: **Repairing road closures in the Europe road graph.** Analysis of how well the turtle ant algorithm translates to repair simulated breaks in a real-world transport network. A) An example of a path in the European E-road network connecting Munich to Berlin, Germany. The roads and junctions form a graph. On the left, the black 'X' shows a road that has been broken or closed along the path. On the right, we show an alternative path that avoids the broken road. B) The success rate of the turtle ant algorithm (*RankEdge*) applied to this network. Map data: Google, DigitalGlobe.

performance on these random networks. Further work is needed to investigate the theoretical relationship between the performance of *RankEdge* and network topology, as defined by properties such as degree distribution, clustering coefficient, or distribution of path lengths.

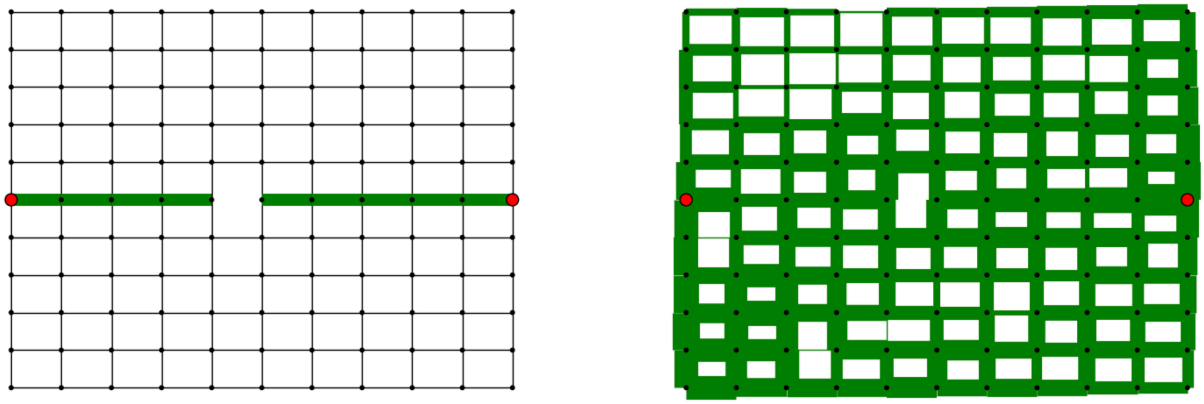
1.3.4.3 Q2a. Converging onto a single consensus path (Table 1.3).

Our second performance metric, called *path entropy*, measures how well foragers commit to a single alternative path.

We find that the *RankEdge* algorithm consistently achieves a path entropy near 0, indicating that on the final network all ants follow the same path when not taking explore steps (Table 1.3). This is particularly challenging for the Full and Spanning grids because both contain a large number of possible paths, and thus a large possible path entropy if the simulated ants exploit many paths. Thus, when the algorithm succeeds in repairing the path, *RankEdge* satisfies our second performance criterion.

More generally, one advantage of non-linear algorithms such as *RankEdge* is that all simulated ants, by simply following the maximal edge (the adjacent edge with the highest pheromone), can travel from one nest to the other using the same path, thereby achieving a path entropy of 0. On the other hand, linear algorithms such as *Weighted* do succeed in finding a path; however, *Weighted* is unable to commit to only one path, and thus has very high path entropy (Figure 1.7).

A) Full grid



B) Spanning grid

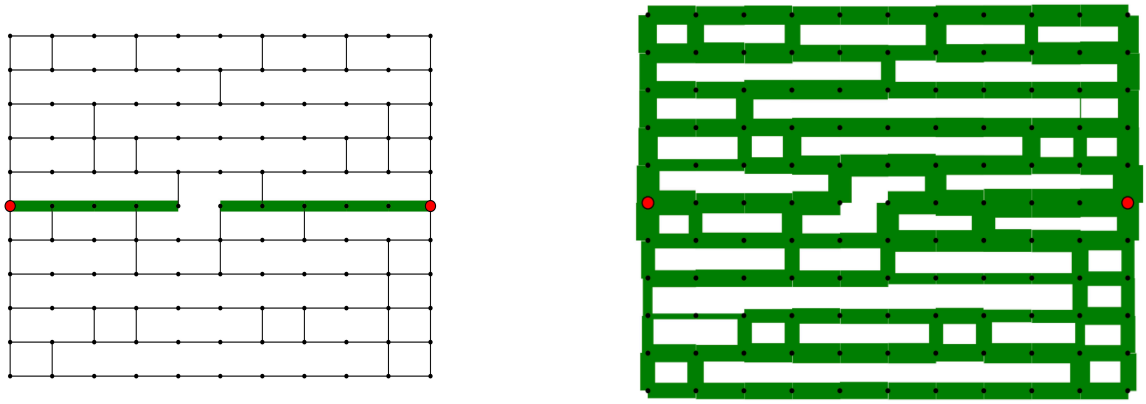


Figure 1.7: **Poor path entropy for *Weighted*.** The initial (left) and final (right) networks for the (A) Full grid and (B) Spanning grid. In both cases, the MLE parameter values ($q_{\text{explore}} = 0.05, q_{\text{decay}} = 0.01$) for *Weighted* did not find a low path entropy solution.

1.3.4.4 Q2b. Finding short paths (Table 1.4).

Our third performance metric measures the path length of the final trail network. We found that *RankEdge* consistently finds paths of lengths that are close to, though slightly larger than, the globally shortest path lengths. For every network, we compared the average path lengths of *RankEdge* versus every other algorithm using Welch's unpaired T-test. *RankEdge* finds significantly shorter paths than *Weighted* and *Unweighted* on the Full grid, Spanning grid, and European roads ($p < 0.05$); RankEdge is not significantly different from the other non-linear algorithms (Table 1.4). The improved performance over *Unweighted* demonstrates the value of using pheromone to solve the network repair problem collectively, instead of using independent search.

Table 1.4: **Average path length for each algorithm.** For each algorithm, we show the average path length of the final graph, measured as the number of nodes in the path. *RankEdge* performed much better than the null model (*Unweighted*) and close to the globally shortest path length (Optimal).

Algorithm	Minimal	Simple	Medium	Full Grid	Spanning Grid
Unweighted	12.00 ± 0.00	12.90 ± 1.00	13.16 ± 3.00	26.04 ± 6.56	20.05 ± 7.24
Weighted	12.00 ± 0.00	12.97 ± 0.85	13.04 ± 0.64	18.36 ± 0.18	16.93 ± 0.24
RankEdge	12.00 ± 0.00	12.98 ± 1.01	10.95 ± 1.01	13.06 ± 0.34	14.56 ± 3.97
MaxEdgeA	12.00 ± 0.00	12.87 ± 1.00	11.29 ± 0.97	13.17 ± 0.56	15.56 ± 4.38
MaxEdgeB	12.00 ± 0.00	13.30 ± 0.96	10.77 ± 0.99	13.06 ± 0.35	15.09 ± 4.07
MaxEdgeC	12.00 ± 0.00	13.02 ± 1.01	11.00 ± 1.01	13.16 ± 0.56	15.45 ± 3.54
MaxWeighted	12.00 ± 0.00	12.87 ± 0.99	10.89 ± 1.03	13.06 ± 0.58	14.46 ± 3.07
Deneubourg	12.00 ± 0.00	13.16 ± 1.00	11.06 ± 1.30	14.18 ± 3.47	13.94 ± 2.51
Optimal	12.00	12.00	10.00	13.00	13.00

1.3.4.5 Q2c. The power of bi-directional search.

We find that a bi-directional search, in which simulated ants attempt to create an alternative path concurrently from both sides of the break, allows the algorithm to perform significantly better than a uni-directional search using ants from only one side of the break. We tested this on the Full grid, and found that for the MLE parameter values for *RankEdge*, the success rate was on average 70% for a bi-directional search versus 14% for uni-directional search.

One might predict that uni-directional search would perform as well as the bi-directional search, while simply taking longer. However, we found this not to be true: using a bi-directional search means that once ants from side A of the break reach side B of the break, the rest of their search is directed by the pheromone trail laid by ants that started on side B. In the uni-directional search, even if ants from side A reach side B, they must still find a path from scratch connecting the dead end on side B to the nest on side B. Although uni-directional search has rarely been observed to occur in turtle ant networks, we tested it here to compare it with bi-directional search, which is often used to improve the performance of search algorithms.

To test the performance of an algorithm using uni-directional rather than bi-directional search, we performed simulations on the Full grid using *RankEdge* in which all ants were initialized to be on one side of the rupture. Figure 1.8 shows a significant loss in the ability to repair the network when using uni-directional search.

1.3.4.6 Q2d. The power of avoiding backtracking

Our simulated ants are able to avoid revisiting the previously visited node, except when an ant reaches a nest and turns around to go back, or when it encounters a dead end.

Here we present results from simulations using the *RankEdge* algorithm with maximum likelihood

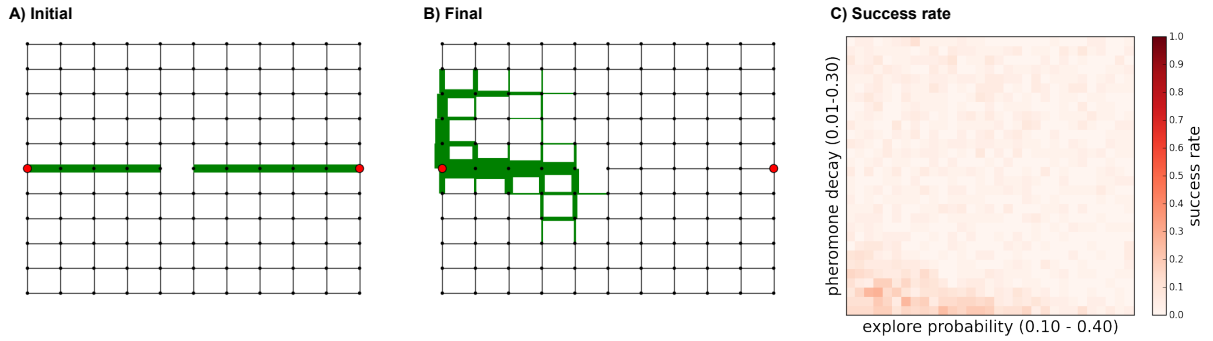


Figure 1.8: **Uni-directional vs bi-directional search.** A) Initial network. Simulations start with all ants placed at a random node on the left of the rupture. B) Final network. The pheromone on the right side quickly decays, making it much more difficult to build off the left path to repair the rupture. C) The success rate is significantly lower with uni-directional search than bi-directional search.

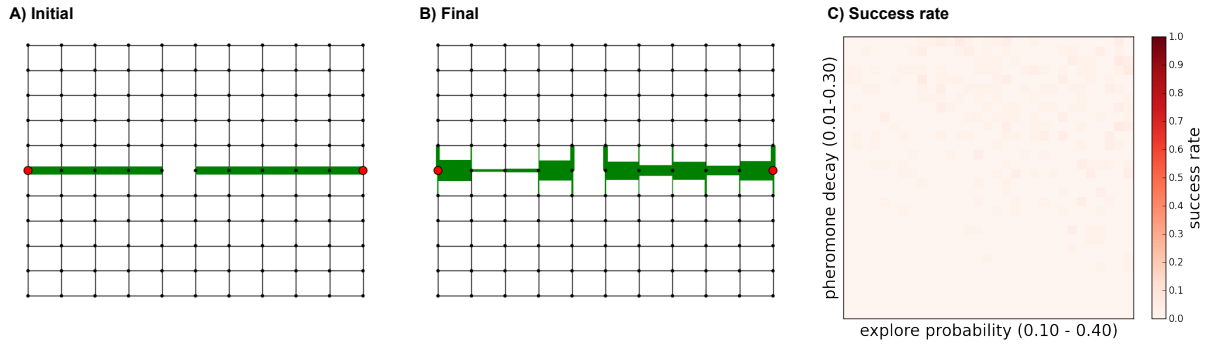


Figure 1.9: **Analysis of backtracking.** A) Initial network. B) Final network, showing that when ants are allowed to backtrack, few edges are repeatedly reinforced. C) The success rate is significantly lower when ants are allowed to backtrack.

parameter values, in which the ants are not prevented from using the previously visited node; i.e., they are allowed to backtrack. We find that providing simulated ants the ability to avoid backtracking, i.e., visiting the same node visited in the previous time-step (Methods) allows for a significant improvement in algorithm performance. In contrast, ants that are not given this ability could keep going back and forth along the same edge.

In particular, ants that used the *RankEdge* algorithm and avoided backtracking produced a success rate of 70% on the Full grid, compared to 0% when an ant was not prevented from potentially returning to the previous node it visited (Figure 1.9). Thus, providing ants with a basic node-to-node sense of direction led to a significant improvement in performance.

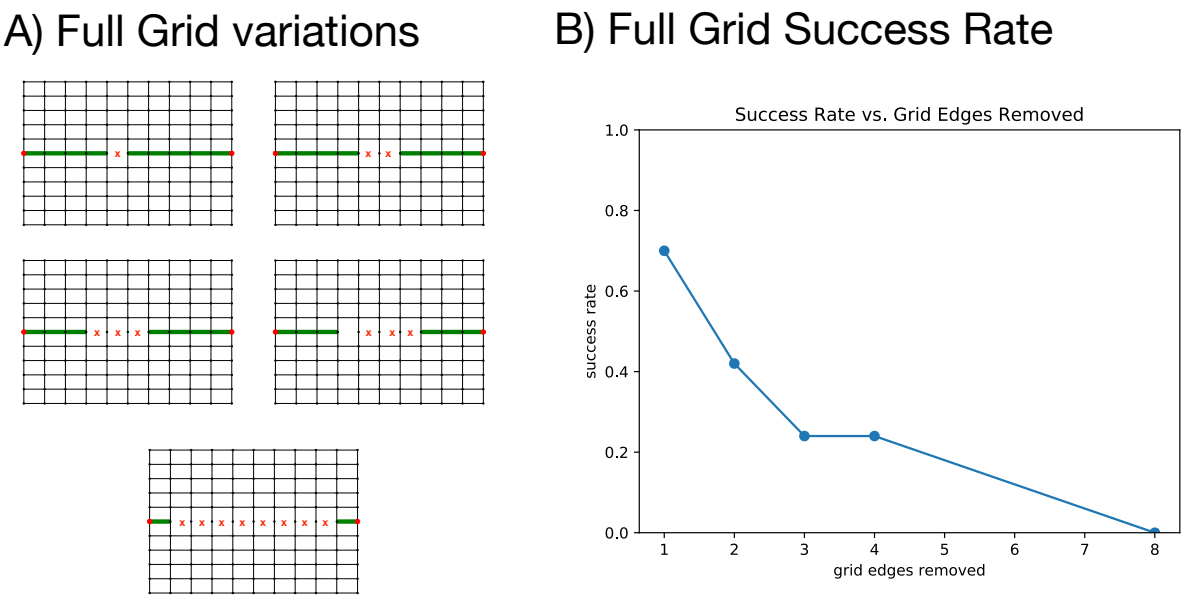


Figure 1.10: **Performance when multiple links are broken.**

1.3.4.7 Q2e. Performance when multiple links are broken

In field experiments and in the simulations reported above, we ruptured only one link in the path. In the field, edges or stems are broken occasionally by the wind or an animal moving through the vegetation. It is extremely unlikely for two or more such events to occur concurrently in the same path, and the repair process for the repair of a single link is quite rapid [3]. Nonetheless, here we test performance of the algorithm with multiple broken links.

Specifically, we tested *RankEdge* on the Full grid with different numbers of broken links; starting from one edge ruptured to the entire path missing (Figure 1.10A). We achieved success rates of 70%, 42%, and 24%, 24%, and 0%, respectively (Figure 1.10B). The performance of the algorithm decreases the more links are broken. These results, as well as field observations, suggest that if many links are broken (e.g., the last panel in Figure 1.10A), a different search strategy is used. This strategy may be similar to the one used when turtle ants initially establish a new path connecting nests; this problem requires further study.

1.3.4.8 Q2f. Critical features for the success of a plausible algorithm.

Here we provide a probabilistic argument for why the *RankEdge* algorithm prunes a dead-end significantly faster if ants do not lay pheromone on the way back from a dead-end. We do this by providing a coarse upper bound on the average time to prune in the two scenarios: when ants do not lay pheromone on the way back from a dead-end versus when they do lay pheromone on the way back.

Consider the Minimal graph, and the junction on the left where the ant must choose between the edge that leads to the nest on the right (the 'good edge'), and the edge that leads to the dead-end (the 'bad edge'). The *RankEdge* algorithm is biased towards selecting the highest weighted edge; therefore, for the ants to find the alternative path, there must be more pheromone on the good edge than on the bad edge.

We make the following assumptions in the course of this section:

- An ant makes a choice at the junction. If it takes the good edge, it will go over the edge and back in one step (Methods), or it will take the bad edge, and eventually come back to the junction after returning from the dead end. It will not lay pheromone on the way back from the dead-end.
  - No backtracking means that the ant might walk all the way to the dead end before turning around and discontinuing pheromone deposition. It will still lay only one unit of pheromone on the edge at the junction that leads to the dead end trail.
- We split time into intervals of K steps. For simplicity, we ignore decay in the current interval, consider choices with decay in the immediately previous interval, and ignore all choices in any prior intervals assuming all pheromone previously deposited has decayed so has no significant impact on the current weights.
- For simplicity, we assume an infinite number of ants; thus, after an ant chooses an edge and lays pheromone, it is put at the end of an infinite queue at node 1. This allows us to ignore the case in which an ant chooses the bad edge, and later is forced to choose the good edge due to the rule against backtracking. While this is obviously not how the simulations actually work, this simplifying assumption allows us to derive the intuition for how the algorithm works without getting bogged down in technical details. The *RankEdge*'s 100% success on the minimal network shows that this assumption does not cause an overbearing discrepancy between our theoretical analysis and the true nature of the simulations.

Informally, consider an ant at the junction that chooses between the good or the bad edge. The choice the ant makes at this junction determines whether it will reach the nest on the right side of the break or reach the dead-end. Thus, it suffices to consider a simplified version of the Minimal graph by considering only the choice made by the ant at that junction before the good and the bad edge. We seek to determine the probability and expected time needed for the ants to place more pheromone on the good edge than the bad edge.

As described in Methods, if an ant takes an "explore step" and chooses the lower weighted edge, it will deposit two units of pheromone on that edge. If an ant chooses the good edge on a non-explore step,

it will leave one unit of pheromone on the way to the nest and one unit on the way back, so that edge will be reinforced with 2 units of pheromone. If an ant chooses the bad edge on a non-explore step, it will not deposit pheromone on the way back from the dead-end, so that edge will receive only 1 unit of pheromone. We will consider the case where the bad edge initially has more weight than the good edge, and we want to determine how much time it takes before the good edge has more weight than the bad edge. Once this happens, the ants will have formed a maximal path between the two nests.

Formally, we have a graph with nodes $V = \{1, 2, 3\}$ and edges $E = \{(1, 2), (1, 3)\}$. Edge $(1, 2)$ represents the bad edge (with initially higher pheromone), and $(1, 3)$ is the good edge. Additionally, let $W_t(u, v)$ to be the weight of edge (u, v) at time t. At each time-step, an ant at node 1 must choose between the two edges. The ants follow the *RankEdge* algorithm, so they take the higher weighted edge with probability $1-q_{\text{explore}}$ and the lower weighted edge with probability q_{explore}. If the ant chooses edge $(1, 2)$ then it augments the weight of that edge by 1:

$$W_{t+1}(1, 2) \leftarrow W_t(1, 2) + 1.$$

If it chooses edge $(1, 3)$ then it augments that edge weight by 2:

$$W_{t+1}(1, 3) \leftarrow W_t(1, 3) + 2.$$

If the two edges have equal weight then it chooses each of the two with equal probability, but for now we will ignore that case.

At the end of each time-step each edge weight decays exponentially by a rate, q_{decay}:

$$W_{t+1}(1, 2) \leftarrow W_{t+1}(1, 2) \times (1 - q_{\text{decay}})$$

and

$$W_{t+1}(1, 3) \leftarrow W_{t+1}(1, 3) \times (1 - q_{\text{decay}}).$$

For example, suppose $W_t(1, 2) = 10$ and $W_t(1, 3) = 5$. With probability q_{explore}, the ant takes edge $(1, 3)$ and with probability $1 - q_{\text{explore}}$ it takes edge $(1, 2)$. If it takes $(1, 3)$ then

$$W_{t+1}(1, 3) \leftarrow W_t(1, 3) + 2 = 7,$$

and if it takes edge $(1, 2)$ then

$$W_{t+1}(1, 2) \leftarrow W_t(1, 2) + 1 = 11.$$

For example, suppose the ant takes edge $(1, 3)$. Then,

$$W_{t+1}(1, 2) \leftarrow 10, \quad W_{t+1}(1, 3) \leftarrow 7.$$

At the end of the time-step, both edges decay by q_{decay}:

$$W_{t+1}(1, 2) \leftarrow W_{t+1}(1, 2) \times (1 - q_{\text{decay}}) = 10 \times (1 - q_{\text{decay}})$$

and

$$W_{t+1}(1, 3) \leftarrow W_{t+1}(1, 3) \times (1 - q_{\text{decay}}) = 7 \times (1 - q_{\text{decay}}).$$

Assume $W_0(1, 2) > W_0(1, 3)$; i.e., there is more pheromone on the bad edge than the good edge.

Let $X_0 = W_0(1, 3) - W_0(1, 2) < 0$, and $(X_t)_{t=1}^{\infty}$ be a sequence of independent and identically distributed (i.i.d.) random variables representing the difference between the amount of pheromone at time t contributed to the good edge minus the bad edge. Then, for $t \in \{1, 2, 3, \dots\}$:

$$\text{Pr}(X_t = -1) = 1 - q_{\text{explore}}$$

and

$$\text{Pr}(X_t = 2) = q_{\text{explore}}.$$

If $X_t = -1$ then the ant chose the bad edge and deposited 1 unit of pheromone on edge $(1, 2)$. If $X_t = 2$ then the ant chose the good edge and deposited 2 units of pheromone on edge $(1, 3)$. We want to determine when $W_t(1, 3) > W_t(1, 2)$.

To determine the difference in pheromone at any time t, we first need to determine how much pheromone from choices made at times $< t$ still remains. If an ant made a choice at time i, then due to exponential decay, at time $t > i$, the contribution of that choice (X_i) to the difference in weight between the good and bad edge is:

$$X_i \cdot (1 - q_{\text{decay}})^{t-i}.$$

And thus,

$$W_t(1, 3) - W_t(1, 2) = \sum_{i=0}^{t} X_i \cdot (1 - q_{\text{decay}})^{t-i}.$$

Let T represent the first time when there is more pheromone on the good edge than the bad edge,

i.e.:

$$T = \inf\left\{ t : \sum_{i=0}^{t} X_i \cdot (1 - q_{\text{decay}})^{t-i} > 0 \right\}$$

Then at time T,

$$W_T(1,3) - W_T(1,2) > 0 \iff W_T(1,3) > W_T(1,2).$$

Thus we are interested in finding $E[T]$ to determine when the probability of taking the good edge is greater than that of taking the bad edge.

Proposition. $E[T]$ *is significantly lower if the ants do not lay pheromone on the way back from a dead end.*

While this is not a mathematically precise statement, we will justify why it is qualitatively true. We will do this by providing a formula that coarsely approximates the upper bound on $E[T]$, and then use this formula to give numerical approximations in the cases where the ants do and do not lay pheromone on the way back from a dead end, respectively.

Proof idea: Exponential decay means that the difference between the two edge weights is mostly determined by the most recent choices (i.e., recent ant choices have disproportionately more effect on the difference in the two edge weights than choices made long ago). Thus, for the good edge to overtake the bad edge in edge weight, it suffices for the ants to explore the good edge more than expected by chance in a short period of time. The ants can take advantage of high variability in small sample sizes to put more pheromone on the good edge than the bad edge, at which point the *RankEdge* algorithm will be biased towards selecting the good edge over the bad edge. Finally, by depositing only half as much pheromone on the bad edge than on the good edge (for reasons explained below), the ants reduce the number of explore steps needed in a short time period to put more pheromone on the good edge than the bad edge. We formalize this intuition below.

Proof: Informally, we want to determine how far back in time we need to consider choices before their contribution to the current difference in weight between the good and the bad edge becomes negligible. Consider a choice made at time nK, where K is some integer greater than 0. We want to determine the value of K such that the difference in weight at time nK depends only on the choices made between times $(n-1)K$ to nK and negligibly on choices made prior to $(n-1)K$. We also want K to be such that, regardless of the choices made prior to nK, it is possible for the good edge to have more weight than the bad edge by time $(n+1)K$. Thus, we want a large enough K to ignore most past choices, but small enough to make significant deviations possible within the next K steps.

Formally, let $K \in \mathbb{N}$, and consider times $t = \{K, 2K, 3K, \dots\}$. If K is sufficiently large such that $(1 - q_{\text{decay}})^{2K} \approx 0$, then to compute the difference at time $(n+1)K$:

$$\begin{aligned}\sum_{i=0}^{(n+1)K} X_i \cdot (1-q_{\text{decay}})^{(n+1)K-i} &= \sum_{i=0}^{(n-1)K} X_i \cdot (1-q_{\text{decay}})^{(n+1)K-i} + \sum_{i=(n-1)K}^{(n+1)K} X_i \cdot (1-q_{\text{decay}})^{(n+1)K-i} \\ &= \sum_{i=0}^{(n-1)K} X_i \cdot (1-q_{\text{decay}})^{2K+(n-1)K-i} + \sum_{i=(n-1)K}^{(n+1)K} X_i \cdot (1-q_{\text{decay}})^{(n+1)K-i} \\ &\approx \sum_{i=(n-1)K}^{(n+1)K} X_i \cdot (1-q_{\text{decay}})^{(n+1)K-i}\end{aligned} \tag{1.3}$$

Taking $K = 75$ and $q_{\text{decay}} = 0.02$ (the ***RankEdge*** maximum likelihood estimate of q_{decay}), we see that

$$(1-q_{\text{decay}})^{2K} < 0.05,$$

which is sufficiently small such that we may approximate $W_t(1,3) - W_t(1,2)$ using Equation (1.3). Thus, due to exponential decay, at time nK, we can ignore X_m for $m < (n-1)K$.

We have now established that we need to consider only the choices made between times $(n-1)K$ to nK. Consider the worst-case scenario, where all choices made in this interval reinforce the bad edge. Specifically, $X_m = -1$ for all $(n-1)K \leq m < nK$. We next want to determine how much pheromone needs to be placed on the good edge in the next K steps to flip the probabilities. At time $(n+1)K$, the contribution of the choices made in this interval will be at least:

$$a = \sum_{m=(n-1)K}^{nK} -1 \cdot (1-q_{\text{decay}})^{(n+1)K-m} \leq \sum_{m=(n-1)K}^{nK} X_m \cdot (1-q_{\text{decay}})^{(n+1)K-m}$$

Thus, to flip the probabilities within the next K steps, the ants need to make enough contribution to the good edge such that:

$$\sum_{i=nK}^{(n+1)K} X_i \cdot (1-q_{\text{decay}})^{(n+1)K-i} > a.$$

Let S represent the number of times the ants chose the good edge in the interval $nK \leq t < (n+1)K$, i.e.:

$$S = |\{t : nK \leq t < (n+1)K, X_t = +2\}|$$

Let $(K-S)$ represent the number of times the ants chose the bad edge in this interval. Then, in the worst case, the number of times the good edge needs to be taken compared to the bad edge to flip the

probabilities is:

$$\begin{aligned} 2S - (K - S) &\geq |a| \\ S &\geq \frac{K + |a|}{3}. \end{aligned}$$

Thus, the probability that the edges flip in the next K steps is at least:

$$\text{Pr}(nK \leq T \leq (n+1)K) \geq \text{Pr}\left(S \geq \frac{K + |a|}{3}\right)$$

The right-hand side of the equation above is approximately binomial with parameters $p = q_{\text{explore}}, n = K$, which can be further approximated by the normal distribution with parameters

$$\begin{aligned} \mu &= q_{\text{explore}} \\ \sigma^2 &= \frac{q_{\text{explore}}(1 - q_{\text{explore}})}{K} \end{aligned}$$

Note that in deriving S we ignored exponential decay between times nK and $(n+1)K$, so that we may treat the X_i in a small window of time as i.i.d. and apply a binomial approximation. It is not obvious how to compute $\text{Pr}(nK \leq T \leq (n+1)K)$ with exponential decay included. Below, we find that this binomial approximation is still sufficient to provide an upper bound, and then we illustrate why it is crucial that the ants do not lay pheromone on the way back from the dead-end.

Taking $q_{\text{explore}} = 0.20, K = 75$ we get that:

$$\text{Pr}\left(S \geq \frac{K + |a|}{3}\right) \approx 0.00011$$

Thus, the probability that the edges flip probabilities within nK and $(n+1)K$ is:

$$\text{Pr}(nK \leq T \leq (n+1)K) \geq 0.00011.$$

If we think of each block of K steps as an independent trial with success probability $p \geq 0.00011$, then we can think of T as an approximately geometric random variable with parameter $p \geq 0.00011$. The expected value of T is then:

$$E[T] \leq K \cdot (1/p) \approx 75 \cdot 9104 = 682800$$

Informally, the exponential decay essentially gives the process an approximately memoryless prop-

erty by nearly erasing the results of decisions that were sufficiently far in the past. To succeed, the ants simply need one block of K steps in which they take edge $(1, 3)$ more often than expected to push $W_t(1, 3)$ above $W_t(1, 2)$ and flip the probabilities. Note that this is the worst case possible, i.e., when the previous 75 time-steps all reinforce the bad edge, which is itself highly unlikely.

Next, we show that if the ants lay pheromone on the way back from the dead-end, then the expected value of T is significantly higher. Formally, if we eschew the condition that ants do not lay pheromone on the way back from a dead-end, then both the good and the bad edges increase their edge weight by 2 when traversed. Then the number of times S the ants need to chose the good edge in the interval $nK \leq t < (n+1)K$ is:

$$\begin{aligned} 2S - 2(K - S) &\geq a \\ S &\geq \frac{a + 2K}{4} \end{aligned}$$

For $q_{\text{explore}} = 0.20, q_{\text{decay}} = 0.02, K = 75$, we get that

$$Pr\left(S \geq \frac{2K + |a|}{4}\right) \approx 6.17 \cdot 10^{-13}$$

and

$$E[T] \leq K \cdot \frac{1}{p} = \frac{75}{6.17 \cdot 10^{-13}} \approx 1.22 \cdot 10^{14}.$$

Thus, the number of successes the ants need in a span of K steps is significantly higher, and so the chances of any one block of K steps flipping the probabilities is much lower and takes much more time.

In the unlikely event that the two edges ever have equal weight, the ant picks between the two edges with equal probability. With probability 0.5 it takes the good edge at which point the good edge has higher weight, and thus the break is fixed. With probability 0.5 the ant takes the bad edge at which point all of the above analysis may be repeated.

We performed 50 simulations of 1000 steps on the Minimal graph in which ants do lay pheromone on the way back from the dead-end. We find a 0% success rate, supporting our theoretical insights.

Necessity of queueing: The theoretical argument above provides intuition for why queuing is necessary for pruning a dead-end. The smaller the sample size, the more likely that enough choices will deviate from expectation toward the good edge to flip the probability toward taking the good edge. When ants queue, this decreases the sample size because fewer ants make a choice in each time step. This decrease in sample size then increases the probability that in a given time interval, ants choose the good

edge over the bad edge. When ants do not queue, they make more choices in each time step, and so more time is required to achieve the deviation toward choosing the good edge over the bad edge. If there is no queuing, then initially only a fraction of ants are at the dead end, and the sample size argument above does not necessarily hold at the junction. Eventually however, all of the ants in the graph eventually approach the junction between the good and bad edge, to the point where our sample size argument above eventually takes hold.

Formally, suppose there are 100 ants. If there is no queuing, then 100 ants will make a choice at each time step, rather than 1 ant. If we follow the argument above, S would still be approximately normally distributed, but its variance would be $\frac{q_{\text{explore}}(1-q_{\text{explore}})}{100K}$, and thus Pr $\left(S \geq \frac{K+|a|}{3}\right)$ becomes much smaller because S is much more tightly distributed around its mean. This theoretical intuition is confirmed by simulation using 1000 steps. If the queuing mechanism is removed, there is a 0% success rate on the Minimal graph.

1.3.5 Q3. Maintaining a trail in the absence of a break

Here we consider whether the same algorithm used to repair a path can also keep a path intact when it is not broken. This is important because if different algorithms were used to maintain trails versus repair trails, then the turtle ants would need some signal to toggle between different methods for choosing among candidate edges, depending on the context. We found that a single algorithm, *RankEdge*, is capable of maintaining trails and responding to breaks.

In particular, we ran the *RankEdge* algorithm on the Spanning grid without breaking the original path and found that the trail was preserved without any modification to the algorithm or its parameters (Figure 1.11). In contrast, the *Weighted* algorithm performed very poorly on this task. In particular, for *RankEdge*, the path entropy using the MLE parameter values from turtle ant data was optimal (0.00). For *Weighted*, however, the path entropy for the MLE parameter values was much higher (5.38), indicating poor maintenance of the original path.

1.3.6 Q4. Pruning as a general principle for discovering alternative paths

Field observations show that turtle ants engage in pruning (Figure 1.12). In our simulations, we also observed that ants explored multiple alternative paths, and then most of these paths were pruned as the colony converged to a single alternative path. Further, the paths tended to become shorter over time. We quantified how many paths were pruned during the simulation using a measure called *path elimination* (Methods). We also quantified how the lengths of the remaining paths changed over time using a measure

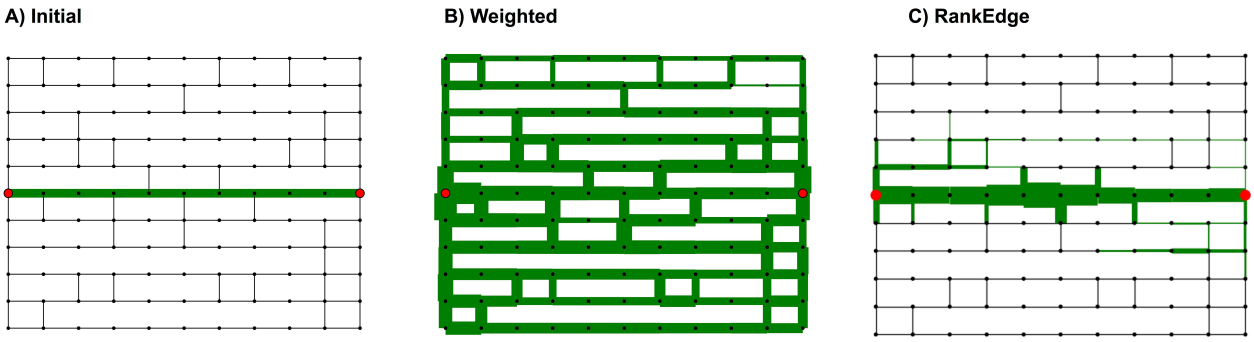


Figure 1.11: **Analysis in the absence of a break.** A) Initial Spanning grid, with no break. B) The final network produced using *Weighted*, which does not find a low entropy solution. C) The final graph using *RankEdge*, which finds a low path entropy solution.

Table 1.5: **Path elimination.** For each algorithm, we show the average reduction in entropy over chosen paths over time (Methods). We omit the Minimal graph, because there is only one possible path from one nest to the other, and thus no path elimination is possible.

Algorithm	Simple	Medium	Full Grid	Spanning Grid	European Roads
Weighted	0.94 ± 0.58	0.48 ± 0.32	0.00 ± 0.00	0.001 ± 0.002	0.14 ± 0.30
RankEdge	0.18 ± 0.26	0.25 ± 0.29	0.14 ± 0.20	0.15 ± 0.18	0.10 ± 0.16
MaxEdgeA	0.54 ± 0.84	0.69 ± 1.14	0.26 ± 0.49	0.78 ± 1.37	0.01 ± 0.01
MaxEdgeB	0.25 ± 0.61	0.66 ± 1.06	0.29 ± 0.56	0.84 ± 1.73	1.18 ± 2.44
MaxEdgeC	0.40 ± 0.78	1.29 ± 1.40	0.48 ± 0.92	1.03 ± 1.58	0.04 ± 0.07
MaxWeighted	0.44 ± 0.78	1.04 ± 1.23	0.68 ± 1.43	1.36 ± 2.93	0.38 ± 0.46
Deneubourg	0.43 ± 0.56	1.60 ± 0.96	0.55 ± 1.45	1.40 ± 2.36	1.47 ± 2.95

Table 1.6: **Path length pruning.** For each algorithm we show the average reduction in lengths of chosen paths over time (Methods) observed. We omit the Minimal graph, because there is only one possible path from one nest to the other, and thus no pruning is possible.

Algorithm	Simple	Medium	Full Grid	Spanning Grid	European Roads
Weighted	0.19 ± 0.29	0.32 ± 0.47	0.09 ± 0.47	0.44 ± 1.74	0.03 ± 0.08
RankEdge	0.30 ± 0.58	0.39 ± 0.58	0.04 ± 0.14	0.97 ± 2.59	0.35 ± 0.50
MaxEdgeA	0.30 ± 0.27	0.28 ± 0.30	0.11 ± 0.17	0.15 ± 0.20	0.05 ± 0.10
MaxEdgeB	0.14 ± 0.23	0.15 ± 0.23	0.06 ± 0.10	0.11 ± 0.16	0.08 ± 0.14
MaxEdgeC	0.31 ± 0.29	0.50 ± 0.36	0.12 ± 0.17	0.13 ± 0.21	0.08 ± 0.13
MaxWeighted	0.29 ± 0.30	0.39 ± 0.33	0.15 ± 0.23	0.19 ± 0.24	0.17 ± 0.23
Deneubourg	0.46 ± 0.17	0.42 ± 0.28	0.001 ± 0.003	0.04 ± 0.09	0.02 ± 0.09

called *path length pruning* (Methods). All of the non-linear algorithms exhibit some path elimination and pruning, and thus could plausibly be used to explain the observed pruning in observed turtle ants. *RankEdge* prunes fewer paths than the other algorithms (Table 1.5) because *RankEdge* does not form as many initial paths as other algorithms. However, of the paths that are pruned, *RankEdge* tends to prune more nodes from the paths (Table 1.6).

For every network, we compared the average path elimination and path length pruning of *RankEdge*

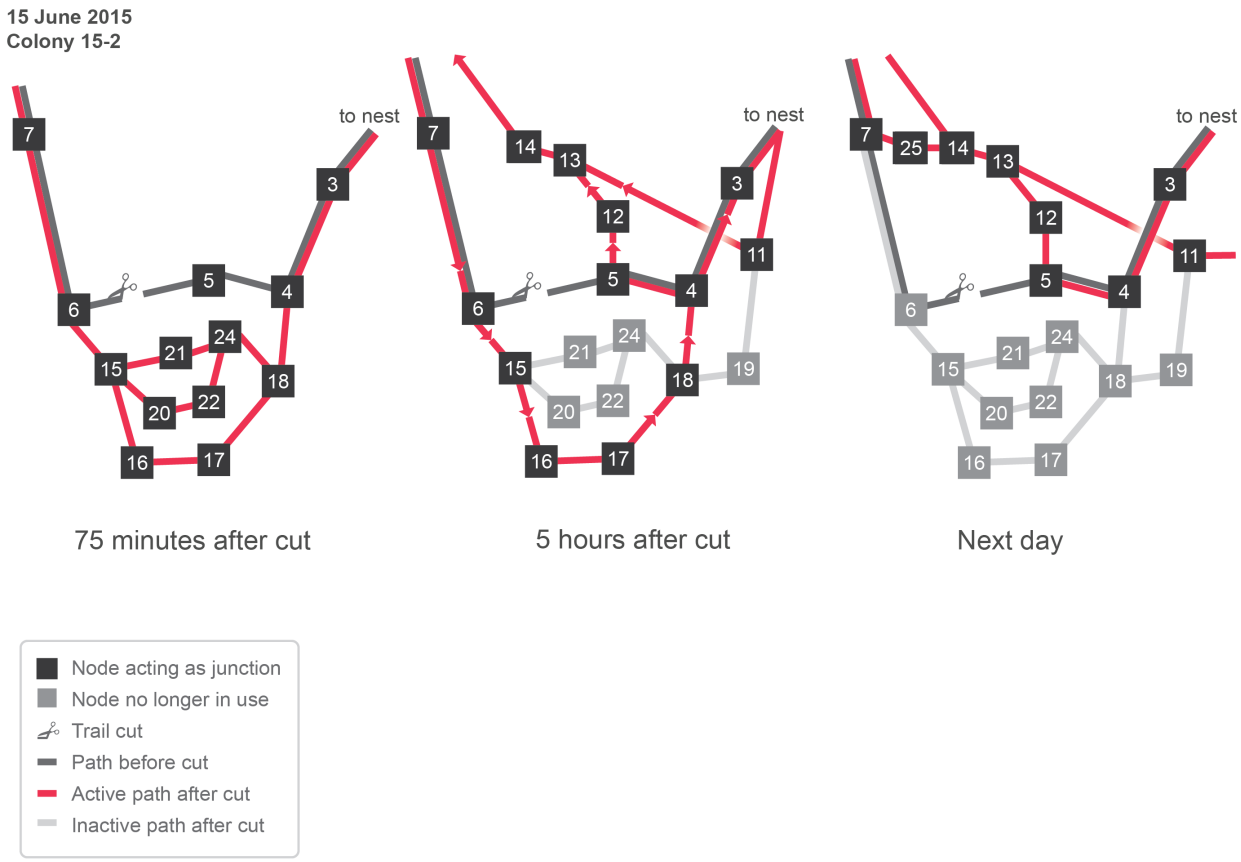


Figure 1.12: **Turtle ants prune paths.** The diagram from [3] shows the results of an experiment in which an edge was cut. Left: The initial trail is shown in grey. The edge connecting nodes 5 and 6 was cut. After 75 minutes, the turtle ants explored several new paths (red). Center: Five hours after the cut, some of the red paths were pruned (transparent grey). Ants traveling down from node 7 took one trail, consisting of nodes 6, 15, 16, 17, 18, 4, and 3, because they could not use 12 in this direction. Ants traveling in the other direction took another trail, consisting of nodes 3, 4, 5, 12, 13, and 14, or the trail consisting of 11, 13, and 14. Right: The next day, there was additional pruning. Because node 12 could now be used in both directions, ants traveled both ways on the indicated trail.

versus every other algorithm using Welch's unpaired T-test. For path elimination, *RankEdge* does not reduce the number of paths more than other algorithms ($p < 0.05$), as we described above. However, for path length pruning, *RankEdge* reduces path lengths significantly more than all other algorithms on the Spanning grid and European roads ($p < 0.05$). On the Full grid, *RankEdge* is significantly better than *Weighted* ($p < 0.05$), is not significantly different from *Deneubourg* or *MaxEdgeB*, and is significantly worse than *MaxEdgeA*, *MaxEdgeC*, and *MaxWeighted*. No statistical difference is observed for the Simple and Medium graphs, likely due to their relatively simple topology. These pruning results suggest that *RankEdge* explores fewer paths initially but is better at selecting the shortest of the paths it explores. *RankEdge* tends to prune fewer paths (path elimination), but of the paths it does explore, *RankEdge* converges to the shorter paths (path length reduction).

Field observations [3] also showed that when ruptured trails are repaired, new nodes are added to the network, and in subsequent days, some of the nodes are pruned. Such pruning of nodes also led to global pruning of paths (Figure 1.12). Such an "explore-exploit" strategy may help turtle ants quickly find a solution that re-connects a rupture in a trail, and may also help the colony to optimize the coherence of the trail, by minimizing the number of junctions at which ants could get lost.

Interestingly, using pruning-based strategies to discover the most appropriate edges or paths to keep is a common strategy used by biological systems. In particular, during the development of neural circuits in the brain, synapses are massively over-produced and then pruned-back over time [61]. This strategy is thought to help neural circuits explore possibly topologies and then converge to the most appropriate topology based on environment-dependent feedback. A similar process occurs during the development of vascular (blood flow) networks in the body [117]. Thus, pruning may be a common biological strategy of network design when multiple topologies need to be explored in a distributed manner.

1.4. Discussion

Our primary contribution is to address an engineering problem (maintaining a trail network and finding alternative paths to route around broken links in the network) using biologically feasible parameters and models motivated by how turtle ants may solve this problem in the field. Successful performance by the algorithm in simulation, using realistic parameter values, indicates that the algorithm is a plausible candidate to describe how the ants create their networks. The *RankEdge* algorithm achieved a better maximum likelihood estimate (Figures 1.2–1.4) than every other non-linear model except for *MaxEdgeA*. When parameterized by data from field observations, *RankEdge* was better able to find a single, short path with high probability compared to other algorithms (Tables 1.2,1.3, 1.4). From this, we conclude that non-linear models, in particular *RankEdge* and *MaxEdgeA*, represent the two best plausible models of turtle ant behavior.

By testing performance across six different networks, we found that the turtle ants appear to have evolved an algorithm that may not be optimal for any particular planar network but is robust to some variation in the topology. Further, non-linear algorithms exhibited pruning, which also occurred in field observations [3] (Table 1.6, Figure 1.12).

There are several features of our algorithm that are critical for success in repairing breaks to the routing backbone. First, to minimize path entropy it is essential to have a stronger-than-linear bias towards choosing the highest-weighted edge (*RankEdge*), rather than choosing edges proportional to their edge weight (*Weighted*). This helps constrain the search space and leads to better convergence to a single consensus path. We emphasize that *Weighted* has a strong success rate because pheromone is left on

every edge in the graph (see e.g., Figure 1.7A–B). This guarantees that there exists some path with positive probability. However, *Weighted* does not commit to a single path as effectively as an algorithm with a strong, non-linear bias toward the highest-weighted edge, such as *RankEdge*. The path entropy of *Weighted* is high because essentially every path in the graph has high probability, and the average path length is high because the algorithm does little to eliminate long paths and commit to short paths. Second, to repair breaks it is essential to use bi-directional search and avoid backtracking. Observations show that when turtle ants encounter a break, ants from both sides of the break attempt to repair the trail [3]. When ants from both sides meet, they each encounter a trail that is already strongly reinforced and guided towards the other nest. In addition, the ability to avoid backtracking allows ants to avoid going back and forth along the same edge. Third, we showed theoretically that the time needed to find an alternative path decreases significantly if turtle ants reaching a dead-end in their trail do not leave pheromone while returning back from the dead-end.

The *RankEdge* algorithm is parsimonious, capable of both maintaining trails and repairing breaks to trails using the same underlying logic. Observed ants encounter diverse situations analogous to breaks in the ongoing maintenance of trails. We find that a single algorithm can solve two diverse problems without requiring the additional complexity of a signal that distinguishes such situations from a rupture in the trail. How each path is established originally is an interesting yet distinct question. Paths are not always the shortest globally, and the physical structure of edges in the canopy appears to affect how these paths are selected.

The algorithm can be extended to improve performance, though this may involve sacrificing biological realism. One possible extension would allow ants to "toggle" between different parameter values or algorithms in different situations. For example, an ant could use *RankEdge*, but if it encounters a dead-end or massive crowding (determined for example by a large increase in the frequency of antennal contacts with other ants [18, 72]), then it increases its probability of exploring new edges. This would be similar to a distributed version of simulated annealing, with the value of q_{explore} corresponding to the decreasing value of the temperature parameter. A second possible extension would be to use multiple types of pheromone [118]. Ants could use negative pheromone to signal to other ants not to select a certain edge, for example, towards a dead-end. Further work is needed to measure the computational abilities of turtle ants to determine whether such extensions depart from biological realism.

There are some differences between our synthetic networks and the environment of the observed turtle ants. First, turtle ant trail networks in the canopy are 3D planar networks, whereas here, to begin the investigation of arboreal ant trail networks, we used 2D planar networks. The ideal test case would be a suite of synthetic networks that are isomorphic to some portion of the turtle ant canopy. In lieu of this, we describe five synthetic networks, some of which have been used by prior work, that each collectively

test the ability of different algorithms to solve the network repair problem. These five networks comprise a necessary (if not exhaustive) set of test cases. Second, in the tropical forest, many edges are physically difficult to traverse, which may provide natural inhibition for selecting certain edges. Third, in the canopy, edges are not all of the same length. In future work that includes variability in edge lengths, synchronous walks will need to be modified since longer edges require more time-steps to traverse. More generally, further work is needed to determine the physical properties of junctions and branches in the canopy and how these properties influence the likelihood of traversing an edge.

Finally, the probabilistic *RankEdge* algorithm is biologically feasible, requiring less computational complexity and assuming fewer memory requirements than many other distributed graph algorithms commonly used in computer science. This suggests that a biological algorithm evolved to deal with the constraints of the tropical forest canopy may be useful in other applications, such as in swarm robotics or molecular robots [113, 12, 114, 115]. For such applications, the best algorithm to choose depends on the requirements of the problem. We find evidence that *RankEdge* is the best algorithm to achieve relatively high success rate and low path entropy. If agents do not need to adhere to a single path, then the *Weighted* algorithm performs better, though the length of the path may be long. It appears that turtle ants use an algorithm that finds a single short path [3], as *RankEdge* provides. For trivial graphs with only one alternative path, both algorithms perform similarly.

Overall, our work contributes to the growing intersection of distributed algorithms used by natural biological processes [27, 67].

1.5. Methods

1.5.1 Maximum likelihood estimation of parameter values

For each candidate algorithm, we varied $q_{\text{decay}}, q_{\text{explore}} \in (0, 1)$ and evaluated the likelihood that the algorithm with a specific set of parameter values would have generated the choices made by turtle ants observed in the field. The edges traversed by the turtle ants, and times the edges were traversed, were used to compute how much pheromone had been added to and had decayed from each edge, to give the amount of pheromone on each edge at any time. In modeling pheromone decay, we treat q_{decay} as the rate of decay per second. When computing the amount of decay between two consecutive ant choices, we decay all of the edges in proportion to the number of seconds elapsed between the two choices. For each candidate algorithm, if we know all of the edge weights at a given time and the value of q_{explore}, we can compute the likelihood of a given choice. Figure 1.2A–B provides an example of a calculation of the

likelihood of a choice for each candidate algorithm.

For a given combination of $q_{\text{decay}}, q_{\text{explore}}$ we performed this likelihood computation for every observed choice in each of the 13 junctions. We updated the edge weights based on the choice and the amount of time that passed between successive choices, and then repeated this process on the next choice made by the next ant. For each junction of observations at a given node on a given day, we computed the maximum likelihood estimate (MLE) for each parameter value pair. We then added the log-likelihoods for all the 13 junctions.

As we formalize in section 1.3.4.8, the exponential rate of pheromone decay means that the most recent ant choices have the largest effect on the current edge weights at a junction. Pheromone added far in the past will have largely decayed and will not contribute much to the current weights. Thus, we do not need an extensive history of the choices to perform accurate modeling. It is not currently possible to measure or manipulate pheromone levels on the branches in the canopy.

1.5.2 Additional technical details

Each algorithm includes the following constraints motivated by field observations:

1. Observations suggest that turtle ants tend not to backtrack, but instead tend to keep moving along the trail in the same direction, indicating that turtle ants have at least enough sense of direction to avoid going back and forth over the same edge. Our simulations include three exceptions to this. First, because our simulations include two nests with ants going back and forth, upon reaching the nest, an ant is allowed to backtrack along the same edge it used to reach the nest. Second, if a simulated ant reaches a dead-end node that has no outgoing edges other than the previously traversed edge, it is allowed to backtrack. However, the ant does not lay pheromone on the way back until it reaches a node with two edges, excluding the edge it previously traversed. In field experiments, it is difficult to determine whether a turtle ant is laying pheromone; however, it is known that *Lasius niger* ants down-regulate pheromone deposition at dead-ends to avoid recruitment during crowding [119, 118, 118]. It is possible that turtle ants similarly down-regulate pheromone in response to dead-ends. In section 1.3.4.8, we also provide a probabilistic argument for why it is critical that ants do not lay pheromone when returning from a dead-end to repair the break. Thus, in addition to the ability to avoid backtracking, each ant requires one binary state variable that is 0 or 1 depending on whether the ant is coming back from a dead end.

2. Turtle ants queue at a node and leave in a first-in first-out manner. In other words, if more than one ant is at the same node, only one ant chooses an edge in each time-step. In the field, turtle ants

walk along narrow branches, almost always one ant at a time in each direction. We find that queueing increases the success rate of the algorithm (section 1.3.4.8).

3. When turtle ants take an "explore step", they often traverse an edge for a short distance, and then return to the original node [3]. This builds a slight extension off the primary path, which can be extended by subsequent ants. In all algorithms, if a simulated ant takes an explore step, it goes across the edge and comes back in one time-step. Thus, two units of pheromone are left on the edge, and the ant is back at the node it started from. (R3-27) An explore step is defined as any choice that cannot occur unless $q_{\text{explore}} > 0$. For *Weighted*, this involves taking an edge with zero weight; for *RankEdge*, this involves taking an edge that does not have the highest weight. For *Unweighted*, there is no 'explore' step, because ants are not inherently biased towards following any particular pheromone trail.

4. Because pheromone decays exponentially, theoretically once an ant lays pheromone on an edge, that edge's weight will never decay to absolute 0. In practice, if the edge weight stays unchanged even after multiplying by the decay rate (due to numerical computation error, occurring at roughly 10^{-300}), then we reset the weight of that edge to absolute 0. Another possible approach would be to introduce a pheromone detection threshold parameter. If an edge had pheromone below this threshold, the ant would treat the edge as if it had no pheromone. We avoided this approach because it would introduce another parameter to optimize and compare.

5. All edges are assumed to have the same length, and it takes exactly one time step for an ant to cross an edge.

1.5.3 Performance metrics

Below we formally describe the three performance metrics used to evaluate each algorithm, after it ran for T time-steps. Intuitively, the simulated ants have successfully found a path if they can reach one nest from the other without taking any "explore steps". We thus measure the performance of each algorithm assuming $q_{\text{explore}} = 0$, and consider all paths that may be taken with positive probability under this constraint. For *RankEdge* and other non-linear algorithms, ants travel from one nest to the other by following the highest-weighted edges. For *Weighted*, ants travel from one nest to the other using only edges of positive weight.

Formally, let the *pheromone subgraph* be the subgraph induced by the two nest nodes, all edges with a nonzero weight, and all nodes adjacent to edges with non-zero weight. Let G be a pheromone subgraph and $P = (v_1, v_2, \ldots v_n)$ be a path in G, with nests v_1 and v_n. For a node $v_{i \neq 1} \in P$, define the candidate edges $C_P(v_i) = \{(v_i, u) \in E(G) : u \neq v_{i-1}\}$, i.e., the edges that the ant could take from v_i without

backtracking to its previous node. Let $v_i, v_{i+1} \in P$ be consecutive nodes in the path; we say edge (v_i, v_{i+1}) is *maximal with respect to* P if $w(v_i, v_{i+1}) = \max_{u \in C_P(v_i)} w(v_i, u)$. The path P is a *maximal path* if for every pair of consecutive nodes $v_i, v_{i+1} \in P$, the edge (v_i, v_{i+1}) is maximal with respect to P. An ant taking a maximal path always takes an edge with the highest weight; thus, a maximal path allows an ant using one of the non-linear algorithms to commute between two nests with positive probability even if $q_{\text{explore}} = 0$.

Next, define a *pheromone path* to be a path in which all edges have positive weight. Such a path allows an ant following the *Weighted* algorithm to commute between two nests with positive probability even when $q_{\text{explore}} = 0$.

For a given algorithm, define a *solution path* to be a path that can be traversed with positive probability under that algorithm when $q_{\text{explore}} = 0$. For all the non-linear algorithms discussed here, solution paths are equivalent to maximal paths. For the *Weighted* algorithm, a solution path is equivalent to a pheromone path.

Let $\hat{p} = (p_1, p_2, \dots), \sum_i p_i = 1$ be a probability distribution. Define the *entropy* of the distribution to be: $S(\hat{p}) = -\sum_i p_i \log(p_i)$.

At the end of the simulation, we evaluate the pheromone subgraph of each algorithm by computing the following measures:

- **Success rate (higher is better):** The probability that the ants form a solution path. This is defined empirically by computing the percentage of the N simulations where a solution path is formed in the final graph.
- **Path entropy (lower is better):** An information-theoretic measure of how well the ants converge onto a single solution path.
  - Let $M_1, M_2, \dots, M_n$ be the set of all n solution paths in the final graph.
  - Let $p_1, p_2, \dots, p_n$ be the probabilities of taking each solution path with $q_{\text{explore}} = 0$. The probabilities $\hat{p} = (p_1, p_2, \dots, p_n)$ form a probability distribution.
  - The path entropy is then: $S(\hat{p})$.
- **Average path length (lower is better):** The average length of the solution paths in the final graph. Path length is defined to be the number of nodes in a path.

To compute the pruning metrics, we first define a *chosen path* as the sequence of nodes $v_1, v_2, \dots, v_n$, after removing cycles, that an ant takes to successfully walk from one nest to another. Figure 1.13 illustrates why removing cycles is necessary when comparing chosen paths.

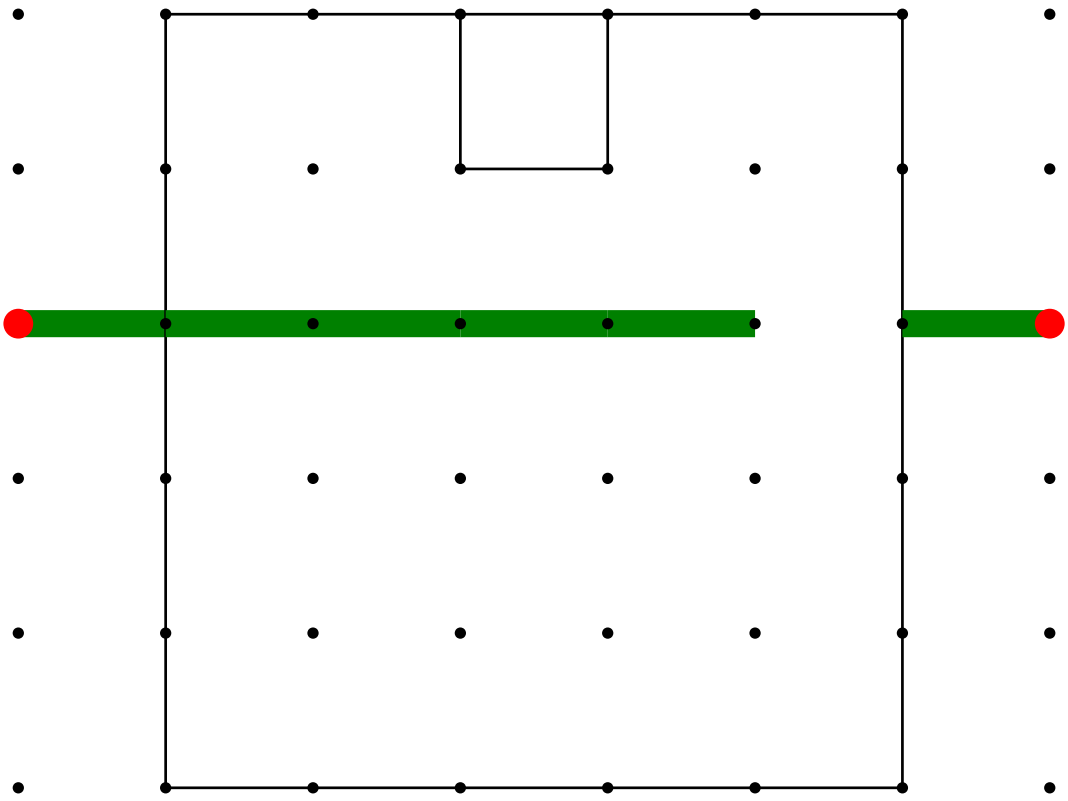


Figure 1.13: **Motivation for removing cycles.**

Over the course of the simulation, we track all chosen paths for all ants that successfully walk from one nest to the other. This includes the number of times each path was chosen — and thus the distribution over the chosen paths — and the lengths of these paths.

- **Path elimination:** An information-theoretic measure of the degree to which paths from one nest to the other are eliminated over time.
  - Let $S_t = S(\hat{p}_t)$ be the entropy over the distribution of chosen paths $\hat{p}$ that have been completed at or before time t.
  - Let $S_{max} = \max_{1 \leq t \leq T} S_t$ be the maximum chosen-path entropy over the entire simulation.
  - The path elimination is then the maximum entropy minus the entropy at the end of the simulation: $S_{max} - S_T$.
- **Path length pruning**: A measure of the degree to which ants reduce the lengths of the paths they take over time.

- Suppose at time t the ants have taken chosen paths $p_1, p_2, \ldots, p_n$ with frequencies $c_1, c_2, \ldots, c_n$. Let $l(p_i)$ be the length of path p_i. We define the *weighted-mean chosen path length* at time t to be: $L_t = \frac{\sum_i c_i \cdot l(p_i)}{\sum_i c_i}$.
- Let $L_{max} = \max_{1 \leq t \leq T} L_t$ be the maximum weighted mean chosen path length over the entire simulation.
- The path length pruning is then: $L_{max} - L_T$.

Robustness across network topologies: To determine which parameter values performed well across all the planar topologies tested, we defined the *robustness* of a set of parameter values ($q_{\text{explore}}, q_{\text{decay}}$) to be the geometric mean of the success rates for those parameter values on all six networks. We use the geometric mean because it penalizes parameter values that perform poorly on any one particular graph; for a set of parameter values to have a high geometric mean, it must perform well on every graph. When computing robustness, we weight the success rates over all networks equally. This is done for two reasons: first, this highlights algorithms that perform well under a variety of distinct but equal conditions; and second, we do not currently have complete data on which topologies are more or less likely to occur in the canopy, and thus it is not clear how weighting factors should be selected.

Application to the European road network: We sampled a portion of the European road network. This sample contained the same number of nodes as the Full grid ($11 \times 11 = 121$ nodes). Sampling was done by selecting a random node and performing a breadth-first search until 121 nodes were visited. The network contained these 121 nodes and all the edges adjacent to these nodes. We then randomly selected two nodes and removed a randomly-chosen edge in the shortest path between those nodes. If removing this edge disconnected the two nodes, we discarded the pair of nodes and picked a new randomly chosen pair of nodes. We then applied our algorithm to repair the trail.

1.5.3.1 Removing cycles when comparing chosen paths

Figure 1.13 shows that ants that take the top path are all using the same core path, even if some ants go around the loop more times than others. To account for this, we removed cycles when computing and comparing chosen paths. An alternative would be to consider each path that takes a different number of loops unique. This, however, is limiting. For example, two paths that exactly overlap except for the number of loops used would be considered equivalent to two paths that have zero edges in common. We wanted to avoid having to set an arbitrary threshold on how "similar" two paths need be before they are considered the same, and thus we simply removed all cycles from chosen paths.

1.6. Source code and datasets

All source code for the algorithm and all datasets for the ant choices are available at our Github repository: `http://github.com/arjunc12/Ants`.

1.7. Acknowledgements

Chapter 1, in full, is a reprint of material as it appears Scientific Reports in 2018. Chandrasekhar, Arjun; Gordon, Deborah M; Navlakha, Saket, Nature Research, 2018. The dissertation author was the primary investigator and author of this paper.

Chapter 2

Better Tired than Lost: Turtle Ants Prioritize Coherence over Shortest Paths.

In the previous chapter I showed how turtle ant behavior can be used to derive a distributed algorithm for repairing breaks in a trail network. The next step is to try to reverse-engineer the algorithm that turtle ants use to construct the trail network in the first place. To do this, I start by exploring what design principles guide the construction of the foraging network. I provide evidence that turtle ants take advantage of environmental variation when foraging (while demonstrating how an algorithmic model gives us a framework to describe and test such ecological hypotheses), suggesting possible ways in which human-designed swarm robots may be improved.

While most of my work involves computational analysis, for this chapter part of my contributions involved field work. In particular, I assisted in the collection of data on trail networks formed by turtle ants. This was an eye-opening experience in several ways. First, seeing how the field work took place gave an appreciation for the practical difficulties involved in data collection. Ants are fickle, and they may not be be wholly amenable to exploring a newly created junction. Great care and delicacy must be taken to not damage a branch or startle the ants. Even if the experiment works to perfection, human-annotated data can be error-prone. This experience gave me a better understanding of how to design feasible experiments and more broadly, what ecology 'is' and how to better communicate with ecologists.

While field work can be physically and mentally tiring, it can be equally illuminating. It was quite

humbling to see the degree to which turtle ants outperformed their simulated counterparts. Turtle ants are not mindless agents following a rote process; they are constantly taking stock of their environment, interacting with peers, and making decisions on the fly, such as turning around in response to a long branch or a newly-inserted metal wire that bears no resemblance to the vegetation. The social aspect of the ants was especially sorely missing from the simulations. Overall, seeing the ants in action will give me more intuition for how the ideal simulations 'ought' to behave.

Abstract

Creating a routing backbone is a fundamental problem in both biology and engineering. The routing backbone of arboreal turtle ants (*Cephalotes goniodontus*) connects many nests and food sources using trail pheromone. Unlike species that forage on the ground, arboreal ants are constrained to form trail networks along branches and vines within the vegetation. We examined what objectives the ant networks meet by comparing the observed turtle ant trail networks with alternative networks of random, hypothetical trails in the same surrounding vegetation. We found that turtle ant trail networks favor coherence, keeping the ants together on the trails, rather than minimizing the distance traveled along edges in the graph. The ants' trails minimized the number of nodes traversed, reducing the opportunity for ants to get lost at each node, and favored nodes with 3D configurations most easily reinforced by pheromone, reducing the opportunities for ants to diverge onto different paths. Thus, rather than forming the shortest paths, the ant networks take advantage of natural variation in the environment to promote the maintenance of a coherent trail that ensures that ants stay connected along the routing backbone.

Keywords: Cephalotes, ant trail network, routing networks, foraging, distributed algorithms, search, exploration, shortest path, spanning tree

2.1. Introduction

Many engineered systems rely on a backbone routing network, whose goal is to ensure that any two entities or devices on the network can communicate through some path [1, 17, 120]. Some biological systems, such as neural arbors [121], plant arbors [122], and slime molds [60], also use routing networks, to transmit information and nutrients. Effective design of routing networks depends on the physical environment, because variation in the environment can affect the accuracy and rate of communication in both engineered [123, 10, 124] and evolved natural networks [125, 126, 127, 128]. The environment influences how the system chooses search strategies, prioritizes competing objectives, and coordinates its local decisions. For example, wireless networks operating in difficult to reach environments may use different routing strategies to minimize energy consumption of devices [123]; similarly, in bacterial navigation, chemicals appearing as localized pulses in the environment can affect gradient sensing and movement patterns [127].

The 14,000 species of ants have evolved diverse distributed routing algorithms to search for, obtain, and distribute resources [129, 67] in diverse environments [72, 19, 69, 4, 130]. Models of engineered routing networks inspired by ants often emphasize the goal of minimizing the distance traveled. Ant colony optimization (ACO), first proposed in 1991, loosely mimics ant behavior to solve combinatorial optimization

problems, such as the traveling salesman problem [34, 83, 37] and other such routing problems [131]. In ACO, individual ants each use a heuristic to construct candidate solutions, and then use pheromone to lead other ants towards better solutions. Recent advances improve ACO through techniques such as local search [84], cunning ants [36], and iterated ants [85].

A fascinating recent area of biological research examines the goals met by the trail networks of ants [130]. Studies of species that forage on a continuous 2D surface [69], including Pharaoh's ants [54], Argentine ants [19, 70, 71], leaf-cutter ants [72], army ants [73, 132], red wood ants [74], and meat ants [133, 134], show that ants use local chemical interactions to form trails [75, 76, 73], regulate traffic flow [135], search collectively [136], and form living bridges [137].

There are many objectives that an ant colony's trail network might meet, including minimizing energy costs by reducing the distance traveled, keeping the ants together to form a coherent trail, resilience to rupture, and effective searching [138]. Ant species that forage and build trails on the ground have few constraints on trail geometry because their trails can form nodes and edges anywhere on the 2D plane. Prior work [102, 71, 138] showed that ground-living ants, such as red wood ants and Argentine ants, may minimize the distance traveled by forming trails with branch points that approximate 2D Steiner trees [19, 139, 140]. However, minimizing distance may not be the only objective that ant trail networks attempt to optimize. Army ants link their bodies to form a bridge across gaps, but may not form the shortest possible bridge if this requires the use of more ants [77]. Meat ants form trail networks that link nests and trees as nodes, and their choices of which nodes are linked, as well as the direction and length of trails, suggest that robustness to the loss of a node is as important as minimizing the distance traveled [134, 133].

Many ant species operate in 3D environments, such as arboreal ants that nest and forage in trees and bushes. Unlike species that have evolved to create graph structures in continuous space in an unconstrained 2D plane, arboreal ants must solve problems on a natural graph structure. They cannot form trails with nodes and edges at arbitrary locations; instead, they can use only the nodes and edges that are available to them. The arboreal turtle ant (*Cephalotes goniodontus*) nests and forages in the tree canopy of tropical forests [141]. A turtle ant colony creates a trail network in the vegetation that connects several nests, providing a routing backbone that must be maintained to allow resources to be distributed throughout the colony [56, 3]. Ants search off the backbone to find and create trails to ephemeral food sources. The colony modifies the trail from day to day, sometimes forming small alternative paths, or loops, of which one is eventually chosen. The colony repairs the backbone in response to frequent changes in the vegetation caused by plant growth and by ruptures made by wind or passing animals [3].

2.2. Results

2.2.1 Candidate objectives

Here we mapped trail networks of turtle ants in their natural habitat and computationally tested what objective functions these trails may be optimizing. We compared the observed networks with simulated random networks, to determine how well the observed networks meet three objectives, possibly in combination [134]:

1: Minimizing the distance traveled, which was measured as the average length of the edges in the trail network. This is equivalent to minimizing the total trail length for a fixed number of edges. Minimizing the distance traveled minimizes the energy cost of building the trail network. This distance, however, is not equivalent to the number of nodes traversed, as is often assumed in various optimization algorithms [142, 131, 39, 38], including our previous work on turtle ants [143], because in the vegetation, the lengths of edges vary; the distance between one node and another ranges from less than a centimeter to more than a meter (Table 2.1).

2: Minimizing the total number of nodes, which promotes the maintenance of a coherent trail by reducing opportunities for ants to get lost, but also reduces the opportunities for exploration and thus for finding new resources off the trail. The number of nodes was measured by counting the nodes traversed along edges used in the trail network. We previously studied how ants at a node select which transition through a node to traverse based on the rate at which volatile pheromone is deposited [143]. Simulation results were consistent with field observations [56] indicating that ants at a node have a constant probability of exploring, or taking a path that is not the one most strongly reinforced by pheromone, of about 0.2 per node [143]. Thus each node presents an opportunity for ants to get lost, and lost ants may lay pheromone trail that could lead other ants astray. Each node is also an opportunity to meet the ants of other colonies that make trails in the same vegetation. We test here the hypothesis, indicated by previous observations [3], that modification and repairs to the backbone tend to reduce the number of nodes over time.

3: Minimizing the difficulty of establishing a pheromone trail by finding trajectories through nodes that are most easily reinforced by pheromone. This objective also contributes to the creation and maintenance of a coherent trail. We hypothesized that the physical configuration of a node influences how likely are successive ants to cross the node in the same way, thus reinforcing it with volatile trail pheromone. The ants have short antennae that detect pheromone only locally, so the transition from one edge through a node to another edge is reinforced only when successive ants take exactly the same trajectory through the node (Figure 2.1). Nodes are variable 3D structures, ranging from a simple fork created by a branch in a

plant, to a cluster of entwined vines and branches from different plants. We used an arbitrary categorical index to estimate how many possible trajectories an ant could take from one edge through a node to another edge, and thus how likely a node is to be reinforced (Figure 2.1, details below). We tested whether the ants' behavior reflected this estimate. Our previous work [3, 143] did not take into account variation among nodes, and did not examine how trails are selected from among the many available alternative networks.

Preference for nodes more likely to be reinforced was measured as the average *transition index* of all transitions in the trail network. The transition index is a categorical value that ranged from 1, where successive ants are most likely to take and reinforce the same trajectory through the transition, to 4, where successive ants are least likely to take and reinforce the same trajectory through the transition (Figure 2.1). The value of the transition index was assigned based on a visual estimate of the number of different trajectories available for ants to traverse the node, drawing on previous observations of the flow of ants on different edges from a node (e.g., see Fig. 7 in [3]). Each transition through a node in the vegetation (e.g., edge $u \rightarrow v$ through node v to edge $v \rightarrow w$; Methods) was assigned a value of the transition index. A particular node v may have more than one transition index if there were many edges connected to that node.

2.2.2 Mapping and modeling turtle ant trail networks

To determine what objectives are optimized by the ants' choice of paths within the vegetation (Figure 2.2A), we mapped the trail networks that connected the nests and naturally occurring, ephemeral food sources of three colonies (Tejon 189, Tejon 446, Turtle Hill 460), for 10–15 days over the course of 6 weeks, in a tropical dry forest at La Estación Biológica de Chamela in Jalisco, Mexico. We visually tracked the path taken by the ants and identified each node or junction in the vegetation, where an ant had a choice among more than one edge in the direction it is traveling, as well as the edges between nodes. We measured the length of each edge. We assigned a transition index to each each transition from one edge through a node to another edge, to estimate how likely were successive ants to take the same trajectory through the node and thus reinforce it with pheromone. To evaluate how the ants choose nodes and edges from the options provided by the surrounding vegetation, we also mapped all nodes, measured all edges, and assigned transition indices, for all possible paths up to five nodes away from each node used by the ants. A trail network on one day is illustrated in Figure 2.2B–C.

We modeled the network of vegetation as a directed, weighted graph, $G = (V_G, E_G)$, where each junction forms a node, and edges represent stems or branches that connect one node to another. Edge weights correspond to physical length. Node weights, which are used in some variants of the Steiner tree problem [144, 145], correspond to transition indices. We modeled transition indices by converting G into

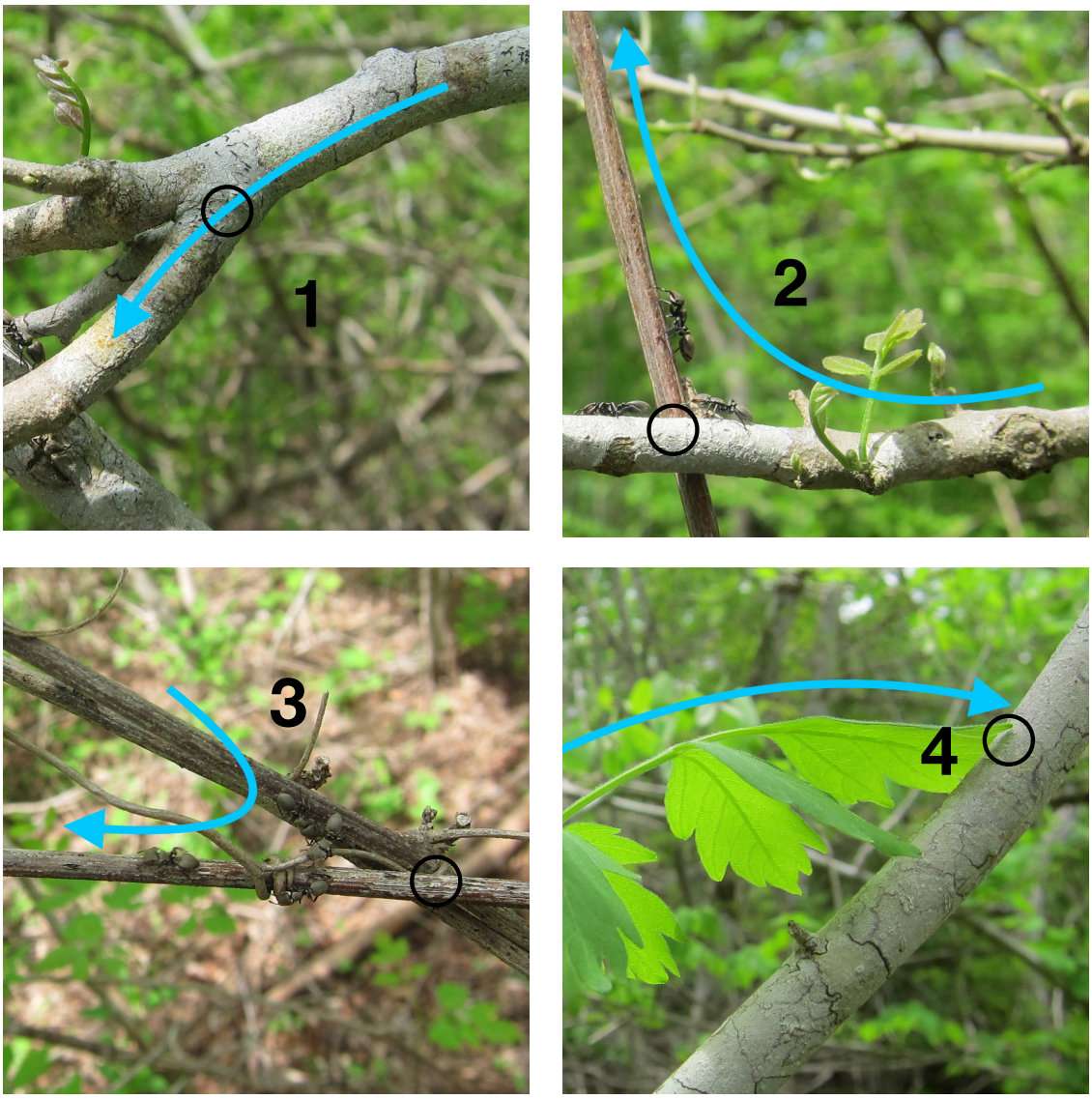


Figure 2.1: **Transition indices.** The photos show examples of nodes of each transition index (TI). The open black circle shows the node traversed. The blue arrow shows the transition, leading from the edge along which ants enter the node, through the node, toward the edge along which they exit. Each transition from an edge to a node to another edge was assigned a transition index with a value between 1 and 4. The lower the transition index, the more likely it is to be traversed by successive ants in the same way, and thus more likely to be reinforced. TI–1 (upper left): a node linking two edges on the same plant; in the example shown, all ants are likely to walk the same way across the top of the branch. TI–2 (upper right): a node that links one plant to another along a trajectory through a node that is likely to be the same for successive ants; in the example shown, most ants are likely to climb up the brown vine from the position shown by the ant approaching the junction from the left. TI–3 (lower left): a node that links one plant to another plant with more than one possible trajectory through the node; in the example shown, ants on the upper vine can reach the lower one either directly or by following the smaller vine that ants in the photo are using. TI–4 (lower right): a node that links one plant to another with many possible trajectories that are often changed by conditions; in the example shown, wind can easily move the leaf so that different ants reach the junction between the leaf and branch, at different places.

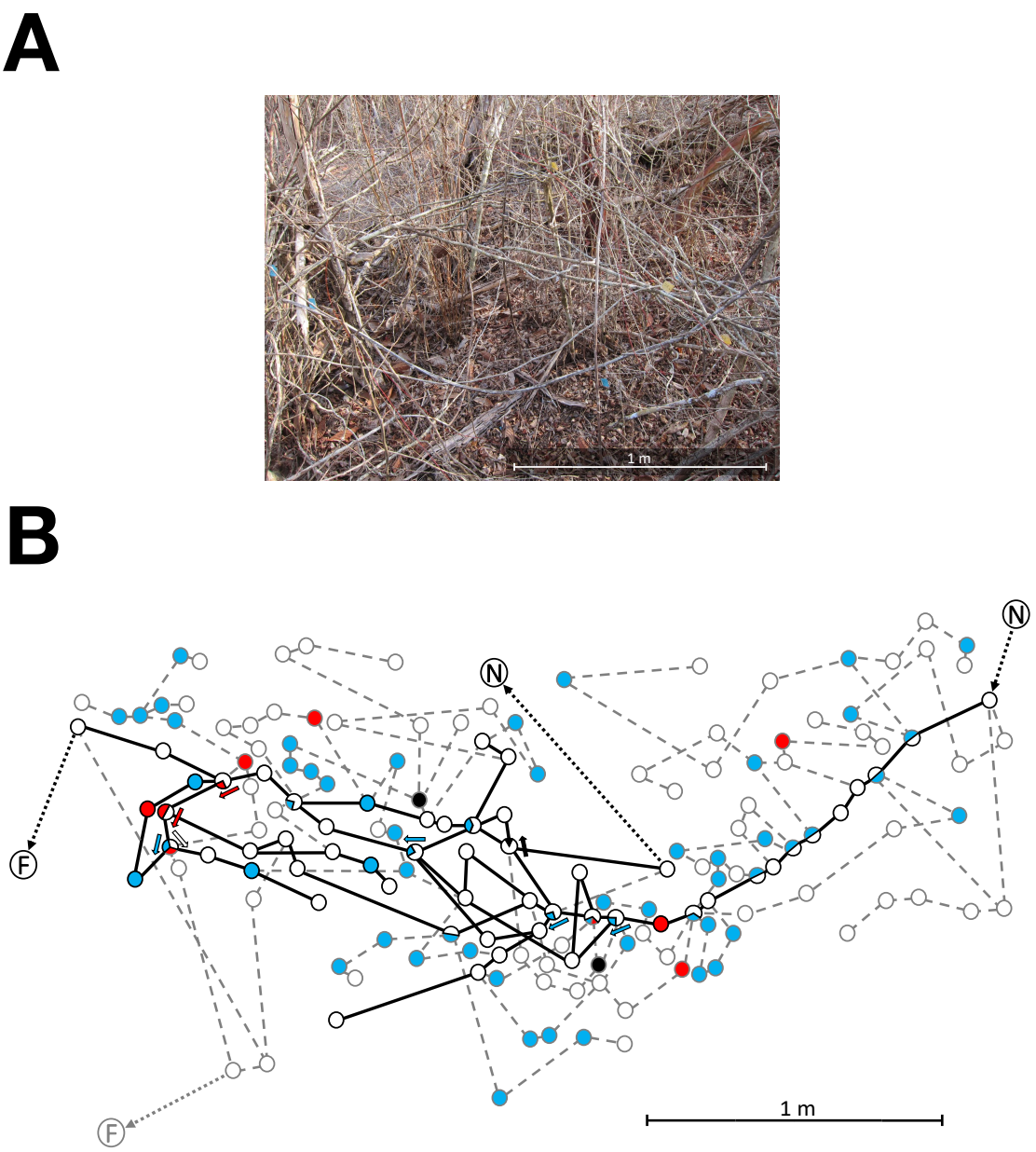


Figure 2.2: **Turtle ant vegetation.** A) Vegetation of the network mapped in B, photographed in the dry season before the branches have leaves. B) Illustration of part of the trail network for Tejon 189 on day 9. The figure shows 166 of the 217 nodes mapped in the surrounding vegetation. Edge lengths are scaled to measured distance, but actual location is not represented here. N represents a nest, F represents a food source. Circles represent nodes. Solid lines represent edges used on that day; dashed lines represent edges not used that day (Methods). The color of a node represents the transition index (TI) from the preceding edge to the following one: TI–1, open circles; TI–2, blue; TI–3, red; TI–4, black. At a node where there is a choice of more than one edge in the indicated direction, so that there could be more than one transition taken through a given node, a TI was assigned to each possible transition. For such nodes with more than one transition index, the TI is represented graphically with a pie chart, and arrows show which transition has the TI represented by the arrow's color.

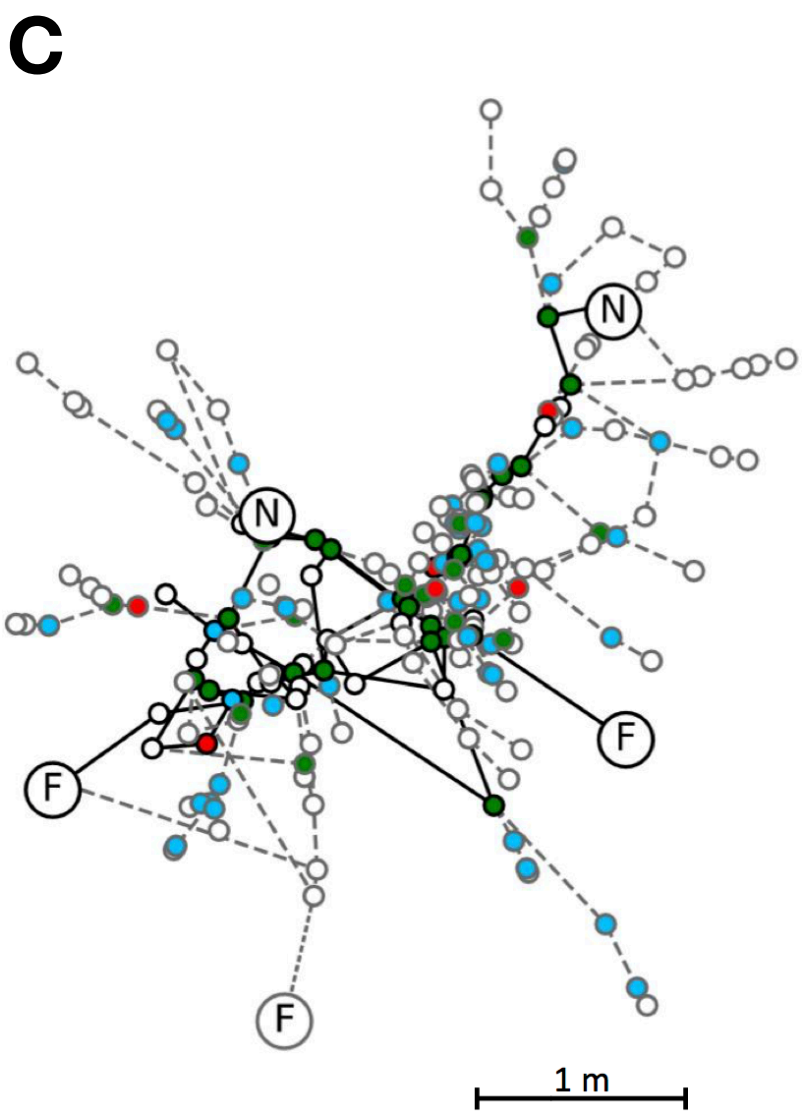


Figure 2.2: **Turtle ant vegetation.** C) Map of all 217 nodes for Tejon 189 on day 9. Symbols as in B. The large size of the network makes it difficult to show all TIs per node, so nodes with more than one TI are colored green.

its corresponding line graph (Methods). The nodes corresponding to the nests and food sources were designated as terminals. We compared the extent to which each observed network on a given day optimized each of the 3 objectives, relative to a set of 100,000 random networks connecting the same nests and food sources (Methods).

2.2.3 Turtle ant trail networks favor coherent trails over shortest paths

We compared the relation of observed and random networks, testing for differences among objectives, colonies, and for a statistical interaction of objective by colony. We tested the hypothesis that the

Table 2.1: **Comparison of observed and available networks within the vegetation for the three colonies observed.** Values for 'Available' are for all of the vegetation mapped. Values for 'Used' are means, averaged over observation days, for the paths used in each colony's network ($n = 10$ for Tejon 189 and Turtle Hill 460, and $n = 11$ for Tejon 446). Connectivity is a measure of the smallest number of nodes needed to get back to the trail used by the ants from an edge off the trail, averaged over all edges that lead off the trail used by the ants (Methods).

	Tejon 189		Tejon 446		Turtle Hill 460	
	Available	**Used**	**Available**	**Used**	**Available**	**Used**
Edge Length (cm)	10698	20.71 ± 6.21	7615	13.46 ± 0.74	14696	39.97 ± 9.43
Total Nodes	217	43.5 ± 9.43	196	36.09 ± 3.81	202	54.0 ± 17.93
Transition Index	1.60	1.31 ± 0.06	1.70	1.44 ± 0.13	1.48	1.44 ± 0.10
Connectivity	—	3.75 ± 0.23	—	4.00 ± 0.24	—	4.46 ± 0.95

networks favor coherent trails, and thus minimize the total number of nodes and the average transition index more than they minimize distance traveled.

In all three colonies, the ants' networks optimized the maintenance of coherent trails. There were significant differences among objectives in how well observerd networks were optimized by observed relative to random networks (Scheirer-Ray-Hare two-factor ANOVA df 2, $H = 31.718$, $p < 0.001$). Trail networks minimized the average transition index (Dunn test , $Z = 4.395$, $p < 0.001$) and minimized the total number of nodes (Dunn test, $Z = 5.247$, $p < 0.001$), significantly more than the distance traveled (Figure 2.3A). There were no significant differences between the extent to which observed networks, compared to random ones, minimized the average transition index and the total number of nodes (Dunn test, $Z = 0.852$, ns).

Colonies did not differ overall in the relation of observed and random networks for the 3 objectives (S-Test, df 2, $H = 2.010$, ns), but there was a significant objective x colony interaction (S-test, df 4, $H = 13.284$, $p < 0.01$). One colony, Turtle Hill 460, differed from the other two colonies in two ways, apparently because of differences in the local vegetation: first, it minimized total nodes, relative to random networks, significantly more than it minimized average transition index (Dunn test, $Z = 3.347$, $p < 0.05$), and second, it did not minimize average transition index significantly more than average length, relative to random networks (Dunn test, $Z = 1.012$, ns).

Random networks with the same number of nodes as the observed network did not differ in total length from the observed network. We tested this to control for the confounding of total edge length and total number of nodes because they are correlated ($R = 0.67$), and to test which one is prioritized when the two objectives give different results. We compared total length in observed and random networks by using

a percentile measure (Methods). At 50%, the observed network is equal for the objective to the average random network; the lower the percentile, the better the observed network optimized the objective compared to random networks. In all three colonies, the total length of observed networks was similar to that of the random networks with the same number of nodes: the percentiles were $35.49 \pm 39.57\%$ for Tejon 189, $40.59 \pm 27.65\%$ for Tejon 446, and $34.89 \pm 38.84\%$ for Turtle Hill 460). For all three colonies, the percentile was within one standard deviation of 50%.

2.2.4 Turtle ant trail networks increase coherence over time

From day to day, progressive changes in the trail networks of all three colonies tended to minimize the average transition index and the total number of nodes more than they minimized average length (Figure 2.3B–E). The extent to which average length was minimized varied greatly from day to day in all three colonies, suggesting that minimizing this objective was not a strong priority. Figure 2.3B shows an example of a day-to-day change that minimized both the number of nodes and average transition index. From day 9 to day 10, the network changed from the path shown in yellow to the path shown in blue, thus eliminating the 6 nodes circled, including 2 nodes with TI–2 and one node of TI–4, in favor of a path with nodes all of TI–1 (Figure 2.3B). Overall, trails in Tejon 189 (Figure 2.3C) consistently minimized the total number of nodes and average transition index but twice increased the average length (days 7 to 10 and 11 to 15). Similarly, in Tejon 446 (Figure 2.3D), trails progressively decreased the total number of nodes and average transition index, but increased average length from days 6 to 7, 8 to 9, and 11 to 13. In Turtle Hill 460 (Figure 2.3E), the network consistently minimized the total number of nodes. This trail network shifted nests and food sources, and the change led to lower transition indices (days 2 to 10). There was an initial decrease in average edge length (days 4 to 7), due to a rupture on day 6 of a node leading to a 95 cm edge, one of the longest edges we measured, rather than to a choice of shorter edges, and then the trails increased in average edge length (days 7 to 14). These day-to-day changes indicate that the networks do not consistently minimize the distance traveled.

2.2.5 Turtle ants form loops to promote coherence

Turtle ants form loops in their paths, consisting of small, temporary alternative paths with the same start and end points, and over time, all but one of these paths tends to be pruned away [3]. Loops are often considered to decrease the efficiency of routing networks [17, 120], but may increase robustness by offering alternative paths if links are broken. Here we hypothesize that loops may occur because trails tend to form along easily reinforced nodes, and some sequence of easily reinforced nodes may naturally form a cycle in

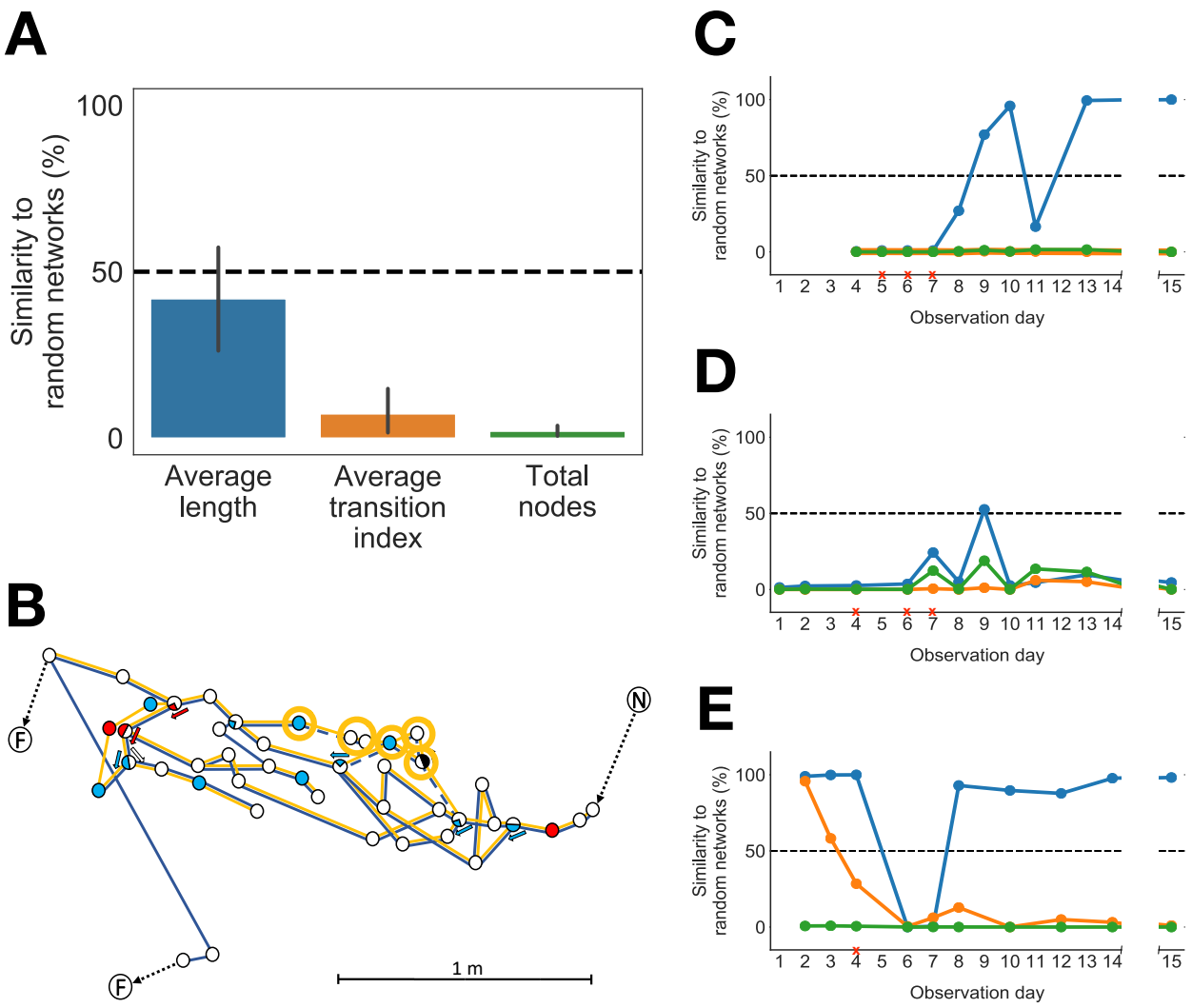


Figure 2.3: **Comparison of random and observed networks.** A) Similarity of observed and random networks measured as mean percentile (Methods). A percentile of 50 (dashed line) indicates that the observed network optimized the objective to the same extent as the average random network; the lower the percentile, the better the observed network optimized the objective compared to random networks. Error bars show standard error of the mean. B) Day to day change in trails in Tejon 189, showing a change that decreased both number of nodes and average transition index. Symbols for transition index are the same as in Figure 2.2B :TI–1, open circles; TI–2, blue; TI–3, red; TI–4, black. Solid yellow lines show trails used on day 9; solid blue lines show trails used on day 10; yellow circles and dashed blue lines show trails linking the six nodes that were used on day 9 but not on day 10. C–E) Day to day changes in mean percentile for each objective. Blue represents average edge length, orange represents average transition index, and green represents total number of nodes. A red 'X' indicates one or more ruptured edges on that day. C, Tejon 189, D, Tejon 446, E, Turtle Hill 460.

the graph.

Compared to available loops with the same start and end points (Methods), the observed loops tended to use nodes with low transition indices, and tended to minimize the number of nodes. We compared the centered ranks (Methods) of the average transition index, number of nodes, and average edge length,

of paths connecting the same two start and end points on each trail in observed and available loops. A negative value of centered rank indicates that an observed path has a low transition index compared to all of the random paths within the available vegetation with the same start and end points. The mean (SD) of the centered rank in observed loops for average transition index was -1.59 (2.96); for number of nodes was -1.88 (2.91) and for average edge length was 0.44 (2.23); in all cases, significantly different from 0 (T-Test, $|T| > 3$, $p < 0.01$). These results suggest that loop formation in trails may occur as a consequence of selecting trails that have lower transition indices, thus promoting coherence.

2.3. Discussion

The trail networks of arboreal ants show how evolution shapes biological distributed algorithms to respond to dynamic environments. Minimizing distance traveled is often considered the main objective in wireless routing algorithms and ant colony optimization [131, 83, 37], since rapid communication is often needed between any two nodes in the network. We showed that the trail networks of turtle ants instead optimize the maintenance of a coherent trail by minimizing the total number of nodes, and by minimizing the average transition index; optimizing the latter objective, we show, is NP-complete (Methods). In previous work [143], we proposed a model for the algorithm used to maintain and repair networks. This algorithm did not distinguish between minimizing the number of nodes and the distance traveled, since each edge had equal length. Our results here show that further work is needed to develop an algorithm that captures the role of physical variation in the environment.

Minimizing the transition index and the number of nodes contributes to maintaining the coherence of the trail because both diminish the risk of losing ants from the trail. Like water flowing over a rocky stream bed, turtle ants tend to find trails most conducive to the flow of ants. By avoiding nodes with high transition indices, turtle ants reduce the chances of ants wandering off the path and laying pheromone trail that can lead other ants also to leave the trail. Nodes with transition indices of 1 keep the trail on the same plant [3]. The vines and trees of the tropical dry forest tend to have long internode distances to reach the sunlight at the edge of the canopy [146]. By staying on the same plant, ants are also led to resources at the edge of the canopy, such as flowers that provide nectar.

The trail networks must also balance the tradeoff between exploration and coherence. Exploration is necessary for the colony to construct trails [110, 71, 138, 19, 137, 135], search for new resources [147, 148, 149, 150, 151, 136] and repair breaks [3, 143, 133], and the connectivity of the vegetation (Table 2.1) sets the probability that ants that leave the trail will return to another node on the trail. There appears to be a constant probability of exploration at each node [3, 143], so the probability of leaving the trail accumulates

with more nodes traversed. Thus, minimizing the number of nodes traversed reduces opportunities for the ants to get lost.

Our results here suggest that, as in engineered networks [144, 145], the cost of including a node in the turtle ant network may vary among nodes, because whether other ants follow an exploring ant that leaves the trail at that node depends on the node's physical configuration. The costs of additional nodes include the loss of ants from the trail, and the pursuit of fruitless paths, which may detract from the colony's ability to distribute resources among its many nests, and make fewer ants available to recruit effectively when a new food source is discovered. In addition, each node provides opportunities for encounters with other, competing species traveling in the same vegetation [152, 153, 154]. Further work is needed to examine variation among colonies [155] in how well they minimize the number of nodes, to learn how selection may shape the process that determines how a trail network is constructed and maintained.

Finally, are there useful applications of the principle that variation in the environment provides useful constraints on routing network design? Evolved algorithms operating in the natural world can use structure in the environment to enhance coordination among distributed agents [156]. In engineering, taking into account the physical structure of the environment may improve the design of routing algorithms [157, 158, 159, 160, 161], for example, by reducing the search space of possible routing paths and steering network construction away from parts of the terrain that are difficult to reach. This could be beneficial in applications such as robot swarms, where distributed agents must coordinate in complex environments using a communication backbone to explore new terrain, exploit of temporary resources, and maintain security against intruders [162].

2.4. Materials and Methods

2.4.1 Observation of trail networks

The trail networks of three turtle ant colonies (Turtle Hill 460, Tejon 446, and Tejon 189) were observed between 06/20/18 and 08/03/18 at La Estación Biológica de Chamela in Jalisco, Mexico. Tejon 189 was observed for 10 days from 6/22/208–6/28/2018 and again on 07/01/2018, 7/04/2018 and 8/3/2018. Tejon 446 was observed for 11 days on 6/20/2018–6/28/2018 and again on 07/01/2018, 07/04/2018, and 08/03/2018; and Turtle Hill 460 was observed for 10 days from 06/21/2018–06/28/2018 and again on 07/02/2018, 07/05/2018, and 08/03/2018. Not every network was observed on each day. The colonies appeared to be of about the same size, based on observations in comparison with previous work in which size was estimated with mark-recapture measures [56], but here we did not measure colony size.

The numbers of observation days were 10 for Tejon 189 and Turtle Hill 460, and 11 for Tejon 446, for a total of $93 = (10 + 10 + 11) \times 3$ observed networks.

Networks were mapped using the same methods as in prior work [3], using visual inspection of the paths the ants took through the vegetation. We mapped each node and edge traversed by ants and all possible paths up to 5 nodes away from each node traversed by ants. Each node was assigned a number, and enough nodes were marked, with small labels or stickers on a dead end branch near the node, to identify all same nodes the next day. The initial map of a colony's trail network included all nodes used by the ants, and all nodes up to 5 nodes from each node on the path. The length of each edge was measured with a ruler, including each edge connecting the nodes on the ants path and all nodes up to 5 nodes from each node on the ants' path. A transition index was assigned corresponding to each pair of edges entering and leaving a node. The assignment of a transition index was based on a visual estimate of the number of different trajectories that were available for ants to traverse the node. We were not able to make any direct measurement of the amount of pheromone deposited.

The assignment of the transition index was made using the following criteria:

- TI–1 (Figure 2.1A): a node linking two edges on the same plant.
- TI–2 (Figure 2.1B): a node that links one plant to another along a trajectory through a node that is likely to be the same for successive ants.
- TI–3 (Figure 2.1C): a node that links one plant to another plant with more than one possible trajectory through the node.
- TI–4 (Figure 2.1D): a node that links one plant to another with many possible trajectories that is often changed by conditions such as the wind.

In each day of observation, we recorded which edges and nodes were used by the ants. As observed previously (illustrated for a different set of 3 colonies [3]), each colony's trail network changed which nodes it used from day to day, although each day's path conserved some parts of the previous day's. In the course of the observations reported here, the path of the ants never went outside the range of the 5 nodes around the trail that were originally mapped. The number of nodes and edges observed in each network is shown in Table 2.1.

2.4.2 Assigning directionality to edges

Experiments with marked ants show that ants tend to use particular routes from a nest [56], and tend not to turn around on the trail. To account for this, for all edges we defined a direction relative to one

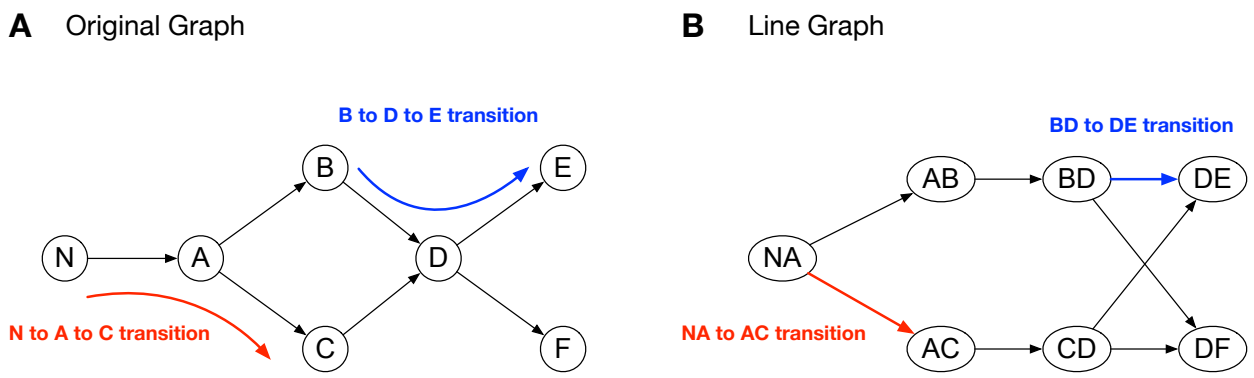


Figure 2.4: **Converting the network to the line graph.** A) Original graph. Nodes correspond to junctions in the vegetation, and edges correspond to links, such as a branch or stem, between junctions. B) Line graph. Every edge in the original graph has a corresponding node in the line graph. Two nodes are connected in the line graph if they correspond to two adjacent edges in the original graph. Transitions in the original graph used in this example are highlighted in red and blue.

terminal; the outbound direction went away from it, and the inbound direction went towards it. We restricted the analysis to paths that proceed in the outbound direction.

2.4.3 Modeling the transition index as a line graph

The transition index of a node describes the probability that successive ants will take the same trajectory from an edge (u, v) through node v to edge (v, w). Thus, a transition index involving node v depends on which incoming edge was used to reach v, and which outgoing edge was used to leave v.

To model the transition index, we used the line graph of G, where $L_G = (V_L, E_L)$. The nodes $V_L = E_G$, and the edges $E_L = \{((x, y), (y, z))|(x, y), (y, z) \in E_G\}$. The line graph creates a node for each edge in G and connects two nodes if they correspond to two adjacent edges in G. Every edge in the line graph denotes a transition in the original graph; traversing the edge $((u, v), (v, w)) \in L_G$ corresponds to starting at u, going to v, crossing the junction at v, and then going to w. We assigned every edge in L a transition index. Figure 2.4 illustrates this process. The line graph is used to compare the observed and random networks, as described below.

2.4.4 Generating random trail networks

We compared the observed trail networks to random trail networks based on their average edge length, total number of nodes, and average transition index. The random networks were simulated in the graph made from the map of the paths used by ants and the surrounding vegetation up to five nodes from the nodes used by the ants. The measures of number of nodes, edge length and transition index were those

measured in the vegetation. The random trail networks may include loops, as did the observed networks.

We generated random trail networks as follows:

1. **Input:** Graph $G = (V_G, E_G)$, terminals $X \subseteq V_G$.
2. Create an empty graph R; add a random terminal $x \in X$ to R.
3. Choose a random terminal $x' \in X$ that has not already been added to R.
4. Perform a random walk on G that starts at x' and stops when it touches any node in R.
5. Add all edges and nodes touched by the random walk to R.
6. Repeat steps 3–5 until all terminals have been added to R.

2.4.5 Comparing observed and random networks

To compare the observed and random networks, we computed for each network, using the line graph (Figure 2.4), the following three objectives: 1) Average edge length: the average length of all edges in the network; 2) Total number of nodes: the total number of nodes in the network; and 3) Average transition index: the average transition index over all transitions in the network.

For each objective, we computed a value of φ_{ants} for the observed network and $\varphi_1, \varphi_2, \ldots, \varphi_n$ for the n random networks. We measured the similarity between the observed network and the random networks for that objective as the percentage of random networks that have a lower value for the objective:

$$100 \times \frac{|\{\varphi_i | \varphi_i \leq \varphi_{\text{ants}}\}|}{n}. \tag{2.1}$$

An observed network is most similar to a random network when its percentile is 50. The closer the percentile is to 0, the better the observed network optimized the objective, and the closer the percentile is to 100, the better the random network optimized the objective. This percentile-based approach is unit-less, making it possible to compare performance for objectives that differ in the range of values.

We treated each comparison of the observed and simulated paths, drawn from observations on a given day, as independent, because we compared each day's path to a set of randomly generated paths using the same number of nodes. The relation between the randomly generated paths and the observed paths for each day was independent of those on any other day. We did not compare the observed paths from one day to those of another day. The path used by the ants on one day was similar to the one used by the ants on the previous day, perhaps because of the effect of the transition indices which generally remained the same from day to day.

For each colony (Tejon 189, Tejon 446, Turtle Hill 460), and for each objective (average length, total nodes, and average transition index), we computed the similarity of each observed network to 100,000 random networks that connected the same set of terminals, using Equation (2.1).

Because the total number of nodes and the total edge length are correlated, we did not simultaneously compare total length and total number of nodes between random and observed trail networks. Instead, we tested whether there was a statistically significant difference in total length between observed and random trail networks with the same number of nodes. For each observation of one colony on one day, we found all the randomly generated networks that had the same number of nodes as that observed network (209 for Tejon 189, 1420 for Tejon 446, 221 for Turtle Hill). We then found the total lengths of all of these networks, and used Equation (2.1) to evaluate how well each observed network optimized total length compared to random networks with the same number of nodes.

2.4.6 Comparision of objectives and colonies

We used the non-parametric Scheirer-Ray-Hare two-factor ANOVA test [163] to test for effects of objective, colony, and the colony x objective interaction on the percentiles of the observed networks compared to random networks. We then applied post hoc Dunn's non-parametric tests [164], using a Bonferroni correction for multiple comparisons, with a significance threshold of $p = 0.05$.

2.4.7 Loops

We compared the average transition index and number of nodes in observed loops with loops that were available in the underlying vegetation. A loop was defined as two or more outbound paths that start and end at the same source and target nodes. For each observed loop, we computed all possible paths in the vegetation between the corresponding source and target nodes. We ranked all paths based on the average transition index of the path, and compared this to the rank of the average transition index of the paths used by the ants. We repeated the same measure for total number of nodes and average edge length.

We computed the *centered rank* of each path used by the ants as follows: We ranked n available paths in the vegetation, by the average transition index of nodes in the path, total number of nodes in the path, or average length of edges in the path. The ranks of the paths are $1, 2, 3, \ldots n$, and the median rank is $r_m = (n+1)/2$. For each observed path, we computed its centered rank by subtracting the median rank from that path's rank. Using average transition index as an example, the centered rank is 0 when the observed loop had the same average transition index as the median random loop, negative when the observed loop has a lower average transition index, and positive when the observed loop has a higher average transition

index. We performed a similar analysis for the total number of nodes and average edge length.

2.4.8 Connectivity

To calculate the connectivity of the vegetation (Table 2.1), we estimated how many nodes are required for an ant that leaves the trail to return to the trail. For each edge (u, v) connecting a node on the trail to a node off the trail, we found the length, in number of nodes, of the path with fewest nodes from v back to a node on the trail. We measured connectivity as the smallest number of nodes in a path back to the trail, averaged over all edges (u, v) leading off the trail on all days.

2.4.9 Optimizing transition index is NP-complete

Here we show that the problem of constructing the foraging network that connects a given set of terminals while minimizing the average transition index of the network is NP-complete. To do this, we start by showing that finding the minimum average-weight path between two vertices in a graph is NP-complete. In this problem, we are given a graph $G = (V_G, E_G)$ and two vertices $u, v \in V$. The goal is to find a path $\mathcal{P} = [(u, r_1), (r_1, r_2), \ldots, (r_k, r_{k+1}), (r_{k+1}, v)]$ that minimizes the average edge length: $\frac{1}{|\mathcal{P}|} \sum_{e \in \mathcal{P}} w(e)$, where $w(e)$ defines the length of edge e. This problem differs from the classic shortest path problem, which seeks a path with minimal total edge length.

The standard method for considering the complexity class of an optimization problem is to consider the equivalent decision version of the problem: given a graph $G = (V_G, E_G)$, two vertices $u, v \in V_G$, and an integer k, we ask: is there a path from u to v whose average weight is $\leq k$?

Lemma 1. *Finding the minimum average-weight path is NP-Complete.*

Proof. First, we show that this problem is in the class NP. If we are given a path from u to v, we can verify that the path is a valid u-v path, and that the average edge weight is $\leq k$. This certificate will clearly be of polynomial length, and we can verify that it is correct in polynomial time.

Next, we show that the problem is NP-hard. We proceed via reduction from the Hamiltonian path problem, which is NP-Complete. Given a directed graph $G = (V_G, E_G)$, the directed Hamiltonian path problem seeks a path that touches every vertex $v \in V_G$ exactly once. We construct a graph G' as follows: G' contains all of the vertices in G along with two additional vertices, s and t. We assign a weight of 1 to all of the original edges in G. We add a directed edge from s to every vertex $v \in V_G$, and we add a directed edge from every vertex $v \in V_G$ to t. We assign a weight of 2 to all edges that include either s or t.

The smallest possible average edge weight for any path from s to t is $\frac{3+|V_G|}{|V_G|+1}$, and such a path exists if and only if G has a directed Hamiltonian path.

The reduction requires adding 2 new nodes, and $2|V|$ new edges to G. Thus the reduction clearly takes only polynomial time. □

We now use Lemma 1 to show that optimizing average transition index is NP-Complete. Let $L = (V_L, E_L)$ be a graph whose edge weights correspond to transition indices, which we represent using the line graph (Figure 2.4). Given a set of terminals $X \subseteq V_L$, we seek to find a subtree $F_X \subseteq L$ that minimizes the average value of all edge weights in F_X. We require that F_X be connected without cycles to represent the fact that cycles in turtle ant trails are typically pruned.

Once again we consider the decision problem: given a line network $L = (V_L, E_L)$, a set of terminals X, and an an integer k, does L contain a subtree F_X whose average weight is $\leq k$?

Lemma 2. *Finding the subtree F_X that optimizes average transition index is NP-Complete.*

Proof. First, we prove that the problem is in NP. Given V_L, X, k, if we are presented with a subgraph F_X as a certificate, we can verify that F_X is a valid solution. We check that F_X is valid subgraph, that F_X is connected, that F_X contains every terminal node X, and that the average weight of the edges in F is $\leq k$. Clearly the size of F_X is polynomially bounded, and we can verify that F_X is a valid solution in polynomial time.

To show that the problem is NP-hard, we proceed via reduction from the minimum average-weight path problem above. Given G and vertices u and v, we simply treat G as the line graph (L), and designate terminals $X = \{u, v\}$; this means F_X is simply a u-v path. We seek the trail that optimizes the average transition index between terminals u and v. By construction, this is the minimum average-weight path from u to v in the line graph, meaning there is a subgraph F_X with average weight of $\leq k$ if and only if the original graph has a u-v path with average edge weight $\leq k$. Further, the reduction clearly takes only polynomial time. □

2.5. Data Availability Statement

All data collected and analyzed will be archived at Stanford Digital Repository at the Stanford Libraries. Python code is available at: `https://github.com/DiODeProject/SteinerAnts`.

2.6. Acknowledgements

Chapter 2, in full, is currently in review. Chandrasekhar, Arjun; Marshall, James; Austin, Cortnea; Navlakha, Saket; Gordon, Deborah M, bioRxiv doi: 10.1101/714410. The dissertation author was the primary investigator and author of this paper.

Chapter 3

Neural Arbors are Pareto-Optimal

The problem of designing a routing network to efficiently transport resources is a fundamental problem in both engineering [17], and in natural systrems such as slime molds [165] and bacterial colonies [166]. In the two previous chapters, I described aspects of the distributed routing algorithm that turtle ants have evolved to navigate between nests and food sources, including design principles that influence the colony's choice of nodes and edges. In this chapter, I extend my study of biological routing networks by exploring how neural arbors manage trade-offs between competing design principles. I demonstrate the value of using neural arbors as an inspiration for network design algorithms, and I highlight the value of an algorithmic model of neural arbors for producing novel hypotheses and explanations for variation in neural arbor morphologies.

Abstract

Neural arbors (dendrites and axons) can be viewed as graphs connecting the cell body of a neuron to various pre- and post-synaptic partners. Several constraints have been proposed on the topology of these graphs, such as minimizing the amount of wire needed to construct the arbor (wiring cost), and minimizing the graph distances between the cell body and synaptic partners (conduction delay). These two objectives compete with each other — optimizing one results in poorer performance on the other. Here, we describe how well neural arbors resolve this network design trade-off using the theory of Pareto optimality. We develop an algorithm to generate arbors that near-optimally balance between these two objectives, and demonstrate that this algorithm improves over previous algorithms. We then use this algorithm to study how close neural arbors are to being Pareto optimal. Analyzing 14145 arbors across numerous brain regions, species, and cell types, we find that neural arbors are much closer to being Pareto optimal than would be expected by chance and other reasonable baselines. We also investigate how the location of the arbor on the Pareto front, and the distance from the arbor to the Pareto front, can be used to classify between some arbor types (e.g., axons versus dendrites, or different cell types), highlighting a new potential connection between arbor structure and function. Finally, using this framework, we find that another biological branching structure — plant shoot architectures used to collect and distribute nutrients — are also Pareto optimal, suggesting shared principles of network design between two systems separated by millions of years of evolution.

3.1. Introduction

Both man-made and biological transport networks face trade-offs in their design. While resources can be spent to increase the speed or reliability of transport, this often comes at a cost. For example, in road networks, users wish to minimize the travel time to get from one point in a city to another, but this goal conflicts with the practical need to minimize costs in building infrastructure. In such cases, engineers typically seek to design network topologies that achieve the best “bang for the buck”. Here, we analyze how well neural arbors resolve a similar performance-versus-cost trade-off.

Neural arbors (dendrites and axons) can be viewed as transport networks rooted at the cell body that process and relay information from one neuron to another [167]. Prior work has proposed competing constraints on the topology of these networks [168]. For example, for both dendrites and axons, the principle of wiring economy states that the total length of an arbor should be minimized [169, 170, 171, 172, 173, 174]; total wire is a measure of network cost and is especially important when space is limited or when wire is a commodity [175]. On the other hand, Cajal’s principle of conduction delay states that, for axons, the distance

or time required to propagate an action potential from the soma (cell body/root) to downstream post-synaptic partners should be minimized for efficient signal propagation [176]. For dendrites, minimizing conduction delay similarly allows for efficient propagation of dendritic action potentials [177] or back-propagating action potentials [178], and it minimizes the effects of attenuation (weakening of the signal as it travels from a synapse to the soma) [179].

The two topologies that minimize wiring economy and conduction delay, respectively, are often different (Figure 3.1A). A Steiner tree [140] minimizes wiring economy, but this structure could have poor conduction delay for some nodes. On the other hand, the Satellite tree, an arbor that connects the cell body directly to each synaptic partner via a straight line would optimally minimize conduction delay, but this structure would be very inefficient for wiring economy.

Here, we use the theory of Pareto optimality to analyze how well neural arbors resolve this network design trade-off. Intuitively, a network is Pareto optimal if there does not exist an alternative topology that improves on one objective without hurting the other. Given enough evolutionary pressure and genetic diversity, we hypothesize that mechanisms generating such sub-optimal topologies would be eliminated from the population [59]. The only topologies that would remain in the population are those that lie on the *Pareto front*, consisting of all topologies that are Pareto optimal.

We formalize this problem, develop a graph-theoretic algorithm to generate close to Pareto optimal topologies, and we evaluate how well neural arbors come to realizing Pareto optimality. In particular, we analyzed the structure of 14145 arbors from a variety of species, cell types, and brain regions and find that most arbors lie on or very close to the Pareto front, and that this is highly unlikely to occur by chance (as measured by three baseline graph models that do not seek to directly optimize either wiring cost or conduction delay). One advantage of this approach is that each individual arbor can be associated with a single parameter ($\alpha \in [0, 1]$), indicating how the arbor weighs or prioritizes each objective. We find that the α values can be used to classify some functional categories of neurons based on their structure (e.g., axons versus dendrites, or different cell types or neurotransmitter types). Finally, to show the generality of this result to other biological branching structures, we re-analyze tracings of hundreds of plant shoot architectures and find plants are also Pareto optimal. To our knowledge, this is the first quantified comparison between plant architectures and neural arbors based on their ability to trade-off two common network design criteria.

3.2. Related work

Since Ramon y Cajal's seminal work [176], the principle of wiring economy has served as an important constraint on the structure of neural circuits. In addition to constraining arbor morphology, wiring

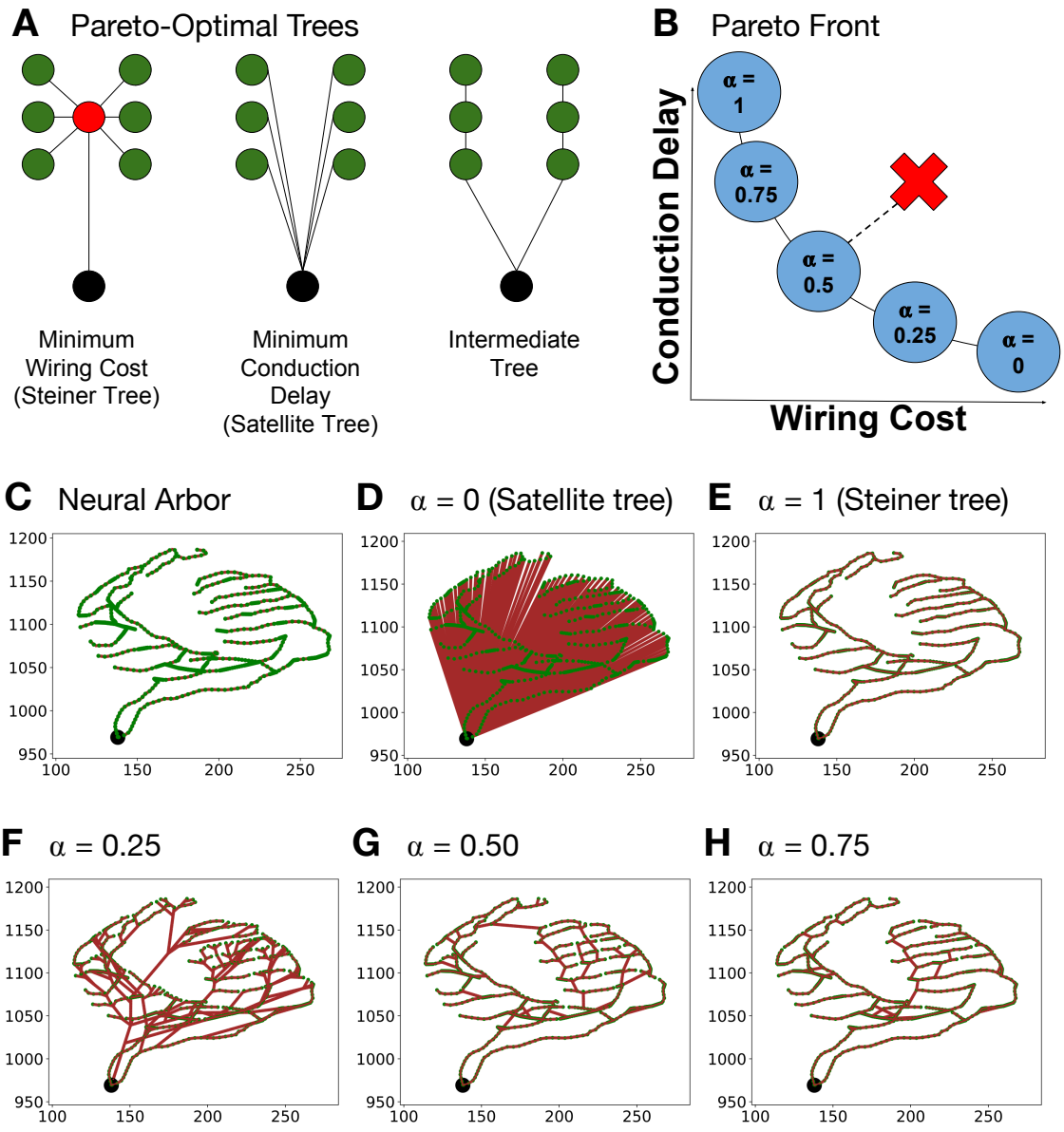


Figure 3.1: **Illustration of Pareto optimal trees and the Pareto front.** A) For a given set of points (green), there are several possible trees that can be formed, each optimizing wiring cost and conduction delay differently. The black node indicates the root (cell soma) of the arbor, and the red points indicate branch points. The Steiner tree minimizes wiring cost. The Satellite tree minimizes conduction delay. Intermediate trees lie in between. B) The Pareto front defines a set of trees for which improving upon one objective requires a loss in the other objective. The value of $\alpha \in [0, 1]$ indicates a prioritization weight that the arbor places on one objective versus the other. We evaluate how Pareto optimal a given tree (red 'X') is by computing the distance from that tree to the Pareto front. C–H) Example Pareto optimal trees generated by our greedy algorithm using different values of α. The black dot is the soma, the green dots are synapses, brown dots are Steiner points, and red lines are edges. The axes are coordinates, in microns.

cost has been used to predict the placement of the cell body within its arbor [169] and the placement of multiple cell bodies within a network [170, 171, 180]. Wiring economy has also been used to help ex-

plain why certain neurons tend to restrict growth to a 2-dimensional plane, even when allowed to grow in 3-dimensions [181]. There are also other measures that constrain arbor topology, including neuron density [182], space-filling [183, 184, 185], and network connectivity measures, such as effective radius, clustering [186], and minimizing refractory delay [187]. Here, we focus primarily on wiring economy and conduction delay to quantify how well arbors balance between a general optimization trade-off without using a model that requires a large number of parameters to capture additional biological constraints; we later discuss deviations in the context of other potential constraints.

There are also several features that affect conduction delay, including wire radius and myelination [188, 189, 190]. In particular, prior models predict a $3/2$-power law relationship between the radii of mother versus daughter wires, a model which has been validated theoretically [191] and experimentally [192]. Additionally, allometric scaling laws have made predictions about what shapes general biological transport networks take on to optimize transport efficiency as a function of volume [193, 194]; similarly, Banavar et al. [195] study metabolic scaling in source-directed transport networks and suggest that efficient topologies minimize the mean distance from the source to the sinks; i.e., the tree is Satellite-like. These studies, however, do not study general trade-offs between efficiency of transport and network construction costs, as we do here. Radius measurements of individual arbor segments are unfortunately not broadly available across arbors from multiple cell types, species, and brain regions, and thus we consider wire as 1-dimensional here.

Multi-objective optimization for understanding how wiring length and conduction delay are balanced has also been considered by prior work [173, 174]. These studies model arbors as spanning trees, which do not allow the addition of branch points, whereas our model uses Steiner trees, which generalize spanning trees by also allowing branch points. Branch points allow us to more realistically and accurately characterize possible arbor topologies. There are also other differences in our work. First, we compare to three baselines topologies to gauge the significance of achieving Pareto optimality. Second, we analyze many more arbors (14145 arbors across numerous species, brain regions, and cell types, compared to only 10 spiny cell axons and 9 basket cell axons from the cat visual cortex, as previously done [174]). Third, we analyze a subset of arbors with exact synapse locations known, rather than simply the arbor tracings. Fourth, we analyze both dendrites and axons, whereas prior work studied only axons [174] or dendrites [173].

In the theoretical computer science community, these two objectives have also been studied under different guises. For example, wiring length minimization is an instance of the Euclidean Steiner tree problem [196, 197, 198]. This problem takes a set of points in Euclidean space as input, and seeks to build a tree connecting all the points in such a way as to minimize the total length of the tree. In contrast, we include a second competing function (conduction delay) to jointly optimize. More related is the problem of finding

a light approximate shortest-path tree (LAST) [199, 10, 200]. Here, the goal is to find a spanning tree on the set of input points whose total weight is not "too high" while also placing each point not "too far" from a pre-designated root. Our work solves a similar problem parameterized differently, and again with the option of adding branch points along the arbor. Empirically, we also show that our algorithm produces trees that are closer to Pareto optimal than LAST trees.

Finally, our work also contributes to the class of multi-objective minimum spanning tree problems. Finding Pareto optimal solutions to these problems is generally NP-hard, and several evolutionary algorithms have been proposed to generate approximate solutions [201, 202]. These problems typically consider objective functions that can be locally decomposed, whereas one of our objective functions (conduction delay) is a global property of the tree.

3.3. Theory

3.3.1 A framework for analyzing neural arbors as transport graphs

As input, we are provided a set of points $\mathcal{P} = \{p_1, p_2, \ldots, p_n; r\} \in \mathbb{R}^3$. The points p_i represent the 3D locations of the n synapses that the arbor connects to (pre-synaptic for dendrites, and post-synaptic for the axon). The point r represents the location of the cell body. Our goal is to output a tree $T_{\mathcal{P}} = (V_{\mathcal{P}}, E_{\mathcal{P}})$ rooted at r that connects the points p_i, where $V_{\mathcal{P}}$ contains all points in $\mathcal{P}$ and can include additional branch points not present in the input for more efficient connectivity. The edges $E_{\mathcal{P}} \subseteq V_{\mathcal{P}} \times V_{\mathcal{P}}$ are such that $T_{\mathcal{P}}$ is a tree, i.e., it is connected with no cycles. Neural arbors typically do not pre-define the precise locations of synaptic contacts ahead of time, but rather connect to potential partners as they grow [203, 204]. Our model is not meant to characterize the process by which neural arbors grow and branch. Rather, given how a neural arbor branched, our model evaluates how optimal the arbor morphology was compared to all the possible morphologies that could have been used to connect the cell body to the synapses p_i.

We evaluate the wiring cost and conduction delay of the arbor as follows. For edge $(u, v) \in E_{\mathcal{P}}$, define the edge length, $l(u, v)$ to to be the Euclidean distance between u and v. For two points $u, v \in V_{\mathcal{P}}$, define the graph distance $d(u, v)$ to be the length of the shortest path from u to v using the edges $E_{\mathcal{P}}$.

The *wiring cost* is the total length of all edges in the tree:

$$W(T_{\mathcal{P}}) = \sum_{(u,v)\in E_{\mathcal{P}}} l(u, v). \tag{3.1}$$

The *conduction delay* is the sum of the graph distances from the root to each synapse:

$$D(T_{\mathcal{P}}) = \sum_{p_i \in \mathcal{P} \setminus \{r\}} d(r, p_i). \tag{3.2}$$

The tree that minimizes wiring cost alone is the Steiner tree (Figure 3.1A), which, unlike a minimum spanning tree, allows for branch points that may help reduce the total length of the arbor. There are efficient algorithms to compute a minimum spanning tree, but finding the optimal Steiner tree for a given set of points in 3D space is NP-hard [205]. The tree that minimizes the conduction delay alone is the 'Satellite' tree, where each synapse is connected via a straight line to the root of the tree (Figure 3.1B). We compare trees built on the same set of input points, and thus we do not need to normalize either objective by the number of synapses, n.

3.3.2 Defining Pareto optimal trees

It is generally impossible to find a tree that simultaneously minimizes both wiring cost and conduction delay because such a tree may not exist. One way to define the concept of an 'optimal tree' then is using the theory of Pareto optimality. Intuitively, a tree is Pareto optimal if no other tree on the same set of points has a lower value for both objectives, or a lower value for one objective without hurting the other objective. Formally, we say a tree T_1 *partially dominates* a tree T_2 (denoted $T_1 \preceq T_2$) if at least one of the two conditions hold:

1. $W(T_1) \leq W(T_2)$ and $D(T_1) < D(T_2)$; or
2. $W(T_1) < W(T_2)$ and $D(T_1) \leq D(T_2)$

This means that T_1 performs equal to or better than T_2 no matter how one prioritizes the two objectives. We say a tree T is *Pareto optimal* if it is not partially dominated by any other tree on the same set of points [206].

To generate Pareto optimal trees, we propose a simple linear interpolation between the two objectives, where the goal is to minimize the *Pareto cost*:

$$\min_{T_{\mathcal{P}}} \alpha W(T_{\mathcal{P}}) + (1 - \alpha) D(T_{\mathcal{P}}), \tag{3.3}$$

where $T_{\mathcal{P}}$ is a tree that connects the cell body (r) to the synapses $(p_1, p_2, \ldots, p_n)$. Cunha and Polak [207] prove that finding a tree that minimizes Equation (3.3) is equivalent to finding a Pareto optimal tree. Here, α denotes how much weight is placed on one objective versus the other.

To generate the *Pareto front* (Figure 3.1B), we need to find the set of trees that minimize Equation (3.3) for each value of $\alpha \in [0, 1]$. If $\alpha = 0$ the optimal tree is the Satellite tree. If $\alpha = 1$ the optimal tree is the minimum Steiner tree. To generate optimal trees for intermediate values of α, we developed an algorithm, as described next.

One advantage of this formulation is that each neural arbor can be associated with global arbor descriptor (its value α) that characterizes the trade-off between the two objectives that the arbor employs. We can then compare the distribution of these α values across arbors of different types to quantify morphological differences.

3.3.3 An algorithm for generating Pareto optimal trees

To generate trees that lie on the Pareto front, we used a greedy algorithm that allows for branch points, which were not considered by prior work [173, 174]. The algorithm takes as input $\mathcal{P}$ and α and outputs $T_{\mathcal{P}}$ that attempts to minimize Equation (3.3).

The algorithm starts with the root r in the tree, and all other points (nodes) outside the tree. In each step, the algorithm considers all possible edges between a node inside the tree and a node (synapse) outside the tree, and picks the edge that minimally increases the objective in Equation (3.3). When the algorithm adds an edge, it adds k Steiner nodes equidistant along the edge, which may be used in subsequent steps as branch points. This process continues until all synapses have been added to the tree. In experiments, we set $k = 10$.

Figure 3.2 illustrates five steps of this process. Importantly, this algorithm is centralized and is designed to generate a near-optimal Pareto front that can be used to evaluate the optimality of neural arbors. This algorithm is not meant to mimic the distributed process by which neural arbors grow. We do not derive approximation bounds for this algorithm; however, in Results we show that this algorithm performs well in practice.

3.3.3.1 Reducing the time complexity of the greedy algorithm

For arbors with thousands or tens of thousands of synapses this algorithm may not be very efficient because of the large number of possible edges to consider in each greedy step. Specifically, there are $\mathcal{O}((kn)^2)$ edges to consider, where n is the number of unconnected points and k is the number of Steiner points that we add along each edge. Here, we show that we can reduce the number of candidate edges to consider in each step to $\mathcal{O}(kn)$ without affecting the quality of the solution generated by the algorithm.

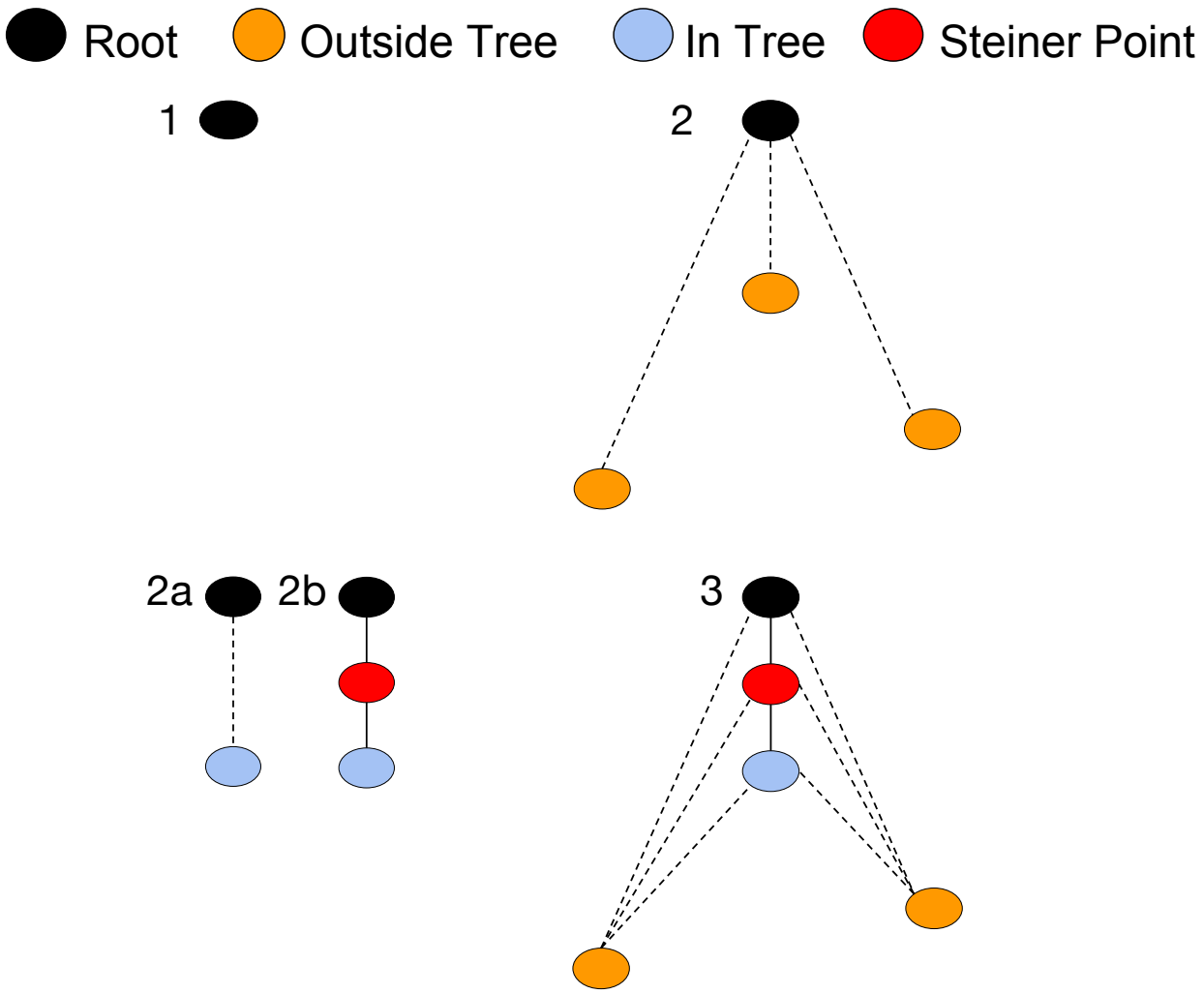


Figure 3.2: **Illustration of the greedy algorithm.** 1) The algorithm iteratively adds nodes to the tree, starting with the root. 2) In each step, it considers all edges between a node currently inside the tree and a node currently outside the tree. The edges considered are shown as dashed. 2a) It greedily adds the edge that increases the cost of the tree by the lowest amount. 2b) Along the chosen edge, the algorithm adds k Steiner points. In the example, $k = 1$, but in our implementation, we set $k = 10$. 3) The algorithm then repeats this process with the updated tree until all nodes outside the tree have been added to the tree.

For each node $u \in V_{\mathcal{P}}$ already added to the tree, define:

$$\text{Close}(u, T_{\mathcal{P}}) = \operatorname*{argmin}_{v} \{l(u, v) : v \in \mathcal{P} \setminus V_{\mathcal{P}}\},$$

i.e. the closest point to u that has not already been added to the tree. Then, define:

$$\text{Cost}(u, v, T_{\mathcal{P}}, \alpha) = \alpha\big(W(T_{\mathcal{P}}) + l(u, v)\big) + (1 - \alpha)\big(D(T_{\mathcal{P}}) + l(u, v) + d(u, r)\big),$$

i.e. the value of the objective, as defined by Equation (3.3), that would result from adding edge (u, v) to $T_{\mathcal{P}}$.

Lemma 3. *Fix $u \in V_{\mathcal{P}}$; then* Cost$(u, v, T_{\mathcal{P}}, \alpha)$ *is minimized by picking* $v =$ Close$(u, T_{\mathcal{P}})$ *over all possible choice of v (nodes not already added to the tree).*

Proof. Suppose $u \in V_{\mathcal{P}}$, $v \notin V_{\mathcal{P}}$, and we add edge (u, v) to $T_{\mathcal{P}}$. The increase in wiring cost is $\alpha \times l(u, v)$. By definition, this is minimized if $v =$ Close$(u, T_{\mathcal{P}})$. When we connect v to u, the shortest path from v to r involves going from v to u and then taking the shortest path from u to r. Thus, the increase in conduction delay is $(1-\alpha) \times d(v, r) = (1-\alpha)\,(l(u, v) + d(u, r))$. Because $d(u, r)$ is unchanged by connecting v to u, the increase in conduction delay is minimized by minimizing $(1 - \alpha) \times l(u, v)$, which is minimized if $v =$ Close$(u, T_{\mathcal{P}})$. Thus, both objective functions are minimized by connecting u to Close$(u, T_{\mathcal{P}})$. □

Thus, at every step, we do not need to consider all possible edges between a node inside the tree and a node outside the tree. The optimal greedy step only involves edges of the form $(u,$ Close$(u, T_{\mathcal{P}}))$, with $u \in V_{\mathcal{P}}$. The full algorithm is shown in Algorithm 3.

Auxiliary data structures: To efficiently compute conduction delays, each node in the tree stores its distance to the root. The root has distance 0 to itself. When we add a node v to the tree via edge (u, v), we set $d(v, r)$ to be $l(v, u) + d(u, r)$. This way, we may access $d(v, r)$ in constant time.

To efficiently compute pairs $(u,$ Close$(u, T_{\mathcal{P}}))$, we store each node's closest neighbors in a hash table, H. In the hash table, a key is a node u currently in the tree, and its value is a list of potential neighbors outside the tree, sorted by distance to u. Initially, we add the root r to H. Whenever we add a new point (including a Steiner point), we add the point as a key in the hash table, and we compute the sorted distances between the point and each point in $\mathcal{P}$ that has not yet been added to the tree. Given a node u, to compute Close$(u, T_{\mathcal{P}})$, we iterate through the sorted list $H[u]$ until we find a node that has not been added to the tree; we then remove all the nodes through which we iterated, including the node we choose, from the sorted list.

Rather than re-computing candidates at every step of the algorithm, we maintain a sorted list of triplets:

$$\text{Candidates} = \{(\text{Cost}(u, \text{Close}(u, T_{\mathcal{P}}), T_{\mathcal{P}}, \alpha), u, \text{Close}(u, T_{\mathcal{P}}))|u \in T_{\mathcal{P}}\}.$$

When we add a node v to the tree, we add v to H and use $H[v]$ to compute Close$(v, T_{\mathcal{P}})$. We then compute how much the Pareto cost of the tree would increase if we add the edge $(v,$ Close$(v, T_{\mathcal{P}}))$. This is equal to:

$$\text{Cost}(v, \text{Close}(v, T_{\mathcal{P}}), T_{\mathcal{P}}, \alpha) = \alpha \times l(v, \text{Close}(v, T_{\mathcal{P}})) + (1 - \alpha) \times (l(v, \text{Close}(v, T_{\mathcal{P}})) + d(v, r)).$$

We then add the triplet $(\text{Cost}(v, \text{Close}(v, T_{\mathcal{P}}), T_{\mathcal{P}}, \alpha), v, \text{Close}(v, T_{\mathcal{P}}))$ to Candidates, which can be done in $\log(n)$ time since Candidates remains in sorted order. Later, when we want to find the pair $(u, \text{Close}(u, T_{\mathcal{P}}))$ whose addition would increase the cost of the tree by the smallest amount, we simply pop the first element from Candidates.

After adding node v to the tree, any triplet of the form $(\text{Cost}(u, v, T_{\mathcal{P}}, \alpha), u, v) \in$ Candidates involving v and any $u \in T_{\mathcal{P}}$ is no longer a valid candidate because v is no longer outside the tree. Thus, we remove all such triplets from Candidates, and find the new candidate involving each u that was removed. To do this, we compute $\text{Close}(u, T_{\mathcal{P}})$ (which can be done efficiently using H) and $\text{Cost}(u, \text{Close}(u, T_{\mathcal{P}}), T_{\mathcal{P}}, \alpha)$ and add the triplet $(\text{Cost}(u, \text{Close}(u, T_{\mathcal{P}}), T_{\mathcal{P}}, \alpha), u, \text{Close}(u, T_{\mathcal{P}}))$ to Candidates (which, again, can be done in $\log(n)$ time).

Running time: There are n loop iterations; each iteration identifies which node to add to the tree by examining the closest neighbors for the $\mathcal{O}(kn)$ nodes that have been currently added to the tree. When adding this node, $k + 1$ nodes are added (including Steiner nodes). For each such node, we compute the distance from the node to each node outside the tree (using a brute-force search that simply iterates through every such pair of nodes and computes their Euclidean distance), and sort these distances. Thus, each step of the algorithm takes $\mathcal{O}(kn + kn \log(n))$ time, and the overall running time is $\mathcal{O}(kn^2 \log(n))$.

To generate the Pareto front we apply the greedy algorithm to values of $\alpha \in \{0.00, 0.01, \ldots, 0.99, 1.00\}$. Examples of arbors generated by the algorithm for different values of α are shown in Figure 3.1C–H.

It remains unclear how close this algorithm gets to finding truly Pareto optimal trees. Minimizing Equation (3.3) is NP-hard; however, some special cases are informative to look at in detail. For $\alpha = 0$, the algorithm will provably generate the optimal tree. At each step, the optimal greedy step will involve connecting the root directly to the node outside of the tree that is closest to the root. Thus, the algorithm will iteratively connect the root to each node until it produces the Satellite tree, which minimizes conduction delay).

For $\alpha = 1$, the optimal tree is the tree that minimizes wiring cost, which by definition is the Steiner tree. In this scenario, the optimal greedy step will involve adding the shortest possible edge that connects a node inside the tree to a node outside the tree. By construction, this means that even if the algorithm ignores the Steiner points it will do no worse than Prim's minimum spanning tree algorithm [208]. Thus, the algorithm will generate a tree with a wiring cost no higher than the wiring cost of the minimum spanning tree,

Algorithm 3 Generate Pareto Front ($\mathcal{P}, \alpha$)

$T_{\mathcal{P}} \leftarrow$ empty graph
$V_{\mathcal{P}} \leftarrow \{r\}$ # add root.
$i \leftarrow 1$ # counter for loop iterations.

while $i < |\mathcal{P}|$ **do**
  # Generate candidate edges to add, and greedily choose the best one.
  Candidates $\leftarrow \{(\text{Cost}(u, \text{Close}(u, T_{\mathcal{P}}), T_{\mathcal{P}}, \alpha), u, \text{Close}(u, T_{\mathcal{P}})) : u \in V_{\mathcal{P}}\}$
  $(u, v) \leftarrow \text{argmin}\{\text{Cost}(u, v, T_{\mathcal{P}}, \alpha) : (\text{Cost}(u, v, T_{\mathcal{P}}, \alpha), u, v) \in \text{Candidates}\}$

  # Connect u to v with Steiner points along the edge.
  $V_{\mathcal{P}} \leftarrow V_{\mathcal{P}} \cup \{v, v_1, v_2, \ldots, v_{10}\}$
  $E_{\mathcal{P}} \leftarrow E_{\mathcal{P}} \cup \{(u, v_1), (v_1, v_2), \ldots, (v_{10}, v)\}$

  $i \leftarrow i + 1$

end while
return $T_{\mathcal{P}}$

which is itself provably within a factor of 2 of the wiring cost of the optimal Steiner tree [209]. In practice, this gap is often much tighter (see e.g., Conn et al. [122], where a similar greedy algorithm was shown to find trees within 5–6% of the Steiner tree for small sets of input points). To further evaluate the optimality of this algorithm, we compare it against a brute-force algorithm [210] for point sets that are small enough to generate the optimal spanning tree, and two other greedy heuristics [199, 200] on larger point sets.

3.4. Results

First, we demonstrate that our greedy algorithm outperforms other algorithms at generating near Pareto optimal topologies. Second, we show that neural arbors lie much closer to the Pareto front than expected by three baselines, and that this result is robust to different assumptions about the locations of the synapses along the arbor. Third, we show that some functional categories of arbors can be differentiated based on how they are structured. Fourth, we use this same network design framework to demonstrate that plant arbors are also Pareto optimal.

3.4.1 The greedy algorithm effectively approximates the Pareto front

As mentioned above, finding Pareto optimal arbors (i.e., minimizing the Equation (3.3)) is NP-hard, forcing us to consider heuristics for generating the Pareto front for large arbors. We first determined how close to Pareto optimal arbors generated by our greedy algorithm were compared to the following three competitor algorithms:

1. **Brute-force algorithm [210]**: This algorithm simply enumerates all possible spanning trees and selects the one that minimizes Equation (3.3) for a given value of α. There are n^{n-2} possible spanning trees, where n is the number of input points. Thus, we only consider the brute-force algorithm for small point sets (up to 8 points).

2. **Khuller's Light Approximate Shortest-Path Tree (LAST) algorithm [199]**: Given a complete graph G with root r and $\beta > 1$, Khuller's algorithm starts with the minimum spanning tree M for G and performs a depth-first search on M starting from r. When the algorithm visits a node u, if the current distance from u to r is more than β times the length of the edge from u to r, it updates M by connecting u to r and removing the edge from u to its previous parent. The algorithm returns the modified version of M that results after visiting each node in the depth-first search. The result is a tree whose conduction delay is provably at most β times the optimal conduction delay, and whose wiring cost is provably at most $\left(1 + \frac{2}{\beta - 1}\right)$-times the optimal wiring cost. Because our problem setup takes $\alpha \in [0, 1]$ as an input, to apply Khuller's algorithm, we must map α to a value of $\beta > 1$. We find that setting $\beta = 1 + \sqrt{\frac{2\alpha W^*}{(1-\alpha)D^*}}$ gives a tree that minimizes Equation (3.3) and retains the algorithm's theoretical bounds on the wiring cost and conduction delay. Here, W^* is the wiring length of the minimum spanning tree of G, and D^* is the conduction delay for the Satellite tree. See Methods for derivation of this mapping.

3. **Karger's algorithm [200]:** This algorithm is equivalent to our greedy algorithm without the use of Steiner points. Comparison against this algorithm allows us to determine the value branch points provide towards minimizing the joint objective. Like our algorithm, Karger's algorithm has been used in geometric settings [200], and has been used for studying neural arbors [173, 174].

None of these three algorithms add branch points. The comparison between our greedy algorithm and these three algorithms thus also highlights the utility of branch points in designing Pareto optimal topologies.

For the test framework, we selected a set of $n + 1$ points ($\mathcal{P}$) randomly from the space $[-10, 10]^3$, and one of these points was selected uniformly at random and designated to be the root (r). For each point set, we varied $\alpha \in (0, 1)$ in step sizes of 0.01 and ran each algorithm above to generate a different Pareto front. We then compared the three Pareto fronts generated by the algorithms by comparing how many trees from each front were partially dominated by trees from one of the other fronts (Methods).

3.4.1.1 Small point sets.

We first examined 411 point sets, each with between 5–8 points. For each point set, we generated a Pareto front (99 trees, using different values of $\alpha \in [0, 1]$) using the brute-algorithm algorithm, Khuller's algorithm, Karger's algorithm, and our greedy algorithm. We computed a Pareto front using each algorithm; we then compared the quality of these Pareto fronts to determine which algorithm generated the Pareto front that was closest to optimal (Methods).

We find that only 7% of the trees generated by our greedy algorithm are dominated by some tree generated by one of the other two algorithms. This is in contrast to 52% for the brute force algorithm, 56% for Karger's algorithm, and 77% for Khuller's algorithm. The brute-force algorithm cannot be dominated by either Karger's or Khuller's algorithms because neither of these algorithms allow branch points. Thus, the brute-force algorithm is actually dominated by the greedy algorithm 48% of the time. The improved performance of the greedy algorithm over brute-force highlights the value of using branch points in finding Pareto optimal arbors.

3.4.1.2 Larger point sets.

Next, we examined 1632 point sets, each with 9–500 points. This size is too large for the brute-force algorithm to run in a reasonable amount of time.

We find that only 13% of the trees generated by our greedy algorithm are dominated by some tree generated by one of the other two algorithms. This is in contrast to 82% for Karger's algorithm, and 99.8% for Khuller's algorithm. Thus, the greedy algorithm generates a more optimal Pareto front compared to Khuller's and Karger's algorithms, which were previously used for this problem [173, 174]. Thus, the greedy algorithm is a better heuristic than previous algorithms on this data, and its use of Steiner points allows for more biological realism.

3.4.2 Neural arbors are Pareto optimal

Next, we used the greedy algorithm to test how close neural arbors were to being Pareto optimal. We obtained 14145 3D arbor reconstructions from Neuromorpho (Methods). For each arbor, the location of the root (r) was designated in the data, and the locations of the synapses (the p_i) were assumed to lie uniformly along the traced arbor (Methods). For each arbor point set, we applied the greedy algorithm for values of $\alpha \in [0, 1]$ to generate the Pareto front. We then evaluated the wiring cost and conduction delay for the actual neural arbor, and the distance of the arbor to the Pareto front (Methods). We also determined the wiring cost and conduction delay for arbors generated by three baseline algorithms — Random, Centroid,

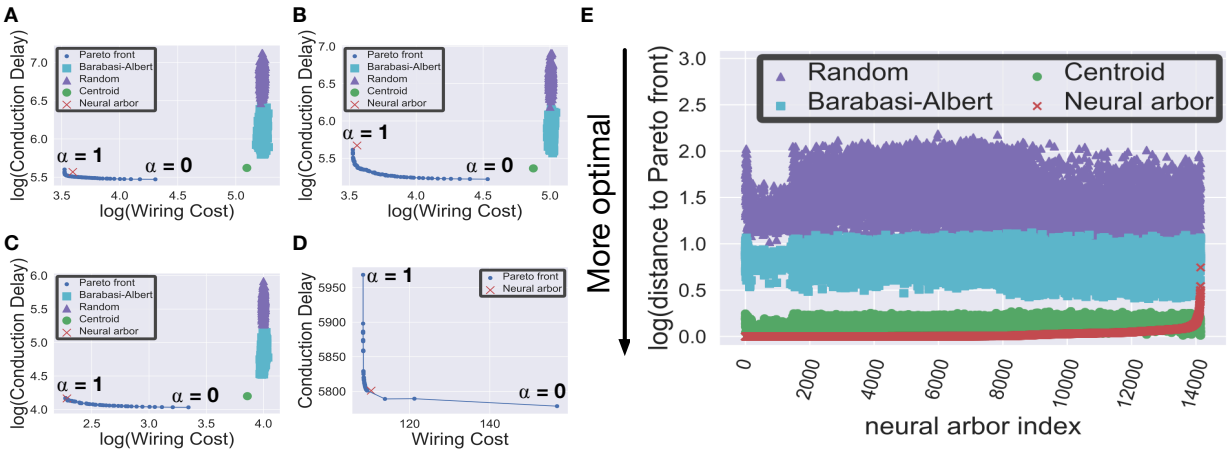


Figure 3.3: **Neural arbors are Pareto optimal.** A–D) Example Pareto front analysis for four arbors: A) sensory receptor cell from the mouse neocortex, B) sensory receptor cell from the mouse dorsal root ganglion, C) amacrine cell from the human retina, and D) sensory receptor neuron from the mouse peripheral nervous system. In each panel, the blue curve show the Pareto front generated using the greedy algorithm. The red 'X' shows the location of the neural arbor, the green circle shows the location of the Centroid tree, the purple triangles show the locations of the uniform random spanning trees (Random), and the aqua squares show the locations of the Barabási-Albert trees. The x-axis is the log wiring length, and the y-axis is the log conduction delay. Neural arbors lie much closer to the Pareto front compared to the other three baselines. In panel D, the arbor is best explained by a value $\alpha = 0.03$ (right-end of the Pareto front). To make its Pareto front clear, we omit the baseline models and we do not use a log scale on either axis. E) Summary of all 14145 arbors. For each arbor (x-axis), we plot the distance to the Pareto front for the neural arbor, the Centroid tree, the average Random tree, and the average Barabási-Albert tree. The x-axis is sorted according to the neural arbor's distance to the Pareto front. Neural arbors tend to fall significantly closer to the Pareto front than the other baselines.

and Barabási-Albert (Methods) — for the same point set to assess how likely it is for the arbor to fall near the Pareto front by chance. These three models generate a tree $T_{\mathcal{P}}$ for a set of points $\mathcal{P}$; however, Random and Barabási-Albert do not attempt to produce a tree with any geometric properties, unlike the Centroid tree. We did not consider other geometric models to assess whether the optimizations made by neural arbors could also be realized by non-geometric models by chance. For example, if in a model a node was simply connected to its nearest-neighbor, this would implicitly optimize for wiring length.

Strikingly, we found that almost all neural arbors fell very close to the Pareto front. In Figure 3.3A–D, we show example Pareto plots for four individual arbors. The Pareto front is shown in blue circles, with one circle per value of $\alpha \in \{0, 0.01, \ldots, 0.99, 1.00\}$. We also mark the location of the neural arbor with a red 'X', and the locations of the three baselines methods all generated using the same input point set.

We found that neural arbors fell much closer to the Pareto front than the three baselines. In Figure 3.3E, we plot the distances from the neural arbor to the Pareto front over all 14145 point sets. In this plot, the y-axis is the log of the distance from the arbor to the Pareto front (lower means more optimal), and the x-axis denotes the arbor number (ranging from 1 to 14145), ordered by distance. Of the 14145 point

Table 3.1: **Comparison of neural arbors to null models**.

Model	Success Rate (Binomial p-value)	Average Ratio (T-Test p-value)
Centroid	0.08 ($p < 10^{-322}$)	1.38 ± 0.20 ($p < 10^{-322}$)
Barabási-Albert	0.0000207 ($p < 10^{-322}$)	5.96 ± 2.36 ($p < 10^{-322}$)
Random	0.00 ($p < 10^{-322}$)	30.22 ± 20.06 ($p < 10^{-322}$)

sets, 92.21% of the neural arbors lay closer to the Pareto front than the Centroid tree; 99.99% lay closer than the Barabási-Albert tree; and 100% lay closer than the Random tree. Using a binomial test (Methods), we find that the neural arbor is significantly more optimal than the three baselines ($p < 10^{-322}$ in all cases). Table 3.1 shows how often arbors generated by each null model were closer to being Pareto optimal than the actual neural arbor (called the 'Success Rate'). Also shown is the average ratio of the null model's distance to the Pareto front versus the neural arbor's distance to the Pareto front.

The results above suggest that nearly all neural arbors lie closer to the Pareto front than other baselines, but they do not describe how much closer. Over all point sets, the average distance ratio (distance to the Pareto front for the baseline divided by distance to the Pareto front for the neural arbor) was 1.38 ± 0.20 for the Centroid tree, 5.96 ± 2.36 for the Barabási-Albert tree, and 30.22 ± 20.06 for the Random tree. In other words, on average, the neural arbors were at least 38% lower for wiring length, or at least 38% for conduction delay, or both, compared to the Centroid tree; at least 496% lower than the average Barabási-Albert tree; and at least 2922% lower than the average random spanning. Using a T-test (Methods), we find that the neural arbors are lie significantly closer to the Pareto front than all baselines ($p < 10^{-324}$), and thus achieve better trade-offs between wiring cost and conduction delay than expected by chance.

3.4.2.1 Synapse locations in Neuromorpho

Neuromorpho provides line segment tracings of neural arbors, but it does not provide the exact synapse locations along the arbor. Based on prior work [211, 212], we placed synapses uniformly along the arbor at a rate of 1 synapse per 0.2 microns for dendrites [212], and 1 synapse per 4 microns for axons [211]. While these rates are likely not constant across all axons and dendrites and across all brain regions and cell types, it provides a way to fairly compare all arbors, which is our primary objective here.

To confirm that our Pareto optimality results are not due to our choice of synapse spacing, we performed the same analysis as described above, except we placed synapses uniformly at a 2x lower rate: 1 synapse per 0.4 microns for dendrites, and 1 synapse per 8 microns for axons. We analyzed a random set of 5871 neural arbors and found that the neural arbor lies closer to the Pareto front than the Centroid tree 95.23% of the time, and closer than the Barabási-Albert tree and the random spanning tree 100% of the time

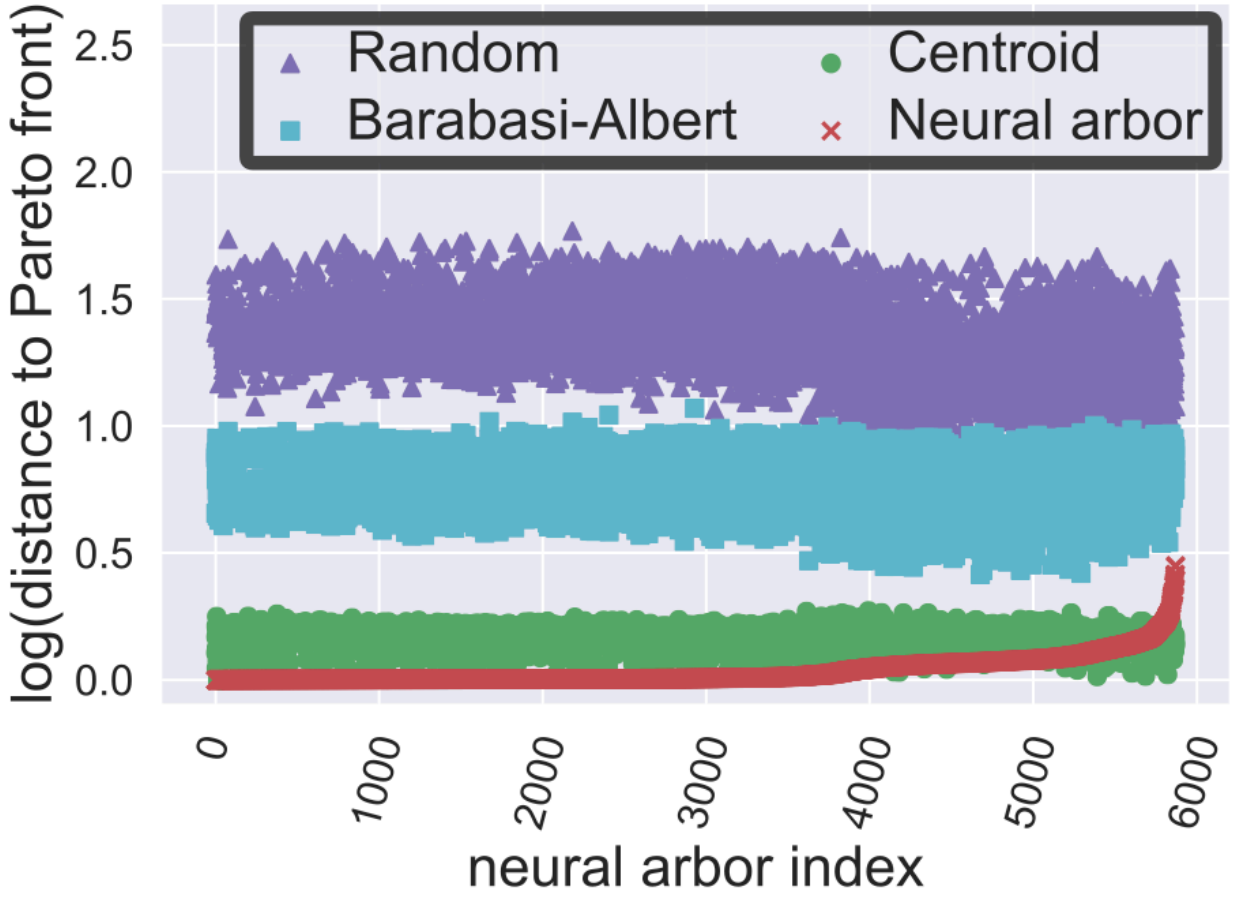


Figure 3.4: **Modifying synapse locations in Neuromorpho arbors.** We ran the same analysis as described above after changing the spacing between synapses to be 0.4 microns for dendrites and 8 microns for axons. We find similar Pareto optimality as we did under our original spacing parameters.

($p < 10^{-322}$ in all cases; Figure 3.4).

3.4.3 Pareto optimality of arbors with known synapse locations

We collected a small set of manually traced arbors with synapse locations precisely identified using a fluorescence synapse labeling technique [213].

These arbors were all from the mouse somatosensory cortex and included 7 dendrites and 1 axon. An example of one arbor is shown in Figure 3.5A–B. For each arbor, we generated the Pareto front using input points only corresponding to the cell body and the identified synapses along the arbor. We found that the neural arbor lied very close to the front, and significantly closer than the baseline models (Figure 3.5C–F). On average, the neural arbor was 1.28 times closer to the Pareto front than the Centroid tree, 6.81 times closer than the Barabási-Albert tree, and 25.87 times closer than the Random tree.

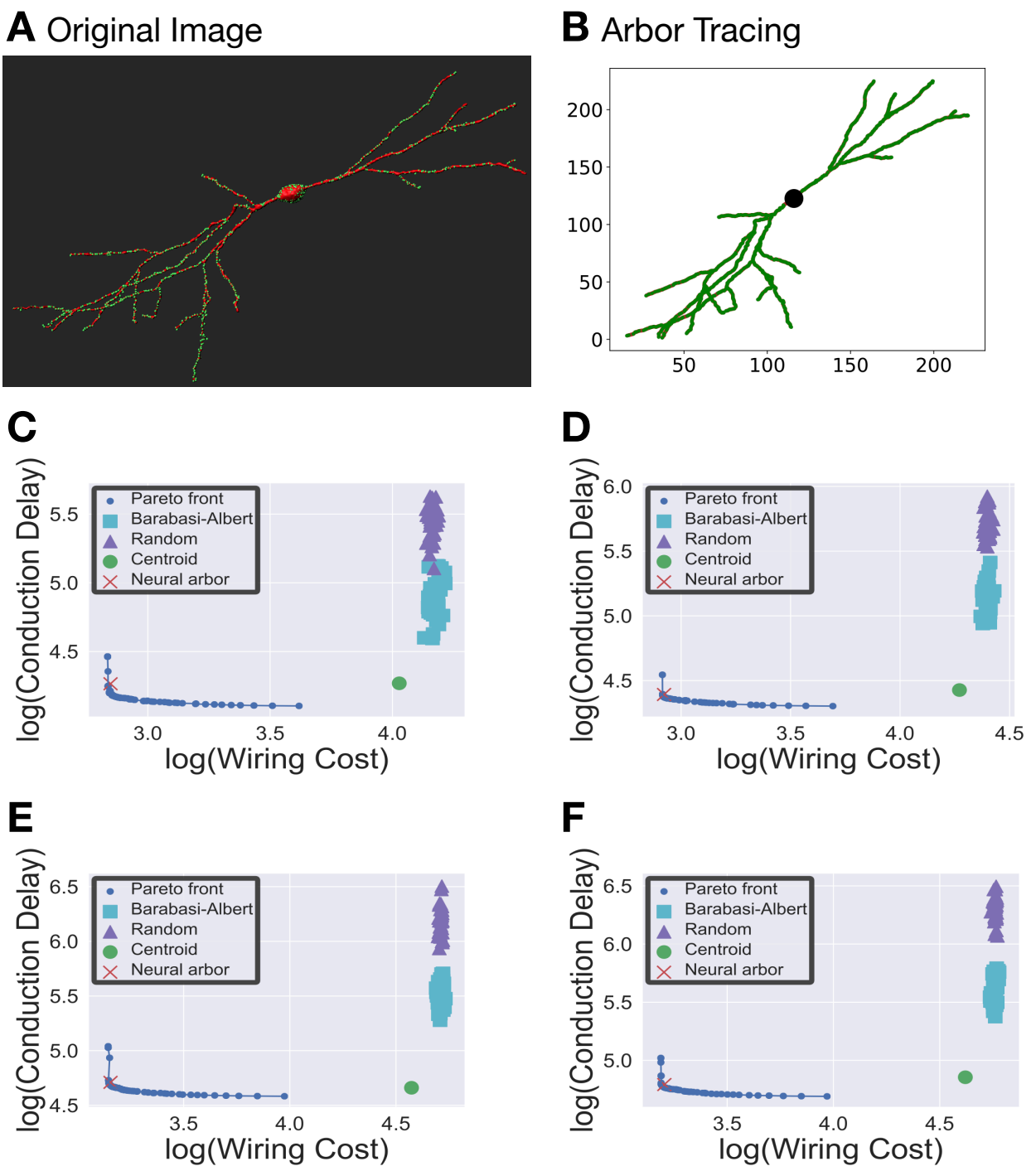


Figure 3.5: **Analysis of Pareto optimality for arbors with tagged synapse locations.** A) 3D image of a partially reconstructed neural arbor. The large red circle in the middle is the cell body. The red lines are the dendrite wiring, and the green dots are the synapse locations. B) A 2D projection of the traced arbor. The large black circle is the cell body, the green dots are the synapses, and the red lines are the edges that trace the dendrite. C–F) Pareto fronts for four arbors. As with the Neuromorpho arbors, the neural arbor is significantly closer to the Pareto front than the baseline models.

This suggests that our results are not likely a consequence of how Neuromorpho neurons are traced, nor how we placed synapses along the Neuromorpho arbor.

3.4.4 Classifying neural arbors by their structure

Are functional differences between arbors reflected in their structure? Here, we study two questions. First, are arbors from different biological categories equally close to being Pareto optimal? Second, do arbors from different biological categories have significantly different values of α (i.e., do they weigh wiring cost and conduction delay differently)? Uncovering such relationships could be used to hypothesize about unknown function given structure.

We first associated each arbor with a biological category based on the type of arbor (apical dendrite, basal dendrite, or axon), the cell type of the neuron (e.g., Purkinje versus granule), or the neurotransmitter type used by the cell (e.g., glutamatergic versus GABAergic). See Table 3.2 for full list of categories. We then associated each arbor with two values: its distance to the Pareto front (Methods, Equation (3.5)), and a value of $\alpha \in [0, 1]$ that comes closest to matching its topology (Methods, Equation (3.6)). Below, we ask if there are systematic differences in these two values for arbors in one category versus another. See methods for formal definition for how to calculate α for a data-derived arbor.

3.4.4.1 Arbor type.

The distance to the Pareto front was smaller for apical dendrites (1.01 ± 0.10) and basal dendrites (1.01 ± 0.02) compared to axons (1.13 ± 0.14) (Figure 3.6A). Error ($\pm$) values represent standard deviations. Using Welch's T-test on the null hypothesis that the average value of this distance is the same, this difference was significant ($p < 10^{-324}$). One possibility for why axons may lie further from the Pareto front than dendrites is that there is a potential third objective, such as space-filling, that axons also need to prioritize (Discussion).

Axons, however, have higher values of α than dendrites: 0.86 ± 0.18 for axons versus 0.26 ± 0.30 for apical dendrites, and 0.52 ± 0.30 for basal dendrites (Figure 3.6B). Using Welch's T-test on the null hypothesis that the average value of α is the same every pair of groups, we find that each pair of groups (axons vs apical dendrites, axons vs apical dendrites, and apical dendrites vs basal dendrites) are significantly different from each other ($p < 10^{-324}$ in all cases).

Thus, apical dendrites appear to prioritize conduction delay over wiring cost; basal dendrites take on intermediate values of α with a more even weighting of both objectives; and axons appear to prioritize wiring cost over conduction delay. How much of this difference is driven by differences in the assumed

Table 3.2: **Arbor categories and sample sizes.**

Category	# of arbors
Arbor type	
apical dendrite	3764
basal dendrite	3187
axon	6361
Cell type	
amacrine	49
basket	149
chandelier	35
fast-spiking	138
golgi	33
granule	1103
interneuron	1341
martinotti	27
mitral	17
neurogliaform	28
projection	72
purkinje	10
pyramidal	3971
sensory receptor	380
stellate	61
Neurotransmitter type	
cholinergic	192
GABAergic	269
glutamatergic	266
nitrergic	311
serontonergic	95

spacing between synapses along the two arbor types? We studied a small set of arbors (119 axons, 47 apical dendrites, and 53 basal dendrites), and placed synapses at a rate of 1 synapse per 4 microns along both axons and dendrites. We find that the average value of α is 0.87 ± 0.13 for axons, 0.46 ± 0.18 for apical dendrites, and 0.37 ± 0.24 for basal dendrites. This provides evidence that even with similar spacing, axons and dendrites prioritize the two objectives differently.

3.4.4.2 Cell type.

Here, we studied differences in arbors belonging to one of 15 well-studied cell types (Figure 3.6C–D; Table 3.2).

The cell types closest to the Pareto front were Golgi cells (1.01 ± 0.03), granule cells (1.01 ± 0.04), and projection cells (1.01 ± 0.04) (Figure 3.6D). By contrast, the cells that lied furthest to the Pareto front included amacrine cells (1.29 ± 0.65) and neurogliaform cells (1.28 ± 0.86) (Figure 3.6C). The cell types with the most extreme α values (i.e., close to 0 or 1) were Golgi cells (0.13 ± 0.32), martinotti cells (0.30 ± 0.41),

Table 3.3: **Classifying neurotransmitters by structure**.

Neurotransmitter	Average scaling distance to Pareto front	Average value of α
Cholinergic	1.107 ± 0.051	0.90 ± 0.11
Gabaergic	1.136 ± 0.224	0.65 ± 0.41
Glutamatergic	1.089 ± 0.072	0.84 ± 0.22
Nitrergic	1.003 ± 0.003	0.24 ± 0.18
Serotonergic	1.106 ± 0.132	0.96 ± 0.06

amacrine cells (0.75 ± 0.32), and stellate cells (0.74 ± 0.33). These results are summarized in Figure 3.6C–D.

While neuroanatomists have long appreciated the visual diversity of arbors from different neural cell types, our work quantifies and provides a signature for some of these differences based on a simple trade-off principle. Potential avenues for further work may involve comparing more detailed aspects of these cell types, such as dendritic signaling, to assess why different types of arbors may prioritize one objective over the other.

3.4.4.3 Neurotransmitter types.

Here, we grouped arbors into one of five classes of neurotransmitter types that the corresponding cell releases (Figure 3.6E–F, Table 3.3).

Nitrergic neurons lay closest to the Pareto front (1.003 ± 0.003) and had significantly smaller values of α (0.24 ± 0.17), indicating a tendency towards optimizing for conduction delay over wiring cost. Other neurotransmitter types lay further from the Pareto front — glutamatergic (1.09 ± 0.07), serotonergic (1.11 ± 0.13), cholinergic (1.11 ± 0.05), GABAergic neurons (1.14 ± 0.22) (Figure 3.6E). GABAergic neurons tended exhibit intermediate α values (0.65 ± 0.41), whereas the other neurotransmitters generally had values of $\alpha \geq 0.84$ — glutamatergic (0.84 ± 0.22), cholinergic (0.90 ± 0.11), and serotonergic (0.96 ± 0.06). The fact that we find systematic clustering of α values for arbors of different neurotransmitter types shows the value in using the Pareto front location to classify neural arbors by function. Curiously, nitrergic neurons are unique topologically: they are significantly closer to being Pareto optimal than any of the other neurotransmitters, and they prioritize minimizing conduction delay much more strongly than other neurotransmitters. Connections between structure and function of these neurons may merit deeper investigation.

We also found a positive correlation between an arbor's distance to the Pareto front and the arbor's α value ($R = 0.42$ using Pearson's correlation coefficient; $p < 10^{-324}$ using a permutation test). In other words, the more strongly a neuron emphasized minimizing wiring cost, the further it lay from the Pareto front. This correlation is positive within every category of arbors we studied (Table 3.2), and the correlation is significantly different from 0 for every category ($p < 0.005$ in all cases using a permutation test).

These results show that certain categories of arbors can be classified using a single value (α), indicating how the arbor trades-off between wiring length and conduction delay. Additional topological descriptors may further help classify these arbors [182]. Our results suggest that during evolution, natural selection may fine-tune arbor growth strategies based on functional requirements, while still following basic design templates that generate Pareto optimal arbors.

3.4.5 Similarities in branching patterns of neural arbors and plant architectures

Nature is abound with branching structures, and it is natural to ask whether other such networks exhibit a similar cost-versus-performance trade-off.

Conn et al. [122] recently identified wiring cost and conduction delay as important design principles constraining plant shoot (above-ground) architectures. Here, synapses are analogous to leaves, and the soma is analogous to the base/root of the plant. Wiring cost for plants corresponds to the amount of resources (e.g., carbon) used to build the architecture. Conduction delay corresponds to a measure of the nutrient transport efficiency (e.g., sugars, water) from the leaves to the root system, and vice-versa. Conn et al. scanned 557 plant architectures (across three species, several growth conditions, and multiple developmental time-points). From each architecture, they extracted the 3D locations of the leaf points ($p_1, p_2, \ldots, p_n$) and the root (r). Using these points, as well as a skeleton tracing of the plant architecture, they showed that plants developed mechanisms to generate Pareto optimal architectures. Here, we re-analyzed their data to compare neural arbors with plant architectures. Specifically, for the set of input points from each plant scan, we varied $\alpha \in [0, 1]$ to generate the Pareto front using the greedy algorithm, and computed the distance to the Pareto front for the plant architecture, as well as the three baseline methods.

Overall, we find that plant architectures are also Pareto optimal (Figure 3.7). We confirm that plant architectures lie closer to the Pareto front than the Centroid tree 96.95% of the time, closer than the Random tree 97.65% of the time, and closer than the Barabási-Albert tree 96.37% of the time (Figure 3.7A). Using a binomial test (Methods), we find that this proportion is statistically significant for all three baselines ($p = 4.59 \times 10^{-136}, p < 10^{-324}$ and $p = 5.0 \times 10^{-324}$, respectively). However, the distribution of α values that represent neural arbors and plant architectures was quite different. The average value of α that best explains plant architectures is 0.25 ± 0.14, as opposed to 0.64 ± 0.35 for neural arbors, indicating that plant architectures tend to prioritize minimizing conduction delay more than wiring cost (Figure 3.7B).

Interestingly, we find that while neural arbors tend to prioritize minimizing wiring cost, plant shoot architectures tend to prioritize minimizing conduction delay (Figure 3.7). We speculate that for plants, space may not be as constrained as for neural arbors packed in the brain, and thus it is possible that wiring

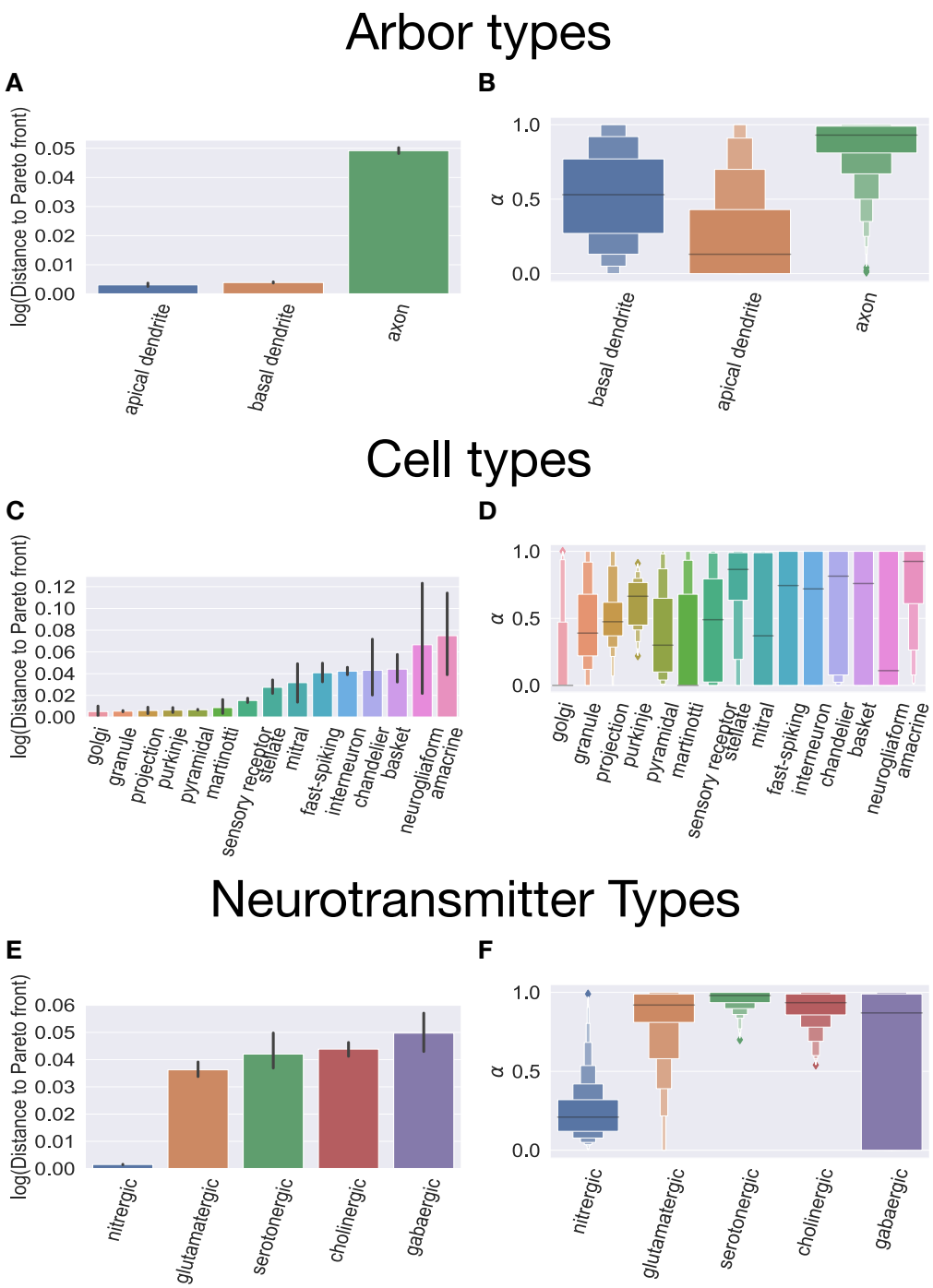


Figure 3.6: **Structure-function differences in neural arbors.** A–B) We categorize arbors in three categories. For each category (x-axis), we plot the distribution of the distances (y-axis) to the Pareto front for arbors in each category in panel A. In panel B, we show a letter plot depicting the distribution of α values for arbors in each category. The middle bar of the letter plot shows the median of the distribution, and the two innermost boxes show the 25th and 75th percentiles respectively; the next two boxes show the 12.5th and 87.5th percentiles, respectively, and so on. Diamonds show individual outliers. C–D) Same analysis with arbors grouped by cell type. E–F) Same analysis with arbors grouped by neurotransmitter type.

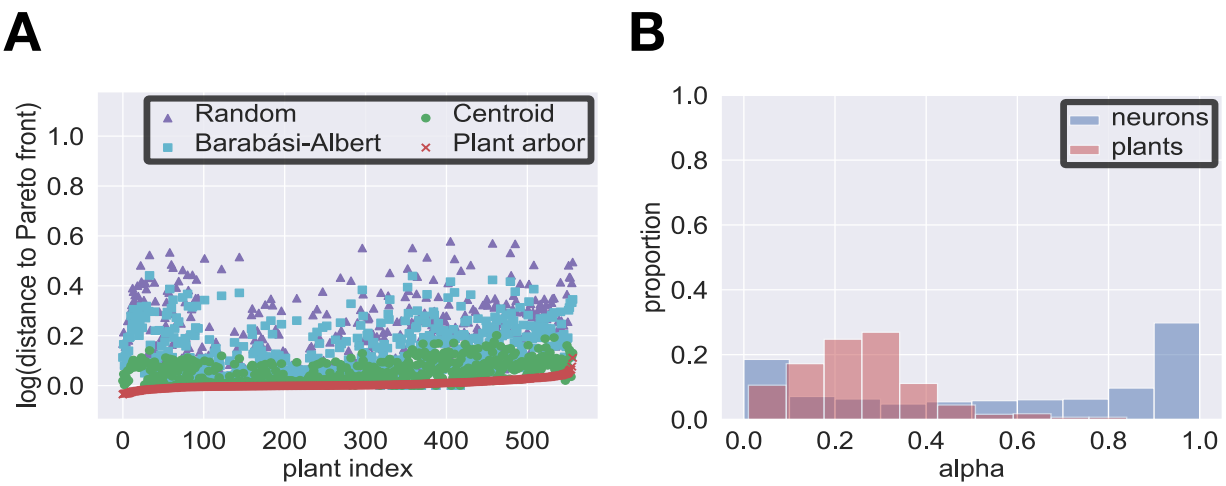


Figure 3.7: **Plant architectures are Pareto optimal but exhibit different trade-offs than neural arbors.** A) The distance to the Pareto front for each plant architecture versus each baseline models. Like neural arbors (compare to Figure 3.3E), plant shoot architectures tend to lie very close to the Pareto front. B) Comparison of α distributions between neural arbors and plant arbors. Low values of α indicate prioritization of conduction delay, and high values indicate prioritization of wiring cost. The majority of neural arbors prioritize wiring cost, whereas plant arbors are biased towards prioritizing conduction delay.

conservation is more important to neurons than plants. Nonetheless, the fact that plants and neurons follow similar design principles was an unexpected result, especially since they are separated by millions of years of evolution.

3.4.6 Analysis of *Arabidopsis* architectures

In follow up work, Conn, et. al. show that the Pareto-optimality framework has the ability to explain variation within the genus *Arabidopsis* [214]. First, the work shows that *Arabidopsis* architectures consistently optimize wiring cost and conduction delay significantly better than would be expected by chance. It then shows how Pareto front location varies between different *Arabidopsis* strains with known genetic differences. This provides clear signposts for future research into genotype-phenotype research for *Arabidopsis*. Moreover, the Pareto-optimality framework is a more consistent and reliable way to characterize *Arabidopsis* plants by morphology; compared to other such metrics, the Pareto front location is less variable within each strain. This work further confirms the value of an algorithmic model of biological transport networks.

3.5. Discussion

We studied a network design trade-off between two competing biological objective functions — minimizing conduction delay and minimizing wiring cost — interpolated between by a single parameter. We

developed a graph-theoretic algorithm to generate near-Pareto optimal topologies that outperformed prior methods. We then showed strikingly that almost all 14145 neural arbors analyzed achieved a level of Pareto optimality significantly better than three other baselines, suggesting that such an optimization is unlikely to occur by chance. We presented instances where specific biological functions (e.g., arbor type, cell type, neurotransmitter type) employed different trade-offs, suggesting that different biological constraints can tune arbor optimization while still obeying the same underlying trade-off principle. Finally, we compared neural arbors to plant architectures and found that both were Pareto optimal despite different trade-offs employed, suggesting broad similarities in branching structures across natural systems that have diverged long ago.

Our work raises three questions for future work. First, our algorithm for computing the Pareto front was designed to find nearly Pareto optimal trees; it was not meant to mimic the growth process used by neural arbors. Specifically, our algorithm used centralized computation when determining which edge to add next to the existing tree. Further, it assumed that the locations of pre- or post-synaptic partners were pre-determined. In reality, arbors likely grow using local rules of computation, and create synapses in locations that are not entirely pre-determined. This raises an open challenge of finding a distributed stochastic growth algorithm to generate Pareto optimal arbors, while incorporating general rules by which neural arbors develop, and including the option to add branch points and allowing some flexibility in where synapses are made. Such an algorithm may involve modifying existing random graph models to produce desirable geometric structures [215]. Second, as discussed in related work, there are several other objectives that may constrain neural arbor topology, such as space-filling, that may help further classify arbor types and better explain instances where arbors lied further from the Pareto front. One advantage of our Pareto optimality framework is that it can naturally be extended to more than 2 objectives. Further, our model did not include wire radii, as these data is not currently widely available for many different types of arbors. These data could also be introduced in our framework using weighted edges. Third, further improvements could be made to our greedy algorithm. For example, instead of only considering one edge to add per iteration, we could add multiple edges per time step. The algorithm would choose the optimal set of k new edges in each time-step; however, such a k-greedy algorithm would increase the run-time by a factor of $O(n^{k+1})$ steps.

3.6. Methods

3.6.1 Adapting Khuller's algorithm for our problem formulation

Here, we describe the steps we take to modify Khuller's light-approximate shortest path tree (LAST) algorithm [199] to solve our problem formulation. Our algorithm takes a value $\alpha \in [0, 1]$ as input, which

describes the weight prioritizing minimizing wiring cost and conduction delay. Khuller's algorithm takes as input a value $\beta \in [1, \infty]$, which serves a similar purpose. Here we describe how to optimally convert our parameter $\alpha \in [0, 1]$ to a parameter $\beta \in [1, \infty]$ such that we may apply Khuller's algorithm to solve our problem.

Let $\mathcal{P} = \{p_1, p_2, \ldots, p_n; r\}$ be points representing the synapses and the soma. A *spanning tree* $T_\mathcal{P}$ has $V(T_\mathcal{P}) = \mathcal{P}$ (i.e., no additional Steiner points). The *minimimum spanning tree* is the spanning tree $M_\mathcal{P}$ that minimizes the wiring cost; the *shortest path tree* $S_\mathcal{P}$ is the spanning tree that minimizes the conduction delay. Both $M_\mathcal{P}$ and $S_\mathcal{P}$ can be constructed efficiently.

Given $M_\mathcal{P}, S_\mathcal{P}$ and $\beta > 1$, Khuller's algorithm produces a tree $T_\mathcal{P}$ such that:

- $W(T_\mathcal{P}) \leq \left(1 + \frac{2}{\beta - 1}\right) W(M_\mathcal{P})$
- $D(T_\mathcal{P}) \leq \beta D(S_\mathcal{P})$

Porting to our formulation, given $\alpha \in [0, 1]$, Khuller's algorithm guarantees a tree $T_\mathcal{P}$ whose total cost is at most:

$$f(\beta) = \alpha \left(1 + \frac{2}{\beta - 1}\right) W(M_\mathcal{P}) + \beta(1 - \alpha) D(S_\mathcal{P}) \tag{3.4}$$

We seek to find $\beta > 1$ which minimizes Equation (3.4). Taking derivatives:

$$\begin{aligned}
\frac{\mathsf{d}f}{\mathsf{d}\beta} &= \frac{-2\alpha W(M_\mathcal{P})}{(\beta - 1)^2} + (1 - \alpha) D(S_\mathcal{P}) = 0 \\
\frac{2\alpha W(M_\mathcal{P})}{(\beta - 1)^2} &= (1 - \alpha) D(S_\mathcal{P}) \\
(\beta - 1)^2 &= \frac{2\alpha W(M_\mathcal{P})}{(1 - \alpha) D(S_\mathcal{P})} \\
\beta &= 1 + \sqrt{\frac{2\alpha W(M_\mathcal{P})}{(1 - \alpha) D(S_\mathcal{P})}}
\end{aligned}$$

From this, we can use Khuller's algorithm to approximate an optimal tree as follows: given $\mathcal{P}$ and α, we compute the Satellite tree ($S_\mathcal{P}$) and minimum spanning tree ($M_\mathcal{P}$), map α to β using the formula above, and then return the result of applying Khuller's algorithm.

Technically, Equation (3.4) is undefined for $\alpha = 1$. In this special case, we output the minimum spanning tree $M_\mathcal{P}$. This does not violate the theoretical guarantees of the algorithm. As $\alpha \to 1$, $\beta \to \infty$, and we get that:

- $W(T_\mathcal{P}) \leq W(M_\mathcal{P})$ (i.e., the same wiring cost as the minimum spanning tree)
- $D(T_\mathcal{P}) \leq \infty \cdot D(S_\mathcal{P})$ (trivially true)

3.6.2 Assessing the quality of the Pareto front generated by an algorithm

Here, we define how we compare the Pareto fronts generated by two algorithms. A tree T_1 is Pareto optimal if it is not partially dominated by any other tree T_2 (i.e., there is no T_2 such that $T_2 \preceq T_1$), according to the two objectives of wiring cost and conduction delay. To measure the quality of the Pareto front generated by one algorithm, we count the number of trees in its Pareto front that are not partially dominated by a tree generated by the second algorithm.

Formally, let T' be a tree generated by one algorithm, and let $\mathcal{T} = \{T_1, T_2, \ldots, T_n\}$ be a set of trees generated by the second algorithm, representing its Pareto front. We say that T' is partially dominated by $\mathcal{T}$ (i.e., $\mathcal{T} \preceq T'$) if $T_i \preceq T'$ for some $T_i \in \mathcal{T}$.

Now, given two sets of Pareto fronts, $\mathcal{T}_1$ and $\mathcal{T}_2$, we count how many trees in $\mathcal{T}_1$ are partially dominated by some tree in $\mathcal{T}_2$:

$$\frac{|\{T_i : T_i \in \mathcal{T}_1, \mathcal{T}_2 \preceq T_i\}|}{|\mathcal{T}_1|}.$$

Larger values of this proportion imply that more trees in $\mathcal{T}_1$ are partially dominated by $\mathcal{T}_2$, meaning that the latter is a closer approximation to the true Pareto front.

This definition can be extended to compare one algorithm's Pareto front $\mathcal{T}_1$ with the Pareto front's of k other algorithms by comparing each tree in $\mathcal{T}_1$ with the union of all the trees from the k other algorithms.

3.6.3 Measuring the distance of an arbor to the Pareto front

Given a set of input points $\mathcal{P}$, our algorithm generates the Pareto front, corresponding to the Pareto optimal topologies for values of $\alpha \in [0, 1]$. Given a neural arbor $T_{\mathcal{P}}$, we seek to compute how far away $T_{\mathcal{P}}$ is from the nearest Pareto optimal topology. Intuitively, this distance corresponds to the amount that the wiring cost and conduction delay for Pareto optimal trees would have to be scaled so that $T_{\mathcal{P}}$ lies on the Pareto front.

Let $T_{\mathcal{P}}$ be a neural arbor, and let $T_{\mathcal{P}}^{\alpha}$ be the Pareto optimal tree on $\mathcal{P}$ for a given value of α. Define the distance from $T_{\mathcal{P}}$ to $T_{\mathcal{P}}^{\alpha}$ to be:

$$s(T_{\mathcal{P}}^{\alpha}, T_{\mathcal{P}}) = \max\left\{\frac{W(T_{\mathcal{P}})}{W(T_{\mathcal{P}}^{\alpha})}, \frac{D(T_{\mathcal{P}})}{D(T_{\mathcal{P}}^{\alpha})}\right\}.$$

The idea is to compute the ratios of the wiring costs and conduction delays for the actual neural arbor versus the optimal tree, and to then set the distance to be the maximum of these two ratios, which is

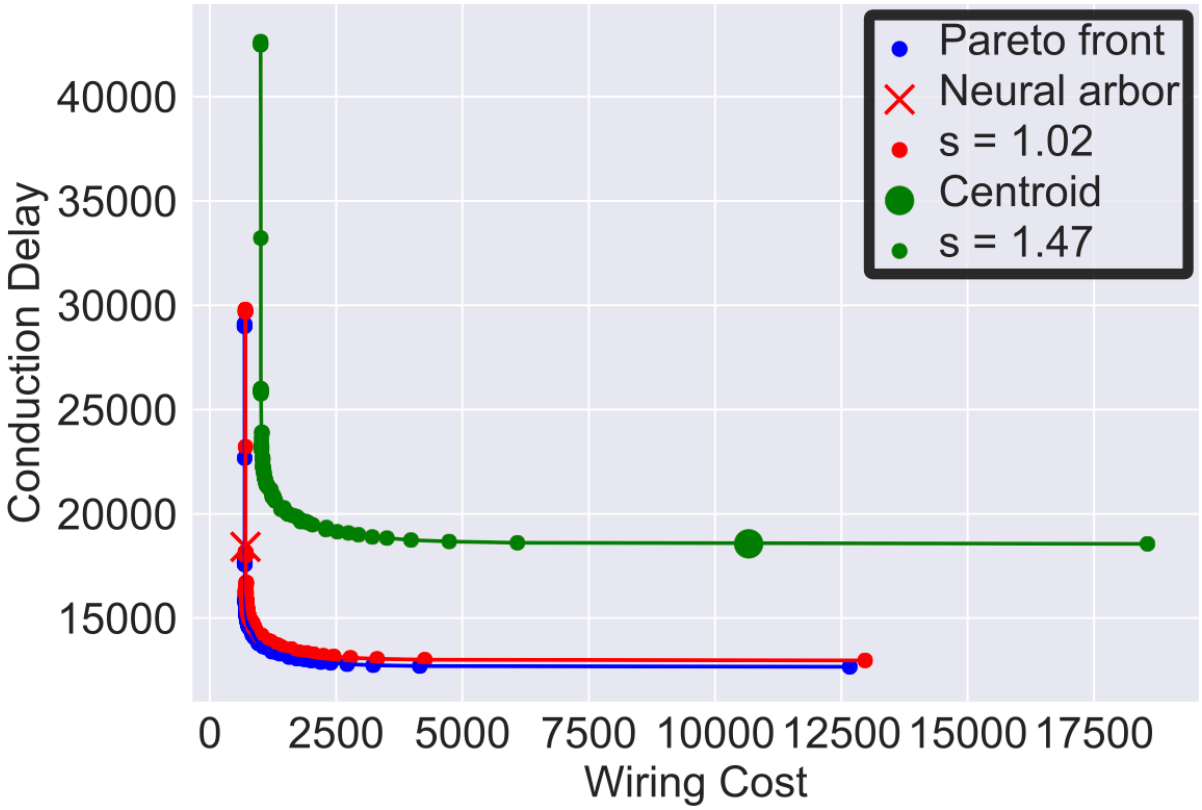


Figure 3.8: **Measuring distance to the Pareto front.**

guaranteed to be ≥ 1. The maximum measures how much one would have to multiply the wiring cost and conduction delay of $T_{\mathcal{P}}^{\alpha}$ until $T_{\mathcal{P}}$ until it is lower than $T_{\mathcal{P}}^{\alpha}$ for both objectives. In Figure 3.8, we illustrate this distance using one example neural arbor. We show the Pareto front (blue), a scaled version of the Pareto front (red) that intersects with a neural arbor, and a further scaled Pareto front (green). For this arbor, its distance to the Pareto front is $s = 1.02$; by comparison, the scaling distance for the Centroid tree is 1.47.

Thus, the scaling *distance* between an arbor $T_{\mathcal{P}}$ and the closest Pareto optimal arbor is:

$$\min_{\alpha_i} s(T_{\mathcal{P}}, T_{\mathcal{P}}^{\alpha_i}), \tag{3.5}$$

where $\alpha_i \in [0, 1]$. If $T_{\mathcal{P}}$ lies exactly on the Pareto front, then its distance is 1.

The value of α that best explains the topology of $T_{\mathcal{P}}$ is:

$$\operatorname{argmin}_{\alpha_i} s(T_{\mathcal{P}}, T_{\mathcal{P}}^{\alpha_i}). \tag{3.6}$$

These two parameters encode global properties of the arbor; the distance tells us how far the arbor

lies from the Pareto front, and the α value tells us how the arbor balances the trade-off.

3.6.3.1 An alternative measure of distance to the Pareto front

An alternative way to measure the distance from a neural arbor to a Pareto optimal tree would be to simply use the ℓ_2 distance between the objectives of the two trees. Let $T_{\mathcal{P}}$ be a neural arbor and $T_{\mathcal{P}}^{\alpha}$ be the Pareto-optimal arbor for some value of α. The ℓ_2 distance between the two trees is:

$$\sqrt{(W(T_{\mathcal{P}}) - W(T_{\mathcal{P}}^{\alpha}))^2 + (D(T_{\mathcal{P}}) - D(T_{\mathcal{P}}^{\alpha}))^2}.$$

The problem with this measure of distance is that there is a significant positive correlation between a neuron's size (number of synapses) and its ℓ_2 distance to the Pareto front ($R = 0.29, p < 10^{-260}$, using a permutation test). This makes it difficult to compare arbors with different sizes. In contrast, using the scaling-based definition of distance above, there is no significant correlation between size and distance to the Pareto front ($R = 0.01, p = 0.56$ using a permutation test).

3.6.3.2 Comparing to baseline networks

How likely is it that a neural arbor would be Pareto optimal by chance? For each set of input points $\mathcal{P}$, we compared against the following three baselines that generate trees on the same set of points:

1. **Random tree:** We construct a complete graph $G_{\mathcal{P}}$ whose nodes are the points in $\mathcal{P}$. We then construct a random spanning tree on $G_{\mathcal{P}}$ using a loop-erased random walk algorithm [216]. For each input point set $\mathcal{P}$, we compute 20 random trees.

2. **Centroid tree:** We compute the centroid of the points in $\mathcal{P}$ and connect every point in $\mathcal{P}$ to this centroid. This tree represents a structurally stable arbor with even weight dispersion [122].

3. **Barabási-Albert tree:** We compute a tree using the Barabási-Albert model [217], with parameters $n = |\mathcal{P}|, k = 1$. First, two random nodes are connected by an edge. Then in each step of the algorithm, an unconnected node is connected to a single node in the tree selected with probability proportional to the node's degree. This uses a "rich-get-richer" model with hubs forming. Because there is no notion of nodes being physically embedded in this model, after constructing this tree, we randomly map nodes in the tree to points in $\mathcal{P}$. For each input point set $\mathcal{P}$, we compute 20 random trees.

Assessing statistical significance. First, we determined how often a tree generated by a baseline model lay closer to the Pareto front than the neural arbor. Specifically, for each baseline and each set of input points, we computed the proportion of trees generated by the baseline that had a smaller distance to the Pareto front than the neural arbor. For Barabási-Albert and random trees, we computed the proportion of the 20 trees that lied closer to the Pareto front than the corresponding neural arbor. We then performed a binomial test on this proportion, using the null hypothesis that the true probability was 0.5 (i.e., the neural arbor and the baseline are equally likely to be closer to the Pareto front compared to the other). The alternative hypothesis was that the true probability was > 0.5 (i.e., the neural arbor lied closer to the Pareto front).

Second, we measured how much farther from the Pareto front trees generated by the baseline model were compared to the neural arbor. We computed the ratio of the average distance between each baseline and the neural arbor. We then performed a T-test on the average value of this ratio, using the null hypothesis that the ratio is 1 (i.e., the null model is, on average, just as close to the Pareto front as the neural arbor). The alternative hypothesis was that the ratio is > 1 (i.e., the null model is, on average, farther from the Pareto front than the neural arbor). For the Barabási-Albert and Random trees, these ratios were pooled over all 20 trees.

3.6.4 Data for neural arbor tracings

To test whether a broad set of neural arbors are Pareto optimal, we collected 14145 digitally reconstructed 3-dimensional neural arbors from the Neuromorpho database [218, 219]. This data comprised 39 species, 296 brain regions, 239 cell types, and 242 labs[1] Each arbor was annotated according to its arbor type, cell type, and neurotransmitter type (Table 3.2). Arbors were selected at random from the database.

While Neuromorpho provides a large benchmark dataset to test the generality of any principle of arbor topology, there are a few caveats to note. First, some of the arbor reconstructions deposited may not be complete (e.g., parts of axons may be cut-off). The size of our dataset, and the number of different labs represented, suggests that we are likely not biasing our analysis towards any particular structure or sampling method; however, in principle, we can only analyze and draw conclusions about the portion of arbors that are available. Second, the radii of each segment is not provided, and thus we treat each segment as 1-dimensional. Third, the precise location of each synapse along the arbor is not known (see below).

[1] There is no standard naming procedure in annotating arbors with these biological categories. For example, different labs may use different names for the same cell type. We do not attempt to standardize this naming procedure here.

3.6.4.1 Synapse locations along the arbor

Each reconstructed arbor contains a point denoting the root (r) of the arbor, and a list of line segments that trace the structure of the arbor. The exact synapse locations $(p_1, p_2, \ldots, p_n)$ are not provided by the tracing, and may be cell-type-specific or even modulated by learning. We follow prior work [173] and distribute synapses uniformly on the arbor. Specifically, for dendrites we space synapses every 0.2 microns [212], and for axons we space synapses every 4 microns [211] along the arbor tracing. An alternative approach would be to randomly select the inter-synapse space interval from a Gaussian or an exponential distribution with the appropriate means and variances, though this would require that for each arbor we generate many versions of the Pareto front per sampling of synapse points, which would greatly increase computational complexity. We show, however, that other inter-synapse spacing values generate similar optimality results (Figure 3.4). We also studied a small set of arbors with fluorescently-tagged synapse locations, and find similar results (Figure 3.5).

Using this approach, our arbors ranged from 100–9999 points, and the total lengths of arbors ranged from 182.57–25753.34 microns.

3.7. Data availability statement

Code is available at `https://github.com/arjunc12/neurons`. Tracings of neural arbors are available on `neuromorpho.org`.

3.8. Acknowledgements

Chapter 3, in full, is a reprint of material as it appears in Proceedings of the Royal Society B. Chandrasekhar, Arjun; Navlakha, Saket, The Royal Society, 2019. The dissertation author was the primary investigator and author of this paper.

Conclusion

In this dissertation I introduced the history of algorithms in nature, i.e., modelling biological systems as distributed algorithms that attempt to solve optimization problems. I showed how drawing inspiration from biological systems has allowed computer scientists to discover innovative algorithmic techniques, and how algorithmic perspectives can help biologists make precise predictions about how these biological systems function. I have contributed to the field of algorithms in nature by mining computational insights from data on turtle ant colonies and neural arbor morphologies.

In Chapter 1, I defined a mathematical model to describe how turtle ants forage, and what problems they must solve. I showed how data on turtle ant colonies can be used to derive a distributed algorithm to repair breaks in a foraging network; the algorithm that I derive requires extremely few resources, and is robust to a variety of potential hazards that turtle ants may face. I showed how an algorithmic model of turtle ants can be used to identify potential critical features of their foraging behavior.

In Chapter 2, I used my algorithmic model of turtle ant foraging to generate testable hypotheses about the higher level design principles that guide their foraging behavior. I analyzed data on turtle ant trail networks from several days, and to see whether turtle ants favored trail networks with fewer nodes, shorter edges, or physically favorable nodes. I showed that turtle ants trail networks tend to have fewer nodes and more physically advantages nodes (but not shorter edges) than would be expected by chance. I concluded that turtle ants take advantage of environmental variation to avoid certain parts of the vegetation and keep the colony coordinated.

In Chapter 3, I developed a mathematical model of neural arbors as transport networks that manage trade-offs between efficiency and economy. I defined a framework for measuring how well individual arbors manage trade-offs, and analyzed morphological data on a large cross section of neurons. I confirmed the hypothesis that neural arbors manage this trade-off in a near-optimal manner, thus demonstrating the potential for using neuroscience as a tool to design better distributed computing algorithms. Finally, I showed how an algorithmic model of neural arbors can help produce hypotheses about structure-function relationships in neural arbors, as well as similarities between neural arbors and plant arbors (and possibly other systems).

Common themes

My research describes ways in which turtle ants and neural arbors have evolved distributed algorithms for fundamental network routing problems. In doing so, I identify several emerging themes that can inform future research into nature-inspired algorithms. First, my work highlights the ubiquity of routing networks in nature, and suggests that distributed approaches to routing algorithms merit deeper investigation. Turtle ants and neural arbors epitomize the potential robustness and scalability benefits that come with a decentralized algorithm; the variability among neural arbors from different cell types also speaks to the remarkable potential for adaptability in distributed algorithms. Research on routing network design will benefit from allocating more investigative resources into these two systems, as well as the other algorithmic biological systems.

Second, my research highlights novel ways in which computer science can contribute to advances in biology. The power of computational techniques for analyzing large biological datasets has been well-documented. However, simply using an algorithmic framework to model biological systems can be beneficial. An algorithmic model, by definition, produces precise predictions about what *should* occur, and gives clear signposts for how to design experiments. An algorithmic model of turtle ants predicts that turtle ants cannot forage effectively without down-regulating pheromone on a dead end; an algorithmic model of neural arbors predicts that certain types of neurons should have functions that require more efficient signal conduction. Going forward, my work may provide a blueprint for innovation in the study of other biological systems.

Finally, the need for managing trade-offs is a re-occurring theme in my research. Both turtle ants and neural arbors attempt to balance multiple objectives when constructing routing networks: turtle ants manage trade-offs between forming short paths between nests versus avoiding junctions where ants can get lost, and neural arbors manage trade-offs between efficiently conducting signals versus conserving material. Moreover, I show that when repairing breaks in foraging trails, turtle ants face inherent trade-offs between targeted optimality and general robustness. This is a pattern common to other biological systems [59]: it is almost always impossible to find a single optimal phenotype, so nature selects for phenotypes that perform well for multiple objectives, if not optimally for any one objective. This work highlights the necessity of considering several possible optimization objectives when modelling any biological system.

Future Work

My research raises open questions about how turtle ant colonies and neural arbors construct routing networks from scratch. These questions involve refining the model of computation for each system, and deriving a novel algorithm based on experimental data and algorithmic techniques. The algorithmic models I have developed may also prove useful in probability theory and neuroanatomy. Solving these problems will make it easier to design efficient optimization algorithms, and will make it easier to understand the behavior of turtle ants, neural arbor growth, and perhaps many other related systems.

What is an appropriate model of computation for turtle ants?

My research uses a simple model of computation for how each individual ant moves about the graph: each ant may make its next choice based on adjacent edge weights and memory of the previous node it was at, and it may temporarily stop laying pheromone if it is going back from a dead end. I seek to research biologically reasonable extensions to this model. Some natural questions include: should simulated ants should more than one time step of memory? Should they lay variable amounts of pheromone, and/or whether they can lay multiple types of pheromone (such as the 'no-entry' signal used by pharaoh ants [81])? Do turtle ants possess a sense of 'direction', i.e., can they determine whether certain edges lead away from the destination nest? This feature could be part of how turtle ants avoid getting trapped in cycles.

Refining these aspects of the turtle ant computational model can be done in two ways. The obvious method would be to design experiments to test what abilities individual turtle ants possess. The other method would be the one described in Chapter 1: mathematically demonstrate that certain features are essential to the success of the algorithm, strongly implying that this feature is present in observed ants.

What distributed algorithm that turtle ants use to form trail networks?

I propose several biologically plausible distributed algorithms for repairing breaks in trail networks. Ultimately, the goal of the problem is find an all-purpose algorithm that can create, maintain, and repair a trail network. Each ant will start at a nest, and be capable of recognizing nests. The goal is for the ants to form a trail network connecting all of the nests. There should not be any cycles that do not include multiple nests. Ideally there should be at least one parameter, such as total time or total number of ants, that can be modified to make the algorithm arbitrarily likely to succeed.

To fully mimic the turtle ant foraging, the algorithm should be able to run continuously and respond to changes in the network. This includes finding shorter trails if they emerge, building and pruning temporary

trails to food sources, and repairing breaks caused by disruptions to an edge.

How should transition indices be incorporated into the mathematical model?

My second chapter shows that turtle ant trail networks include nodes with lower transition indices than would be expected by chance. We hypothesize that this is because these transitions can be crossed in fewer ways, and thus can be incorporated into the foraging network more quickly. The next step is to design field experiments to confirm this explanation or propose an alternate one. Assuming our hypothesis is true, there are multiple ways to incorporate this into our line graph model of turtle ant foraging. One possible way to model the transition index would be that turtle ants can only sense a fraction of the pheromone when traversing a high transition index edge. Another way would be to model the turtle ant vegetation as a multi-graph; higher transition indices correspond to more multi-edges between two nodes in the line graph. When an ant crosses a multi-edge, it only senses pheromone from one of the multi-edges (rather than a constant fraction of the total pheromone). Both of these changes would bias ants towards low transition indices for which they can sense all of the pheromone.

What are the connections between turtle ant foraging and random walk theory?

In Chapter 1, I highlight connections between turtle ants and probability theory. Turtle ants are essentially a fully realized version of the edge-reinforced random walks that have been well studied by probability theorists. Just like turtle ants may be able to teach us about distributed algorithms, they may be able to teach us about random walk theory. An interesting hypothesis to test would be whether turtle ants have evolved behavior to take advantage of asymptotic properties of edge-reinforced random walks. The classic edge-reinforced random walk includes one walker, and non-decreasing edge weights. Turtle ants represent a way for probability theorists study an edge-reinforced random walk with multiple walkers and exponentially decaying edge weights. Two important problems are determining many (i.e., a finite set, all but a finite set, etc.) edge weights asymptotically go to infinity, and how many edge weights asymptotically decay to 0.

Do turtle ant trail networks resemble Steiner trees?

It has been shown that other ant species form trail networks that resemble Euclidean Steiner trees [19] – that is, they connect nests using the trail with the smallest possible length. A natural questions is whether turtle ant trail networks also resemble Steiner trees. Unlike the Euclidean Steiner tree, the turtle ant Steiner

tree doesn't necessarily need to minimize physical length; it could minimize total nodes, or total transition index of all transitions used. It may also minimize the average transition index, which, as I show in Chapter 2, is also an NP-hard problem. It is possible the turtle ants achieve some Pareto-optimal solution between these objectives; or they may not construct Steiner trees, but instead form multiple paths that start and end at the same pair of nests.

How do trail networks differ among species in the Cephalotes genus?

My research to this point has focused on the species *Cephalotes goniodontus*. The results I present raise several questions about other ant species within the arboreal *Cephalotes* genus. Field data can tell us whether other arboreal ants use a similar repair algorithm, whether they use similar parameters, and whether their trail networks optimize the same objectives. It would be particularly interesting to see if other species are equally biased towards low transition indices. It is possible that other species have more ability to sense pheromone across a node, or forage in less tangled vegetation, leading to trail networks that do not prioritize low transition indices.

How should conduction delay be measured?

I measured conduction delay by measuring the total length from the synapse to the cell body. In reality, there may be other factors that affect signal propagation. This may include the number of bifurcations the arbor must traverse, the radius of the wire, and the material used to construct the arbor. A more accurate model of conduction delay will yield even stronger analysis of neural arbor morphology.

What is an appropriate model of computation for neural arbors?

While my work confirms that neural arbors achieve Pareto optimal trade-offs between wiring cost and conduction delay, it doesn't explain how. The first step to doing so involves defining a model for how the neural arbor performs computation. This will likely involve viewing the neural arbors as a continuously growing tree, in which every tree tip is a distributed process. This will also involve defining what operations each distributed process can perform; these may include extending a trial branch, bifurcating into two processes, and scanning within a fixed radius for synaptic connections.

What distributed algorithm to neural arbors use to form Pareto-optimal morphologies?

After defining an appropriate computational model, the natural next step is to algorithm that is plausible within that model. The algorithm should take α as an input and output an arbor that is close to α on the Pareto front. I have experimented with algorithms in which the arbor repeatedly extends a trial branch from one of its tips, and either connects to all synapses in the area, or retracts the branch if no synapses are found. I initially found that a simple algorithm is not as optimal as neural arbors, and it doesn't generate the same diversity of morphologies (most arbors resembled minimum wiring cost arbors). This suggests the need for a more sophisticated distributed algorithm.

What other objectives do neural arbors seek to optimize?

I find that some types of neural arbors are consistently farther from being Pareto optimal than other types of arbors. While it is possible that these arbors are simply worse at optimizing the two objectives, it is more likely that these arbors are optimizing other objectives. The beauty of our Pareto front model is that it can easily be extended to multiple objectives. Neuroscience research can be used to refine the model by to determining other plausible design principles, and test whether certain neural arbors that perform poorly for wiring cost or conduction delay manage to outperform other neurons at different objectives.

How does Pareto front location correspond to arbor function?

I find that different types of arbors prioritize objectives differently. This is likely related to the functions of those neurons. A hypothesis worth exploring is whether arbors that prioritize conduction delay because the brain requires those arbors to transmit signals as quickly as possible, and whether efficient signal conduction is less important for arbors that prioritize wiring cost.

Bibliography

[1] Lynch, N. A. *Distributed Algorithms*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1996.

[2] Gordon, D. M. “The expandable network of ant exploration”. *Animal Behaviour* 50.4 (1995), pp. 995–1007.

[3] Gordon, D. M. “Local regulation of trail networks of the arboreal turtle ant, Cephalotes goniodontus”. *The American Naturalist* 190.6 (2017), E156–E169.

[4] Gordon, D. M. “The ecology of collective behavior in ants”. *Annual review of entomology* 64 (2019), pp. 35–50.

[5] Raine, N. E., Ings, T. C., Dornhaus, A., Saleh, N., and Chittka, L. “Adaptation, genetic drift, pleiotropy, and history in the evolution of bee foraging behavior”. *Advances in the Study of Behavior* 36 (2006), pp. 305–354.

[6] Weimerskirch, H., Martin, J., Clerquin, Y., Alexandre, P., and Jiraskova, S. “Energy saving in flight formation”. *Nature* 413.6857 (2001), p. 697.

[7] Hua, J. Y. and Smith, S. J. “Neural activity and the dynamics of central nervous system development”. *Nature neuroscience* 7.4 (2004), p. 327.

[8] Cook, W., Lovász, L., and Seymour, P. D. *Combinatorial optimization: papers from the DIMACS Special Year*. Vol. 20. American Mathematical Soc., 1995.

[9] Ehrgott, M. and Gandibleux, X. “A survey and annotated bibliography of multiobjective combinatorial optimization”. *OR-Spektrum* 22.4 (2000), pp. 425–460.

[10] Nguyen, U. T. and Xu, J. “Multicast routing in wireless mesh networks: Minimum cost trees or shortest path trees?” *IEEE Communications Magazine* 45.11 (2007).

[11] Delves, P. J. and Roitt, I. M. “The immune system”. *New England journal of medicine* 343.1 (2000), pp. 37–49.

[12] Brambilla, M., Ferrante, E., Birattari, M., and Dorigo, M. “Swarm robotics: a review from the swarm engineering perspective”. *Swarm Intell* 7.1 (2013), pp. 1–41.

[13] Raghavan, P. “Information retrieval algorithms: A survey”. *Proceedings of the eighth annual ACM-SIAM symposium on Discrete algorithms*. Society for Industrial and Applied Mathematics. 1997, pp. 11–18.

[14] Dasgupta, S., Stevens, C. F., and Navlakha, S. “A neural algorithm for a fundamental computing problem”. *Science* 358.6364 (2017), pp. 793–796.

[15] Kitano, H. “Biological robustness”. *Nature Reviews Genetics* 5.11 (2004), p. 826.

[16] Kitano, H. “Towards a theory of biological robustness”. *Molecular systems biology* 3.1 (2007).

[17] Alwan, H. and Agarwal, A. “A survey on fault tolerant routing techniques in wireless sensor networks”. *2009 Third International Conference on Sensor Technologies and Applications*. IEEE. 2009, pp. 366–371.

[18] Prabhakar, B., Dektar, K. N., and Gordon, D. M. “The regulation of ant colony foraging activity without spatial information”. *PLoS Comput Biol* 8.8 (2012), e1002670.

[19] Latty, T., Ramsch, K., Ito, K., Nakagaki, T., Sumpter, D. J., Middendorf, M., and Beekman, M. “Structure and formation of ant transportation networks”. *J R Soc Interface* 8.62 (2011), pp. 1298–1306.

[20] Banerjee, S. and Moses, M. “Scale invariance of immune system response rates and times: perspectives on immune system architecture and implications for artificial immune systems”. *Swarm Intelligence* 4.4 (2010), pp. 301–318.

[21] Ben-Jacob, E., Shmueli, H., Shochet, O., and Tenenbaum, A. “Adaptive self-organization during growth of bacterial colonies”. *Physica A: Statistical Mechanics and Its Applications* 187.3-4 (1992), pp. 378–424.

[22] Heller, N. E. and Gordon, D. M. “Seasonal spatial dynamics and causes of nest movement in colonies of the invasive Argentine ant (Linepithema humile)”. *Ecological Entomology* 31.5 (2006), pp. 499–510.

[23] Tian, Y., Sannomiya, N., and Nakamine, H. “A simulation study on adaptability to environmental variations based on ecological systems”. *Proceedings of the 1999 American Control Conference (Cat. No. 99CH36251)*. Vol. 3. IEEE. 1999, pp. 1759–1763.

[24] Suddath, C. “A brief history of: Velcro”. *Time Magazine* (2010).

[25] Brubaker, C. E. and Messersmith, P. B. “The present and future of biologically inspired adhesive interfaces and materials”. *Langmuir* 28.4 (2012), pp. 2200–2205.

[26] Rozenberg, G., Bäck, T., and Kok, J. N. *Handbook of natural computing*. Springer, 2012.

[27] Navlakha, S. and Bar-Joseph, Z. “Distributed Information Processing in Biological and Computational Systems”. *Commun. ACM* 58.1 (2014), pp. 94–102.

[28] Brabazon, A., O’Neill, M., and McGarraghy, S. *Natural computing algorithms*. Springer, 2015.

[29] Navlakha, S. and Bar-Joseph, Z. “Distributed information processing in biological and computational systems”. *Communications of the ACM* 58.1 (2015), pp. 94–102.

[30] Afek, Y., Alon, N., Bar-Joseph, Z., Cornejo, A., Haeupler, B., and Kuhn, F. “Beeping a maximal independent set”. *Distributed computing* 26.4 (2013), pp. 195–208.

[31] Emek, Y., Smula, J., and Wattenhofer, R. “Stone age distributed computing”. *arXiv preprint arXiv:1202.1186* (2012).

[32] Aspnes, J. and Ruppert, E. “An introduction to population protocols”. *Middleware for Network Eccentric and Mobile Applications*. Springer, 2009, pp. 97–120.

[33] Protic, J., Tomasevic, M., and Milutinovic, V. “Distributed shared memory: Concepts and systems”. *IEEE Parallel & Distributed Technology: Systems & Applications* 4.2 (1996), pp. 63–71.

[34] Colorni, A., Dorigo, M., and Maniezzo, V. “Distributed optimization by ant colonies”. *Proceedings of the first European conference on artificial life*. Vol. 142. Paris, France. 1991, pp. 134–142.

[35] Stützle, T. and Dorigo, M. “ACO algorithms for the traveling salesman problem”. *Evolutionary algorithms in engineering and computer science* (1999), pp. 163–183.

[36] Tsutsui, S. “Ant colony optimization with cunning ants”. *Transactions of the Japanese Society for Artificial Intelligence* 22 (2007), pp. 29–36.

[37] López-Ibáñez, M., Stützle, T., and Dorigo, M. “Ant Colony Optimization: A Component-Wise Overview”. *Handbook of Heuristics*. Ed. by Martí, R., Panos, P., and Resende, M. G. Cham: Springer International Publishing, 2016, pp. 1–37.

[38] Karaboga, D. *An idea based on honey bee swarm for numerical optimization*. Tech. rep. Technical report-tr06, Erciyes university, engineering faculty, computer engineering department, 2005.

[39] Eberhart, R. C. and Kennedy, J. “A new optimizer using particle swarm theory”. *Proceedings of the sixth international symposium on micro machine and human science*. Vol. 1. New York, NY. 1995, pp. 39–43.

[40] Gandomi, A. H. and Alavi, A. H. “Krill herd: a new bio-inspired optimization algorithm”. *Communications in nonlinear science and numerical simulation* 17.12 (2012), pp. 4831–4845.

[41] Zheng, J., Chen, Y., and Zhang, W. “A survey of artificial immune applications”. *Artificial Intelligence Review* 34.1 (2010), pp. 19–34.

[42] Andrews, R., Diederich, J., and Tickle, A. B. “Survey and critique of techniques for extracting rules from trained artificial neural networks”. *Knowledge-based systems* 8.6 (1995), pp. 373–389.

[43] Liu, W., Wang, Z., Liu, X., Zeng, N., Liu, Y., and Alsaadi, F. E. “A survey of deep neural network architectures and their applications”. *Neurocomputing* 234 (2017), pp. 11–26.

[44] Zhang, Q., Yang, L. T., Chen, Z., and Li, P. “A survey on deep learning for big data”. *Information Fusion* 42 (2018), pp. 146–157.

[45] Rojas, R. “The backpropagation algorithm”. *Neural networks*. Springer, 1996, pp. 149–182.

[46] Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I., and Fergus, R. “Intriguing properties of neural networks”. *arXiv preprint arXiv:1312.6199* (2013).

[47] Hidasi, B., Karatzoglou, A., Baltrunas, L., and Tikk, D. “Session-based recommendations with recurrent neural networks”. *arXiv preprint arXiv:1511.06939* (2015).

[48] Mardis, E. R. “The impact of next-generation sequencing technology on genetics”. *Trends in genetics* 24.3 (2008), pp. 133–141.

[49] Tønnesen, J. and Nägerl, U. V. “Superresolution imaging for neuroscience”. *Experimental neurology* 242 (2013), pp. 33–40.

[50] Heath, M. T. *Scientific computing: an introductory survey*. Vol. 80. SIAM, 2018.

[51] Kays, R., Crofoot, M. C., Jetz, W., and Wikelski, M. “Terrestrial animal tracking as an eye on life and planet”. *Science* 348.6240 (2015), aaa2478.

[52] Cao, Y., Jiang, T., and Girke, T. “Accelerated similarity searching and clustering of large compound sets by geometric embedding and locality sensitive hashing”. *Bioinformatics* 26.7 (2010), pp. 953–959.

[53] Cannon, J. L., Moses, M. E., Byrum, J. R., Mrass, P., Fricke, G. M., and Tasnim, H. “Modeling T Cell Motion in Tissues During Immune Responses”. *Biophysical Journal* 116.3 (2019), 322a.

[54] Malíčková, M., Yates, C., and Boďová, K. “A stochastic model of ant trail following with two pheromones”. *arXiv:1508.06816* (2015).

[55] Moses, M. E., Cannon, J. L., Gordon, D. M., and Forrest, S. “Distributed adaptive search in T cells: Lessons from ants”. *Frontiers in Immunology* 10 (2019), p. 1357.

[56] Gordon, D. M. “The dynamics of foraging trails in the tropical arboreal ant Cephalotes goniodontus”. *PLoS ONE* 7.11 (2012), e50472.

[57] Hochman, H. M. and Rodgers, J. D. “Pareto optimal redistribution”. *The American economic review* 59.4 (1969), pp. 542–557.

[58] Ngatchou, P., Zarei, A., and El-Sharkawi, A. “Pareto multi objective optimization”. *Proceedings of the 13th International Conference on, Intelligent Systems Application to Power Systems*. IEEE. 2005, pp. 84–91.

[59] Shoval, O., Sheftel, H., Shinar, G., Hart, Y., Ramote, O., Mayo, A., Dekel, E., Kavanagh, K., and Alon, U. “Evolutionary trade-offs, Pareto optimality, and the geometry of phenotype space”. *Science* 336.6085 (2012), pp. 1157–1160.

[60] Tero, A., Takagi, S., Saigusa, T., Ito, K., Bebber, D. P., Fricker, M. D., Yumiki, K., Kobayashi, R., and Nakagaki, T. “Rules for biologically inspired adaptive network design”. *Science* 327.5964 (2010), pp. 439–442.

[61] Navlakha, S., Barth, A. L., and Bar-Joseph, Z. “PLOS Computational Biology: Decreasing-Rate Pruning Optimizes the Construction of Efficient and Robust Distributed Networks”. *PLoS ONE* (2015). (Accessed on 07/13/2016).

[62] Brabazon, A., O’Neill, M., and McGarraghy, S. *Natural Computing Algorithms (Natural Computing Series)*. Springer, 2015.

[63] Bottinelli, A., Wilgenburg, E. van, Sumpter, D. J., and Latty, T. “Local cost minimization in ant transport networks: from small-scale data to large-scale trade-offs”. *J R Soc Interface* 12.112 (2015).

[64] Lanan, M. C., Dornhaus, A., and Bronstein, J. L. “The function of polydomy: the ant Crematogaster torosa preferentially forms new nests near food sources and fortifies outstations”. *Behavioral Ecology and Sociobiology* 65.5 (2011), pp. 959–968.

[65] Newman, M. *Networks: An Introduction*. New York, NY, USA: Oxford University Press, Inc., 2010.

[66] Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. *Introduction to algorithms*. Vol. 6. MIT press Cambridge, 2001.

[67] Gordon, D. M. “The evolution of the algorithms for collective behavior”. *Cell Syst* 3.6 (2016), pp. 514–520.

[68] Gomez, C., Gilabert, F., Gomez, M. E., López, P., and Duato, J. “Deterministic versus adaptive routing in fat-trees”. *Parallel and Distributed Processing Symposium, 2007. IPDPS 2007. IEEE International*. IEEE. 2007, pp. 1–8.

[69] Middleton, E. J. and Latty, T. “Resilience in social insect infrastructure systems”. *Journal of The Royal Society Interface* 13.116 (2016), p. 20151022.

[70] Flanagan, T. P., Pinter-Wollman, N. M., Moses, M. E., and Gordon, D. M. “Fast and flexible: Argentine ants recruit from nearby trails”. *PloS one* 8.8 (2013), e70888.

[71] Garnier, S., Guérécheau, A., Combe, M., Fourcassié, V., and Theraulaz, G. “Path selection and foraging efficiency in Argentine ant transport networks”. *Behavioral Ecology and Sociobiology* 63.8 (2009), pp. 1167–1179.

[72] Dussutour, A., Fourcassie, V., Helbing, D., and Deneubourg, J.-L. “Optimal traffic organization in ants under crowded conditions”. *Nature* 428.6978 (2004), pp. 70–73.

[73] Deneubourg, J.-L., Goss, S., Franks, N., and Pasteels, J. “The blind leading the blind: modeling chemically mediated army ant raid patterns”. *Journal of insect behavior* 2.5 (1989), pp. 719–725.

[74] Cherix, D., Werner, P., and Catzeflis, F. “Spatial organisation of a polycalic system in Formica (Coptoformica) exsecta Nyl.(Hymenoptera: Formicidae).” *Mitteilungen der Schweizerischen Entomologischen Gesellschaft* 53.2/3 (1980), pp. 163–172.

[75] Deneubourg, J. L., Aron, S., Goss, S., Pasteels, J., and Duerinck, G. “Random behaviour, amplification processes and number of participants: how they contribute to the foraging properties of ants”. *Physica D: Nonlinear Phenomena* 22.1 (1986), pp. 176–186.

[76] Franks, N. R. “Army ants: a collective intelligence”. *American Scientist* 77 (1989), pp. 138–145.

[77] Reid, C. R., Lutz, M. J., Powell, S., Kao, A. B., Couzin, I. D., and Garnier, S. “Army ants dynamically adjust living bridges in response to a cost–benefit trade-off”. *Proceedings of the National Academy of Sciences* 112.49 (2015), pp. 15113–15118.

[78] Jackson, D., Holcombe, M., and Ratnieks, F. “Coupled computational simulation and empirical research into the foraging system of Pharaoh’s ant (Monomorium pharaonis)”. *Biosystems* 76.1 (2004), pp. 101–112.

[79] Jackson, D. E., Martin, S. J., Holcombe, M., and Ratnieks, F. L. “Longevity and detection of persistent foraging trails in Pharaoh’s ants, Monomorium pharaonis (L.)” *Animal Behaviour* 71.2 (2006), pp. 351 –359.

[80] Robinson, E. J., Jackson, D. E., Holcombe, M., and Ratnieks, F. L. “Insect communication:‘no entry’signal in ant foraging”. *Nature* 438.7067 (2005), pp. 442–442.

[81] Robinson, E. J., Green, K., Jenner, E., Holcombe, M, and Ratnieks, F. L. “Decay rates of attractive and repellent pheromones in an ant foraging trail network”. *Insectes sociaux* 55.3 (2008), pp. 246–251.

[82] Robinson, E. J., Ratnieks, F. L., and Holcombe, M. “An agent-based model to investigate the roles of attractive and repellent pheromones in ant decision making during foraging”. *Journal of Theoretical Biology* 255.2 (2008), pp. 250–258.

[83] Dorigo, M. and Blum, C. “Ant colony optimization theory: A survey”. *Theoretical Computer Science* 344.2-3 (2005), pp. 243–278.

[84] Gambardella, L. M., Montemanni, R., and Weyland, D. “Coupling ant colony systems with strong local searches”. *European Journal of Operational Research* 220.3 (2012), pp. 831–843.

[85] Wiesemann, W. and Stützle, T. “Iterated ants: An experimental study for the quadratic assignment problem”. *International Workshop on Ant Colony Optimization and Swarm Intelligence*. Springer. 2006, pp. 179–190.

[86] Fraigniaud, P., Ilcinkas, D., Peer, G., Pelc, A., and Peleg, D. “Graph exploration by a finite automaton”. *Theoretical Computer Science* 345.2 (2005), pp. 331–344.

[87] Hanusse, N., Kavvadias, D., Kranakis, E., and Krizanc, D. “Memoryless search algorithms in a network with faulty advice”. *Theoretical Computer Science* 402.2 (2008), pp. 190–198.

[88] Wagner, I. A., Lindenbaum, M., and Bruckstein, A. M. “Efficiently searching a graph by a smell-oriented vertex process”. *Annals of Mathematics and Artificial Intelligence* 24.1-4 (1998), pp. 211–223.

[89] Feinerman, O., Korman, A., Lotker, Z., and Sereni, J.-S. “Collaborative Search on the Plane Without Communication”. *Proceedings of the 2012 ACM Symposium on Principles of Distributed Computing*. PODC ’12. Madeira, Portugal: ACM, 2012, pp. 77–86.

[90] Emek, Y., Langner, T., Stolz, D., Uitto, J., and Wattenhofer, R. “Towards More Realistic ANTS”. *2nd Workshop on Biological Distributed Algorithms (BDA)*. 2014.

[91] Lenzen, C. and Radeva, T. “The power of pheromones in ant foraging”. *1st Workshop on Biological Distributed Algorithms (BDA)*. 2013.

[92] Kleinberg, J. and Tardos, E. *Algorithm Design*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2005.

[93] Chandy, K. M. and Misra, J. “Distributed computation on graphs: Shortest path algorithms”. *Communications of the ACM* 25.11 (1982), pp. 833–837.

[94] Humblet, P. A. “Another adaptive distributed shortest path algorithm”. *IEEE transactions on communications* 39.6 (1991), pp. 995–1003.

[95] Perlman, R. “An algorithm for distributed computation of a spanningtree in an extended LAN”. *ACM SIGCOMM Computer Communication Review*. Vol. 15. 4. ACM. 1985, pp. 44–53.

[96] Garay, J. A., Kutten, S., and Peleg, D. “A sublinear time distributed algorithm for minimum-weight spanning trees”. *SIAM Journal on Computing* 27.1 (1998), pp. 302–316.

[97] Suomela, J. “Survey of local algorithms”. *ACM Computing Surveys (CSUR)* 45.2 (2013), p. 24.

[98] Merkl, F. and Rolles, S. “Linearly edge-reinforced random walks”. *Institute of Mathematical Statistics Lecture Notes - Monograph Series*. Institute of Mathematical Statistics, 2006, pp. 66–77.

[99] Diaconis, P. and Freedman, D. “de Finetti’s theorem for Markov chains”. *The Annals of Probability* (1980), pp. 115–130.

[100] Davis, B. “Reinforced random walk”. *Probability Theory and Related Fields* 84.2 (1990), pp. 203–229.

[101] Stevens, A. and Othmer, H. G. “Aggregation, blowup, and collapse: the ABC’s of taxis in reinforced random walks”. *SIAM Journal on Applied Mathematics* 57.4 (1997), pp. 1044–1081.

[102] Aron, S, Pasteels, J. M., and Deneubourg, J. L. “Trail-laying behaviour during exploratory recruitment in the argentine ant, Iridomyrmex humilis (Mayr)”. *Biology of Behaviour* 14 (1989), pp. 207–217.

[103] Pinter-Wollman, N., Wollman, R., Guetz, A., Holmes, S., and Gordon, D. M. “The effect of individual variation on the structure and function of interaction networks in harvester ants”. *J R Soc Interface* 8.64 (2011), pp. 1562–1573.

[104] Mersch, D. P., Crespi, A., and Keller, L. “Tracking individuals shows spatial fidelity is a key regulator of ant social organization”. *Science* 340.6136 (2013), pp. 1090–1093.

[105] Jeanson, R., Ratnieks, F. L., and Deneubourg, J.-L. “Pheromone trail decay rates on different substrates in the Pharaoh’s ant, Monomorium pharaonis”. *Physiological Entomology* 28.3 (2003), pp. 192–198.

[106] Simon, T and Hefetz, A. “Trail-following responses ofTapinoma simrothi (Formicidae: Dolichoderinae) to pygidial gland extracts”. *Insectes Sociaux* 38.1 (1991), pp. 17–25.

[107] Perna, A., Granovskiy, B., Garnier, S., Nicolis, S. C., Labédan, M., Theraulaz, G., Fourcassié, V., and Sumpter, D. J. “Individual rules for trail pattern formation in Argentine ants (Linepithema humile)”. *PLoS computational biology* 8.7 (2012), e1002592.

[108] Deneubourg, J.-L., Pasteels, J. M., and Verhaeghe, J.-C. “Probabilistic behaviour in ants: a strategy of errors?” *Journal of Theoretical Biology* 105.2 (1983), pp. 259–271.

[109] Fonio, E., Heyman, Y., Boczkowski, L., Gelblum, A., Kosowski, A., Korman, A., and Feinerman, O. “A locally-blazed ant trail achieves efficient collective navigation despite limited information”. *eLife* 5 (2016), e20185.

[110] Sumpter, D. J. and Beekman, M. “From nonlinearity to optimality: pheromone trail foraging by ants”. *Animal behaviour* 66.2 (2003), pp. 273–280.

[111] Deneubourg, J.-L., Aron, S., Goss, S., and Pasteels, J. M. “The self-organizing exploratory pattern of the argentine ant”. *Journal of insect behavior* 3.2 (1990), pp. 159–168.

[112] Kunegis, J. “KONECT – The Koblenz Network Collection”. *Proc. Int. Conf. on World Wide Web Companion*. 2013, pp. 1343–1350.

[113] Lund, K., Manzo, A. J., Dabby, N., Michelotti, N., Johnson-Buck, A., Nangreave, J., Taylor, S., Pei, R., Stojanovic, M. N., Walter, N. G., Winfree, E., and Yan, H. “Molecular robots guided by prescriptive landscapes”. *Nature* 465.7295 (2010), pp. 206–210.

[114] Werfel, J., Petersen, K., and Nagpal, R. “Designing collective behavior in a termite-inspired robot construction team”. *Science* 343.6172 (2014), pp. 754–758.

[115] Hecker, J. P. and Moses, M. E. “Beyond pheromones: evolving error-tolerant, flexible, and scalable ant-inspired robot swarms”. *Swarm Intelligence* 9.1 (2015), pp. 43–70.

[116] Watts, D. J. and Strogatz, S. H. “Collective dynamics of ‘small-world’networks”. *nature* 393.6684 (1998), p. 440.

[117] Korn, C. and Augustin, H. G. “Mechanisms of Vessel Pruning and Regression”. *Dev. Cell* 34.1 (2015), pp. 5–17.

[118] Czaczkes, T. J., Grüter, C., and Ratnieks, F. L. “Trail pheromones: an integrative view of their role in social insect colony organization”. *Annual review of entomology* 60 (2015), pp. 581–599.

[119] Czaczkes, T. J., Grüter, C., and Ratnieks, F. L. “Negative feedback in ants: crowding results in less trail pheromone deposition”. *Journal of the Royal Society Interface* 10.81 (2013), p. 20121009.

[120] Chouikhi, S., El Korbi, I., Ghamri-Doudane, Y., and Saidane, L. A. “A survey on fault tolerance in small and large scale wireless sensor networks”. *Computer Communications* 69 (2015), pp. 22–37.

[121] Chandrasekhar, A. and Navlakha, S. “Neural arbors are Pareto optimal”. *Proc. Biol. Sci.* 286.1902 (2019), p. 20182727.

[122] Conn, A., Pedmale, U. V., Chory, J., and Navlakha, S. “High-resolution laser scanning reveals plant architectures that reflect universal network design principles”. *Cell systems* 5.1 (2017), pp. 53–62.

[123] Fei, Z., Li, B., Yang, S., Xing, C., Chen, H., and Hanzo, L. “A survey of multi-objective optimization in wireless sensor networks: Metrics, algorithms, and open problems”. *IEEE Communications Surveys & Tutorials* 19.1 (2016), pp. 550–586.

[124] Gong, H., Fu, L., Fu, X., Zhao, L., Wang, K., and Wang, X. “Distributed multicast tree construction in wireless sensor networks”. *IEEE Transactions on Information Theory* 63.1 (2016), pp. 280–296.

[125] Levin, D. “The environment constrains successful search strategies in natural distributed systems” (2016).

[126] Wiles, T. J., Jemielita, M., Baker, R. P., Schlomann, B. H., Logan, S. L., Ganz, J., Melancon, E., Eisen, J. S., Guillemin, K., and Parthasarathy, R. “Host gut motility promotes competitive exclusion within a model intestinal microbiota”. *PLoS biology* 14.7 (2016), e1002517.

[127] Hein, A. M., Brumley, D. R., Carrara, F., Stocker, R., and Levin, S. A. “Physical limits on bacterial navigation in dynamic environments”. *J R Soc Interface* 13.114 (2016), p. 20150844.

[128] Couzin, I. D., Krause, J., Franks, N. R., and Levin, S. A. “Effective leadership and decision-making in animal groups on the move”. *Nature* 433.7025 (2005), p. 513.

[129] Gordon, D. M. “The ecology of collective behavior”. *PLoS Biol* 12.3 (2014), e1001805.

[130] Perna, A. and Latty, T. “Animal transportation networks”. *Journal of The Royal Society Interface* 11.100 (2014), p. 20140334.

[131] Di Caro, G. and Dorigo, M. “AntNet: Distributed stigmergetic control for communications networks”. *Journal of Artificial Intelligence Research* 9 (1998), pp. 317–365.

[132] Couzin, I. D. and Franks, N. R. “Self-organized lane formation and optimized traffic flow in army ants”. *Proceedings of the Royal Society of London. Series B: Biological Sciences* 270.1511 (2003), pp. 139–146.

[133] Cabanes, G., Wilgenburg, E. van, Beekman, M., and Latty, T. “Ants build transportation networks that optimize cost and efficiency at the expense of robustness”. *Behavioral Ecology* 26.1 (2014), pp. 223–231.

[134] Bottinelli, A., Wilgenburg, E van, Sumpter, D. J., and Latty, T. “Local cost minimization in ant transport networks: from small-scale data to large-scale trade-offs”. *Journal of the Royal Society Interface* 12.112 (2015), p. 20150780.

[135] Bouchebti, S., Travaglini, R. V., Forti, L. C., and Fourcassié, V. “Dynamics of physical trail construction and of trail usage in the leaf-cutting ant Atta laevigata”. *Ethology Ecology & Evolution* 31.2 (2019), pp. 105–120.

[136] Countryman, S. M., Stumpe, M. C., Crow, S. P., Adler, F. R., Greene, M. J., Vonshak, M., and Gordon, D. M. “Collective search by ants in microgravity”. *Frontiers in Ecology and Evolution* 3 (2015), p. 25.

[137] Garnier, S., Combe, M., Jost, C., and Theraulaz, G. “Do ants need to estimate the geometrical properties of trail bifurcations to find an efficient route? A swarm robotics test bed”. *PLoS computational biology* 9.3 (2013), e1002903.

[138] Cook, Z., Franks, D. W., and Robinson, E. J. “Efficiency and robustness of ant colony transportation networks”. *Behavioral ecology and sociobiology* 68.3 (2014), pp. 509–517.

[139] Buhl, J., Hicks, K., Miller, E. R., Persey, S., Alinvi, O., and Sumpter, D. J. “Shape and efficiency of wood ant foraging networks”. *Behavioral Ecology and Sociobiology* 63.3 (2009), pp. 451–460.

[140] Prömel, H. J. and Steger, A. *The Steiner tree problem: a tour through graphs, algorithms, and complexity*. Springer Science & Business Media, 2012.

[141] Powell, S., Costa, A. N., Lopes, C. T., and Vasconcelos, H. L. “Canopy connectivity and the availability of diverse nesting resources affect species coexistence in arboreal ants”. *Journal of Animal Ecology* 80.2 (2011), pp. 352–360.

[142] Dorigo, M. and Stützle, T. *Ant Colony Optimization*. Cambridge, MA: MIT Press, 2004.

[143] Chandrasekhar, A., Gordon, D. M., and Navlakha, S. “A distributed algorithm to maintain and repair the trail networks of arboreal ants”. *Sci Rep* 8.1 (2018), p. 9297.

[144] Byrka, J., Lewandowski, M., and Moldenhauer, C. “Approximation algorithms for node-weighted prize-collecting Steiner tree problems on planar graphs”. *arXiv preprint arXiv:1601.02481* (2016).

[145] Bateni, M. H., Hajiaghayi, M. T., and Liaghat, V. “Improved Approximation Algorithms for (Budgeted) Node-weighted Steiner Problems”. *SIAM Journal on Computing* 47.4 (2018), pp. 1275–1293.

[146] Olson, M. E., Aguirre-Hernández, R., and Rosell, J. A. “Universal foliage-stem scaling across environments and species in dicot trees: plasticity, biomechanics and Corner’s Rules”. *Ecology Letters* 12.3 (2009), pp. 210–219.

[147] Emek, Y., Langner, T., Stolz, D., Uitto, J., and Wattenhofer, R. “How many ants does it take to find the food?” *Theoretical Computer Science* 608 (2015), pp. 255–267.

[148] Feinerman, O. and Korman, A. “The ANTS problem”. *Distributed Computing* 30.3 (2017), pp. 149–168.

[149] Stickland, T., Britton, N. F., and Franks, N. R. “Complex trails and simple algorithms in ant foraging”. *Proceedings of the Royal Society of London. Series B: Biological Sciences* 260.1357 (1995), pp. 53–58.

[150] Britton, N., Stickland, T., and Franks, N. “Analysis of ant foraging algorithms”. *Journal of Biological Systems* 6.04 (1998), pp. 315–336.

[151] Monmarché, N., Venturini, G., and Slimane, M. “On how Pachycondyla apicalis ants suggest a new search algorithm”. *Future generation computer systems* 16.8 (2000), pp. 937–946.

[152] Yanoviak, S. and Kaspari, M. “Community structure and the habitat templet: ants in the tropical forest canopy and litter”. *Oikos* 89.2 (2000), pp. 259–266.

[153] Vandermeer, J., Perfecto, I., and Philpott, S. M. “Clusters of ant colonies and robust criticality in a tropical agroecosystem”. *Nature* 451.7177 (2008), p. 457.

[154] Philpott, S. M., Perfecto, I., and Vandermeer, J. “Behavioral diversity of predatory arboreal ants in coffee agroecosystems”. *Environ. Entomol.* 37.1 (2008), pp. 181–191.

[155] Jandt, J. and Gordon, D. “The behavioral ecology of variation in social insects”. *Current opinion in insect science* 15 (2016), pp. 40–44.

[156] Werfel, J., Petersen, K., and Nagpal, R. “Designing collective behavior in a termite-inspired robot construction team”. *Science* 343.6172 (2014), pp. 754–758.

[157] Rees, W. M. van, Vouga, E., and Mahadevan, L. “Growth patterns for shape-shifting elastic bilayers”. *Proceedings of the National Academy of Sciences* 114.44 (2017), pp. 11597–11602.

[158] Qi, L., Griego, A. D., Fricke, G. M., and Moses, M. E. “Comparing Physical and Simulated Performance of a Deterministic and a Bio-inspired Stochastic Foraging Strategy for Robot Swarms”. *Proceedings of the International Conference on Robotics and Automation (ICRA). IEEE*. 2019.

[159] Bajaj, R. and Agrawal, D. P. “Improving scheduling of tasks in a heterogeneous environment”. *IEEE Transactions on Parallel and Distributed Systems* 15.2 (2004), pp. 107–118.

[160] Munguia, L.-M., Bader, D. A., and Ayguade, E. “Task-based parallel breadth-first search in heterogeneous environments”. *2012 19th International Conference on High Performance Computing*. IEEE. 2012, pp. 1–10.

[161] Pavone, M., Frazzoli, E., and Bullo, F. “Adaptive and distributed algorithms for vehicle routing in a stochastic and dynamic environment”. *IEEE Transactions on Automatic Control* 56.6 (2010), pp. 1259–1274.

[162] Duan, X., George, M., Patel, R., and Bullo, F. “Robotic Surveillance Based on the Meeting Time of Random Walks”. *arXiv preprint arXiv:1912.02693* (2019).

[163] Scheirer, C. J., Ray, W. S., and Hare, N. “The analysis of ranked data derived from completely randomized factorial designs”. *Biometrics* (1976), pp. 429–434.

[164] Ogle, D. H., Wheeler, P., and Dinno, A. *FSA: Fisheries Stock Analysis*. R package version 0.8.22. 2018.

[165] Li, K., Torres, C. E., Thomas, K., Rossi, L. F., and Shen, C.-C. “Slime mold inspired routing protocols for wireless sensor networks”. *Swarm Intelligence* 5.3-4 (2011), pp. 183–223.

[166] Sribala, S and Virudhunagar, T. “Energy efficient routing in wireless sensor networks using modified bacterial foraging algorithm”. *IJREAT International Journal of Research in engineering and Advanced Technology* 1.1 (2013).

[167] Laughlin, S. B. and Sejnowski, T. J. “Communication in neuronal networks”. *Science* 301.5641 (2003), pp. 1870–1874.

[168] Wen, Q. and Chklovskii, D. B. “A cost–benefit analysis of neuronal morphology”. *Journal of neurophysiology* 99.5 (2008), pp. 2320–2328.

[169] Rivera-Alba, M., Peng, H., Polavieja, G. G. de, and Chklovskii, D. B. “Wiring economy can account for cell body placement across species and brain areas”. *Current Biology* 24.3 (2014), R109–R110.

[170] Chen, B. L., Hall, D. H., and Chklovskii, D. B. “Wiring optimization can relate neuronal structure and function”. *Proceedings of the National Academy of Sciences of the United States of America* 103.12 (2006), pp. 4723–4728.

[171] Rivera-Alba, M., Vitaladevuni, S. N., Mishchenko, Y., Lu, Z., Takemura, S.-y., Scheffer, L., Meinertzhagen, I. A., Chklovskii, D. B., and Polavieja, G. G. de. “Wiring economy and volume exclusion determine neuronal placement in the Drosophila brain”. *Current Biology* 21.23 (2011), pp. 2000–2005.

[172] Wang, I. E. and Clandinin, T. R. “The influence of wiring economy on nervous system evolution”. *Current Biology* 26.20 (2016), R1101–R1108.

[173] Cuntz, H., Forstner, F., Borst, A., and Häusser, M. “One rule to grow them all: a general theory of neuronal branching and its practical application”. *PLoS Comput Biol* 6.8 (2010), e1000877.

[174] Budd, J. M., Kovács, K., Ferecskó, A. S., Buzás, P., Eysel, U. T., and Kisvárday, Z. F. “Neocortical axon arbors trade-off material and conduction delay conservation”. *PLoS Comput Biol* 6.3 (2010), e1000711.

[175] Chklovskii, D. B. “Synaptic connectivity and neuronal morphology: two sides of the same coin”. *Neuron* 43.5 (2004), pp. 609–617.

[176] Ramon, Y and Cajal, S. *Textura del Sistema Nervioso del Hombre y de los Vertebrados*. Vol. 2. Madrid Nicolas Moya, 1904.

[177] Vetter, P., Roth, A., and Häusser, M. “Propagation of action potentials in dendrites depends on dendritic morphology”. *Journal of neurophysiology* 85.2 (2001), pp. 926–937.

[178] Gasparini, S. and Migliore, M. “Action Potential Backpropagation”. *Encyclopedia of Computational Neuroscience* (2015), pp. 133–137.

[179] Bekkers, J. M. and Häusser, M. “Targeted dendrotomy reveals active and passive contributions of the dendritic tree to synaptic integration and neuronal output”. *Proceedings of the National Academy of Sciences* 104.27 (2007), pp. 11447–11452.

[180] Ahn, Y.-Y., Jeong, H., and Kim, B. J. “Wiring cost in the organization of a biological neuronal network”. *Physica A: Statistical Mechanics and its Applications* 367 (2006), pp. 531–537.

[181] Kim, Y., Sinclair, R., Chindapol, N., Kaandorp, J. A., and De Schutter, E. “Geometric theory predicts bifurcations in minimal wiring cost trees in biology are flat”. *PLoS computational biology* 8.4 (2012), e1002474.

[182] Teeter, C. M. and Stevens, C. F. “A general principle of neural arbor branch density”. *Current Biology* 21.24 (2011), pp. 2105–2108.

[183] Sugimura, K., Shimono, K., Uemura, T., and Mochizuki, A. “Self-organizing mechanism for development of space-filling neuronal dendrites”. *PLoS computational biology* 3.11 (2007), e212.

[184] Scott, E. K. and Luo, L. “How do dendrites take their shape?” *Nature neuroscience* 4.4 (2001), p. 359.

[185] Panico, J. and Sterling, P. “Retinal neurons and vessels are not fractal but space-filling”. *Journal of Comparative Neurology* 361.3 (1995), pp. 479–490.

[186] Aćimović, J., Mäki-Marttunen, T., and Linne, M.-L. “The effects of neuron morphology on graph theoretic measures of network connectivity: the analysis of a two-level statistical model”. *Frontiers in neuroanatomy* 9 (2015).

[187] Puppo, F., George, V., and Silva, G. A. “An Optimized Structure-Function Design Principle Underlies Efficient Signaling Dynamics in Neurons”. *Scientific reports* 8.1 (2018), p. 10460.

[188] Hodgkin, A. L., Huxley, A. F., and Katz, B. “Measurement of current-voltage relations in the membrane of the giant axon of Loligo”. *The Journal of physiology* 116.4 (1952), pp. 424–448.

[189] Rall, W. “Theory of physiological properties of dendrites”. *Annals of the New York Academy of Sciences* 96.1 (1962), pp. 1071–1092.

[190] Rall, W. “Electrophysiology of a dendritic neuron model”. *Biophysical journal* 2.2 (1962), pp. 145–167.

[191] Chklovskii, D. B. and Stepanyants, A. “Power-law for axon diameters at branch point”. *BMC neuroscience* 4.1 (2003), p. 18.

[192] Rall, W, Burke, R., Holmes, W., Jack, J., Redman, S., and Segev, I. “Matching dendritic neuron models to experimental data”. *Physiological Reviews* 72.suppl 4 (1992), S159–S186.

[193] West, G. B., Brown, J. H., and Enquist, B. J. “The fourth dimension of life: fractal geometry and allometric scaling of organisms”. *Science* 284.5420 (1999), pp. 1677–1679.

[194] West, G. B., Brown, J. H., and Enquist, B. J. “A general model for the structure and allometry of plant vascular systems”. *Nature* 400.6745 (1999), pp. 664–667.

[195] Banavar, J. R., Damuth, J., Maritan, A., and Rinaldo, A. “Supply-demand balance and metabolic scaling”. *Proc. Natl. Acad. Sci. U.S.A.* 99.16 (2002), pp. 10506–10509.

[196] Arora, S. “Polynomial time approximation schemes for Euclidean TSP and other geometric problems”. *Foundations of Computer Science, 1996. Proceedings., 37th Annual Symposium on*. IEEE. 1996, pp. 2–11.

[197] Karpinski, M. and Zelikovsky, A. “New approximation algorithms for the Steiner tree problems”. *Journal of Combinatorial Optimization* 1.1 (1997), pp. 47–65.

[198] Agrawal, A., Klein, P., and Ravi, R. “When trees collide: An approximation algorithm for the generalized Steiner problem on networks”. *SIAM Journal on Computing* 24.3 (1995), pp. 440–456.

[199] Khuller, S., Raghavachari, B., and Young, N. “Balancing minimum spanning trees and shortest-path trees”. *Algorithmica* 14.4 (1995), pp. 305–321.

[200] Alpert, C. J., Hu, T. C., Huang, J., Kahng, A. B., and Karger, D. “Prim-Dijkstra tradeoffs for improved performance-driven routing tree design”. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 14.7 (1995), pp. 890–896.

[201] Chen, G., Chen, S., Guo, W., and Chen, H. “The multi-criteria minimum spanning tree problem based genetic algorithm”. *Information Sciences* 177.22 (2007), pp. 5050–5063.

[202] Sourd, F. and Spanjaard, O. “A multiobjective branch-and-bound framework: Application to the biobjective spanning tree problem”. *INFORMS Journal on Computing* 20.3 (2008), pp. 472–484.

[203] Rees, C. L., Moradi, K., and Ascoli, G. A. “Weighing the Evidence in Peters’ Rule: Does Neuronal Morphology Predict Connectivity?” *Trends Neurosci.* 40.2 (2017), pp. 63–71.

[204] Kasthuri, N., Hayworth, K. J., Berger, D. R., Schalek, R. L., Conchello, J. A., Knowles-Barley, S., Lee, D., Vazquez-Reina, A., Kaynig, V., Jones, T. R., Roberts, M., Morgan, J. L., Tapia, J. C., Seung, H. S., Roncal, W. G., Vogelstein, J. T., Burns, R., Sussman, D. L., Priebe, C. E., Pfister, H., and Lichtman, J. W. “Saturated Reconstruction of a Volume of Neocortex”. *Cell* 162.3 (2015), pp. 648–661.

[205] Garey, M. R., Graham, R. L., and Johnson, D. S. “The complexity of computing Steiner minimal trees”. *SIAM journal on applied mathematics* 32.4 (1977), pp. 835–859.

[206] Shoval, O., Sheftel, H., Shinar, G., Hart, Y., Ramote, O., Mayo, A., Dekel, E., Kavanagh, K., and Alon, U. “Evolutionary trade-offs, Pareto optimality, and the geometry of phenotype space”. *Science* (2012), p. 1217405.

[207] Da Cunha, N. and Polak, E. “Constrained minimization under vector-valued criteria in finite dimensional spaces”. *Journal of Mathematical Analysis and Applications* 19.1 (1967), pp. 103–124.

[208] Greenberg, H. J. “Greedy algorithms for minimum spanning tree”. *University of Colorado at Denver* (1998).

[209] Kou, L, Markowsky, G., and Berman, L. “A fast algorithm for Steiner trees”. *Acta informatica* 15.2 (1981), pp. 141–145.

[210] Gabow, H. N. and Myers, E. W. “Finding all spanning trees of directed and undirected graphs”. *SIAM Journal on Computing* 7.3 (1978), pp. 280–287.

[211] Shepherd, G. M., Raastad, M., and Andersen, P. “General and variable features of varicosity spacing along unmyelinated axons in the hippocampus and cerebellum”. *Proceedings of the National Academy of Sciences* 99.9 (2002), pp. 6340–6345.

[212] O’Brien, J. and Unwin, N. “Organization of spines on the dendrites of Purkinje cells”. *Proceedings of the National Academy of Sciences* 103.5 (2006), pp. 1575–1580.

[213] Kuljis, D. A., Park, E., Telmer, C. A., Lee, J., Ackerman, D. S., Bruchez, M. P., and Barth, A. L. “Fluorescence-based quantitative synapse analysis for cell-type specific connectomics”. *eNeuro* (2019).

[214] Conn, A., Chandrasekhar, A., Rongen, M. v., Leyser, O., Chory, J., and Navlakha, S. “Network trade-offs and homeostasis in Arabidopsis shoot architectures”. *PLoS computational biology* 15.9 (2019), e1007325.

[215] Sporns, O. *Networks of the Brain*. MIT press, 2010.

[216] Wilson, D. B. “Generating random spanning trees more quickly than the cover time”. *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*. ACM. 1996, pp. 296–303.

[217] Barabási, A.-L. and Albert, R. “Emergence of scaling in random networks”. *science* 286.5439 (1999), pp. 509–512.

[218] Ascoli, G. A., Donohue, D. E., and Halavi, M. “NeuroMorpho. Org: a central resource for neuronal morphologies”. *Journal of Neuroscience* 27.35 (2007), pp. 9247–9251.

[219] Halavi, M., Polavaram, S., Donohue, D. E., Hamilton, G., Hoyt, J., Smith, K. P., and Ascoli, G. A. “NeuroMorpho. Org implementation of digital neuroscience: dense coverage and integration with the NIF”. *Neuroinformatics* 6.3 (2008), p. 241.