The goal of this dissertation is to generate organizational structures that increase the overall performance of a multiagent coalition, subject to the system's complex coordination requirements and maintenance of a certain operating point. To this end, a generalized framework capable of producing *distributed* approximation algorithms based on the new concept of *multidirectional graph search* is proposed and applied to a family of connectivity problems. It is shown that a wide variety of seemingly unrelated multiagent organization problems live within this family. Sufficient conditions are identified in which the approach is guaranteed to discover a solution that is within a constant factor of the cost of the optimal solution. The procedure is guaranteed to require no more than linear—and in some well defined cases logarithmic—communication rounds. A number of examples are given as to how the framework can be applied to create, maintain, and optimize multiagent organizations in the context of real world problems. Finally, algorithmic extensions are introduced that allow for the framework to handle problems in which the agent topology and/or coordination constraints are dynamic, without significant consequences to the general runtime, memory, and quality guarantees.

Committee:
William C. Regli (Advisor)
Professor, Department of Computer Science, Drexel University
Director, AJ Drexel Institute for Applied Communications and Information Networking
Ali Shokoufandeh (Advisor)
Professor and Associate Department Head, Department of Computer Science, Drexel University
Rachel Greenstadt (Chair)
Assistant Professor, Department of Computer Science, Drexel University
Jeremy Johnson
Professor and Department Head, Department of Computer Science, Drexel University
Sven Koenig
Associate Professor, Department of Computer Science, University of Southern California
Program Director, National Science Foundation
Joseph P. Macker
Senior Communications and Network Research Scientist, Naval Research Laboratory

Evan A. Sultanik

Ph.D. Dissertation

September, 2010

Ph.D. Dissertation

Automatic Construction, Maintenance, and Optimization of Dynamic Agent Organizations

Evan A. Sultanik

A Dissertation
Submitted to the Faculty of Drexel University
in Partial Fulfillment of the Requirements
for the Degree of
Doctor of Philosophy

Automatic Construction, Maintenance, and Optimization of Dynamic Agent Organizations

Evan Andrew Sultanik

Dedications

This dissertation is dedicated to. . .

- my parents, Judi and Jeff, who furnished me with the requisite curiosity, patience, and freedoms; and

- my wife Nadya—my most fervent supporter through this long process—who postponed her dreams in deference to mine.

Acknowledgments

It is with great pleasure that I thank all of the people that helped me get to this point. I have often fantasied *what* I would say to them, but now in writing I struggle with *how*. This dissertation would not have been possible had my advisor, William C. Regli, lacked the patience and foresight in guiding me into the world of academic research. I am equally indebted to my co-advisor, Ali Shokoufandeh, who taught me technical diligence and tempered my writing with humility. I must also thank my late mentor, Pragnesh Jay Modi, who—during his brief tenure in my education—so profoundly affected the course of my research by introducing me to multiagent optimization. I am also grateful to Moshe Kam who in many ways has also acted as an unofficial co-advisor.

The content of this dissertation would not have been what it is had neither Dr. Regli provided me the freedom to identify and frame the problems of my choice, nor Dr. Shokoufandeh donated his time in my pursuit. Some of my pedantic freedom was also attributable to several fellowships I received throughout my matriculation, for which I owe sincere thanks to the George Hill, Jr. Endowment and the Koerner family. The remainder of my support was in the form of grants from various United States government entities, all of which were ultimately made possible by annual financial support from Viewers Like You.

I would also like to extend my thanks to everyone who donated time in proof-reading and commenting on this document, notably Robert N. Lass and my wife Nadya Sultanik (*née*—and known professionally as—Nadya Belov). Robert deserves additional mention, as he was my collaborator on much of my work during my studies, including the Mobed algorithm (*cf.* §6.2). Most notably, I owe my deepest gratitude to my committee members: Rachel Greenstadt, Jeremy Johnson, Sven Koenig, and Joseph Macker, for their time and continuous input on my ideas.

Most of all I must again thank Nadya for taking care of everything else going on in my life during the void that was the completion of this dissertation.

For additional credits see page 151.

Contents

List of Tables

List of Figures

Abstract

Automatic Construction, Maintenance, and Optimization of Dynamic Agent
Organizations

Evan Andrew Sultanik
Advisors: William C. Regli, Ph.D. and Ali Shokoufandeh, Ph.D.

The goal of this dissertation is to generate organizational structures that
increase the overall performance of a multiagent coalition, subject to the
system's complex coordination requirements and maintenance of a certain
operating point. To this end, a generalized framework capable of producing
distributed approximation algorithms based on the new concept of *multidi-
rectional graph search* is proposed and applied to a family of connectivity
problems. It is shown that a wide variety of seemingly unrelated multia-
gent organization problems live within this family. Sufficient conditions are
identified in which the approach is guaranteed to discover a solution that
is within a constant factor of the cost of the optimal solution. The proce-
dure is guaranteed to require no more than linear—and in some well defined
cases logarithmic—communication rounds. A number of examples are given
as to how the framework can be applied to create, maintain, and optimize
multiagent organizations in the context of real world problems. Finally, al-
gorithmic extensions are introduced that allow for the framework to handle
problems in which the agent topology and/or coordination constraints are
dynamic, without significant consequences to the general runtime, memory,
and quality guarantees.

Chapter 1

Introduction

The goal of this dissertation is to formalize the coordination requirements of complex systems; the vision is that such systems might distributedly self-organize in order to increase the overall performance of the coalition. This phenomenon manifests itself in human societies: Workers often naturally coalesce into a leadership structure, as in a corporation. Efficiency is often achieved through the parallelism inherent in managerial hierarchies. How can such concepts be extended to the domain of intelligent software agents? The problem posed by this dissertation can be stated as follows:

> Given a notion of the way in which a group of agents need to interact, what is the best organizational structure for the system that might expedite the process of coordination while maintaining a certain operating point?

This shall be referred to as the Dynamic Distributed Multiagent Hierarchy Generation (DynDisMHG) problem.

The proliferation of mobile and handheld computers—such as laptops, personal digital assistants, and smart phones—has propelled distributed computing into mainstream society. Over the past decade these technologies have spurred interest in both decentralized multiagent systems and wireless mobile *ad hoc* networks. Such networks, however, present many challenges to information sharing and coordination. Interference, obstacles, and other environmental effects conspire with power- and processing-limited hardware to impose a number of challenging networking characteristics. Messages are routinely lost or delayed, connections may be only sporadically available and frequently lost, and network transfer capacity is nowhere near that available on modern wired networks.

Many distributed optimization problems arise by virtue of these networks' limitations on efficiency and robustness, while others arise merely due to the networks' existence. For example, some problems are naturally distributed,

requiring extra effort to capture and centralize the state of the world such that a traditional centralized technique can be employed. Furthermore, hardware limitations can be so severe that there may not even exist a single node with the resources to compute a global solution. This dissertation addresses problems that arise in such situations.

When deploying a dynamic distributed system, one must always weigh the cost incurred by network communication against any benefits of using a centralized approach. In some cases, the problem may be changing so fast or the communications overhead so expensive that a centralized algorithm is not able to maintain stability due to dynamism and/or the fact that the system state is distributed across the network [1]. It is therefore imperative to emphasize *local* decision making and autonomy over a centralized analogue, insofar as it is possible.

Sensor networks, for example, often communicate over *ad hoc* networks. Such networks suffer from high amounts of bit and frame errors, requiring constant retransmission of data. Moreover, mobile *ad hoc* networks (MANETs) suffer high packet losses and frame error rates, resulting in less than 50% of the theoretical maximum throughput being achieved [2]. Since MANET network topology is inherently in flux, the additional messaging overhead of a centralized control approach can have a significant negative effect on the controllability, stability, and overall robustness of the system.

Not all distributed solutions are satisfactory, however. A reasonable distributed algorithm must run in worst case polynomial time at each agent and require no more than a linear number of communication rounds to find a solution. The latter requirement is necessary since the global state can always be centralized in linear time via naïve flooding.

In distributed systems there is very often a logical or societal structure that is overlaid upon the actual communications network, dictating the patterns in which the agents interact. In the remainder of this dissertation, such structures shall be interchangeably referred to as *agent hierarchies*, *overlay networks*, *interaction graphs*, or *constrained forests*. As an example, note the *interaction graph* and *network topology* in Figure 1.1. The interaction graph is constructed completely by the agents (possibly as a function of the problem they are trying to solve), while the network topology is an artifact of the physical orientation of the agents and is usually not controllable. Isomorphism between the overlay topology and the network topology can be very important; discrepancy between the two can cause unnecessary delays in messaging. Messaging at the application layer will be dictated by the logical topology of the interaction graph, which in Figure 1.1 induces a circuitous route through the network.

This dissertation identifies a family of connectivity problems that are efficiently distributedly approximable with a bounded performance guarantee using the general paradigm of the primal-dual schema. This is a bit surpris-
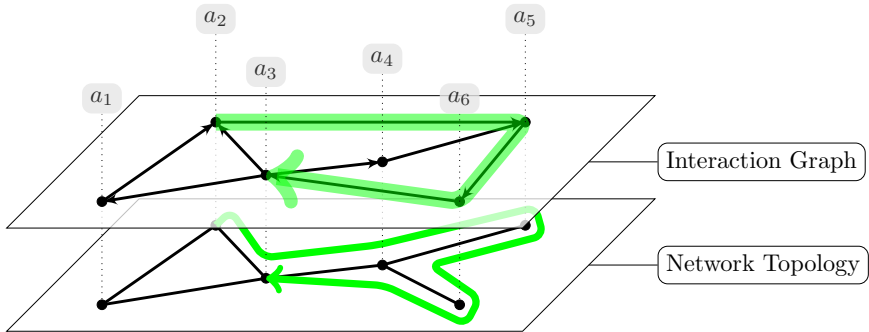
Figure 1.1: The interaction graph of a multiagent system layered on top of the corresponding network topology. A message from a_2 to a_6—necessitated by the application-layer coordination algorithm and social topology—takes a circuitous route through the network.

ing because many combinatorial problems formulated as linear programming optimizations are known to be **P**-Complete (*i.e.*, they are likely inherently sequential). It is shown that a wide variety of seemingly unrelated multiagent organization problems live within this family. Optimally solving many such problems is **NP**-Hard, necessitating bounded approximation, *viz.*, tractable algorithms that find solutions within a constant factor of optimal. It is argued in §1.3 that, for distributed multiagent algorithms, "tractable" should be taken to mean any algorithm whose local runtime is no worse than polynomial in the size of the problem and, more importantly, requires no more than a linear number of messaging rounds. This dissertation proposes a framework that meets these requirements. Our examination of a family of problems illustrates that the framework can be applied to create, maintain, and optimize multiagent organizations in the context of real world problems.

1.1 Exemplary Problems & Scenarios

This section serves to motivate some of the multiagent organization problems for which this dissertation is concerned.

1.1.1 Location Design & Vehicle Routing Problems

The Location Design and Routing problem asks to find a subset of "depot" nodes and a spanning forest of a graph such that every connected component in the forest contains at least one depot [3]. A typical scenario takes the form of creating an optimal distribution network: Given a network of roads, how many distribution centers need be built—and in which locations—such that

(a) A weighted distribution network where a depot can be opened at any vertex.

(b) Optimal solution if the depot opening cost is ∈ [100, 150].

(c) Optimal solution if the depot opening cost is ∈ [150, 224].
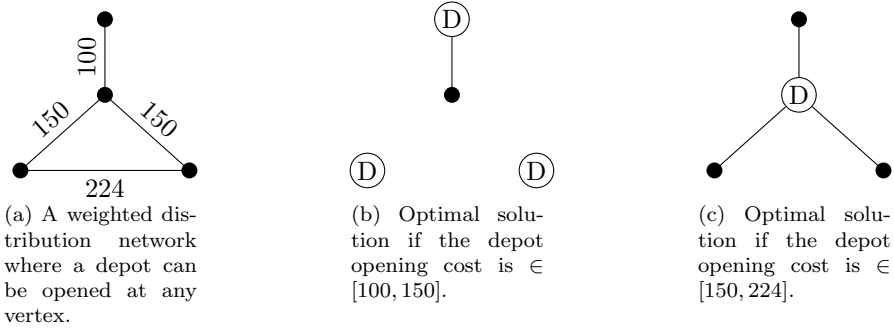
Figure 1.2: An example weighted distribution network, (a), along with the optimal spanning forest and depot assignment for various depot opening costs, (b) & (c).

deliveries can be carried out as quickly as possible? What if the cost of building distribution centers is nonuniform with respect to location? Scenarios are not strictly limited to logistical domains; in computer networking and multiagent systems, for example, the unavailability of a central server/database necessitates data redundancy in the system [4]. Many peer-to-peer systems, such as service oriented computing platforms and distributed hash tables, promote certain nodes as *supernodes*. In this setting, the supernodes are akin to the depots in the location design and routing problem. The problem of selecting which subset of peers should perform a certain role such that they are well dispersed in the network—what is referred to as the *supernode selection problem* [5]—is therefore equivalent to the location design and routing problem. Similar problems also appear in the fields of sensor networks and peer-based grid computing [6]. It is important to consider *distributed* solutions to this problem as it is naturally distributed; extra effort is required to centralize the problem. Furthermore, as is the case in many sensor networks, there may not be a single node with adequate resources to solve the global problem. Computation is therefore subject to the same topology as the network itself. Figure 1.2 gives an example of a weighted distribution network, Figure 1.2a, along with two optimal solutions, Figures 1.2b & 1.2c, depending on the cost of opening a depot. In general, finding a set of depots and a spanning forest of minimal weight is **NP**-HARD [7].

1.1.2 Art Gallery Problems

Art gallery problems generally ask to find the minimum number of guards required to observe the interior of a polygonal area [8]. Over the past thirty years since their proposition, these problems have been thoroughly studied by the computational geometry community. Interest in art gallery problems has seen a recent resurgence given their application to a number of areas of multiagent systems. For example, many robotics, sensor network, wireless networking, and surveillance problems can be mapped to variants of the art gallery problem [9]. Since such problems are naturally distributed, a logical approach is to apply the multiagent paradigm (*i.e.*, each guard is an agent).

As a motivating scenario, consider a wireless sensor network such as the one pictured in Figure 1.3. Since one goal of the network is to maximize survivability, it may be desirable to conserve battery power by having as few sensors active as necessary, especially for sensors with wide overlapping fields of view. The problem is then to find a minimum subset of sensors that need to remain active in order to provide a desirable level of coverage. As another scenario, consider a group of mobile robots each equipped with a wireless access point. The objective of the robots is to maximally cover an area with the wireless network. As the robots are traveling between waypoints, though, it is highly likely that there will be a large amount of overlap in the coverage. Therefore, in order to save power, the robots might want to choose a maximum subset of robots that can lower their transmit power while still retaining coverage. The difficulty in each of these scenarios is for the agents to collectively find the solution without relying on centralization of computation. Centralization is infeasible either due to lack of resources (*i.e.*, no single agent has powerful enough hardware to solve the global problem) or due to lack of time (*i.e.*, centralizing the problem will take at least a linear number of messaging rounds). The decision versions of these problems are equivalent to art gallery problems which are known to be **NP**-Complete [10, 11] and **APX**-Hard [12].

1.1.3 Steiner Network Problems

Steiner network problems generally ask to find a minimum weight set of edges that interconnect subsets of a graph's vertices. Many variants of Steiner network problems are **NP**-Complete, one of which was among Karp's original 21 **NP**-Complete problems [13].

The applications of Steiner networks to distributed systems are manifold. For example, a common approach to *ad hoc* multicast networking is to construct an acyclic overlay network connecting all nodes in a group, such that multicast packets can be broadcast across the overlay network [14, 15].
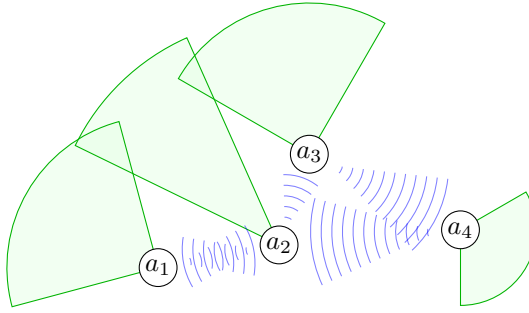
(a) An *ad hoc* sensor network wirelessly coordinating to optimize interior coverage.
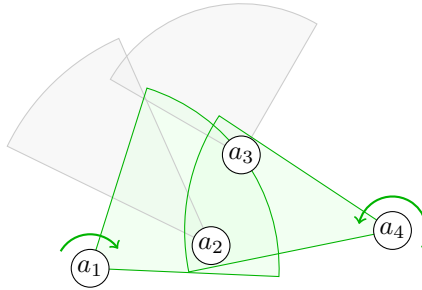
(b) Dynamically-generated orientation.

Figure 1.3: In (a), an *ad hoc* sensor network must distributedly reorient. In (b), agents a_1 and a_4 rotate to guard the interior.

1.1.4 Dynamic Organization Problems

It is useful to impose organizational structure over multiagent coalitions. Hierarchies, for instance, allow for compartmentalization of tasks: If organized correctly, tasks in disjoint subtrees of the hierarchy may be performed in parallel. Intuitively, the shallower the hierarchy the more subordinates per manager, leading to more potential for parallelism. The difficulty lies in determining a minimum depth hierarchy that is isomorphic to the problem being solved. In a business, for example, there is very little sense in assigning an accountant from the billing department as the superior of a marketing associate. Given a notion of the way in which the agents need to interact, the initial problem, then, is to determine the best hierarchy that might expedite the process of coordination. Henceforth we will refer to this problem as the *Dynamic Distributed Multiagent Hierarchy Generation* (DynDisMHG) problem.

Solutions to the DynDisMHG problem currently have direct application in the field of multiagent systems, including distributed problem solving [16], cooperative multiagent systems [17], distributed constraint reasoning (DCR), command and control, mobile *ad hoc* networks (MANETs), sensor nets, and manufacturing. For example, the computation time required by most complete DCR algorithms is determined by the topology of a hierarchical ordering of the agents [18, 19, 20, 21]. The difficulty is that (1) most algorithms assume that an oracle exists to provide an efficient hierarchy; and (2) the few existing solutions to the Multiagent Hierarchy Generation problem are either centralized or do not deal well (or at all) with dynamic addition and removal of agents from the hierarchy.

Let us consider the multiagent variable assignment problem pictured in Figure 1.4. Dotted lines represent connections in the interaction graph, *vi&.* constraints between variables. It is assumed that the variable assignment algorithm to be employed (*e.g.*, a DCR algorithm) only requires communication between agents whose variables are constrained to each other. The network topology is linear; any messages sent between hosts h_1 and h_3 must be relayed through h_2. Observe that the interaction graph is decoupled from (*i.e.*, non-isomorphic to) the network topology. For instance, any messages agent a_2 might have to send a_4 regarding their constrained variables v_5 and v_9 must be routed through host h_2. By reorganizing the interaction graph (*i.e.*, the variable/agent mapping and/or the agent/host mapping), it may be possible to reduce the overhead imposed by network routing, thereby reducing the cost of coordination.

Other potential applications of a solution to DynDisMHG exist in MANETs. Protocols such as Fireworks [22] overlay a communications structure onto a wireless network, which is highly dynamic as the nodes are constantly moving. There is the potential for cross-layer design: If the mobile nodes are executing a multiagent system, a communications structure could be created
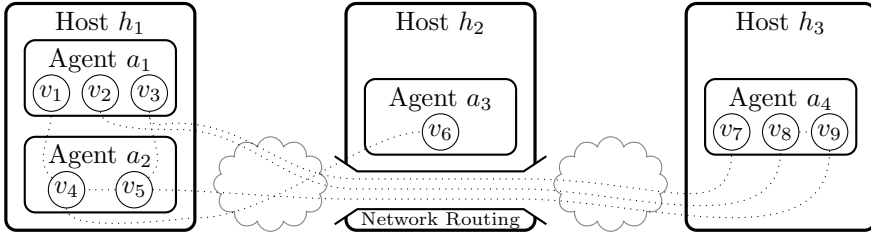
Figure 1.4: A multiagent variable assignment problem mapped atop a peer-to-peer network. The topology is linear: $h_1 \leftrightarrow h_2 \leftrightarrow h_3$. Dotted lines represent connections in the interaction graph, *vi&.* constraints between variables.

to exploit knowledge of both network and application layer properties.

There has been interest in DCR algorithms that are able to solve constantly changing problems [23, 24], including those in which agents can dynamically enter and leave the hierarchy [25]. All existing provably superstabilizing (*i.e.*, "complete") dynamic DCR algorithms, however, make a similar assumption to their static DCR counterparts: that a separate algorithm exists to generate and maintain the dynamically changing agent hierarchy.

Similar to dynamic DCR, there has been much interest in hierarchies of *holonic* multiagent systems (or *holarchies*), with wide ranging applications in distributed problem solving and manufacturing [26]. Some have even claimed that a prerequisite for innovation in multiagent systems is the capability for subsets of agents to dynamically create *ad hoc* hierarchies, called "*adhocracies*" [27]. Empirical evaluations have concluded that agents in a dynamic hierarchy are able to perform distributed problem solving better than agents in a static hierarchy [28]. It is anticipated that solutions to the problem of distributed multiagent hierarchy/holarchy/adhocracy generation will motivate many other applications.

1.1.5 The Pseudotree Creation Problem

Agent hierarchies, often called "variable orderings," are employed in many Distributed Constraint Reasoning (DCR) algorithms, usually as a means to parallelize computation for portions of the constraint graph. Most provably optimal DCR algorithms require a special hierarchy in the form of *pseudotree* [29, 30]. Furthermore, the computational complexity of such algorithms is often a function of the breadth of the tree, since branches in the tree are what allow for parallelism. One metric for the amount of possible parallelism in a tree is induced-width; in general, the problem of finding a minimum-induced-width spanning tree of a graph is **NP**-HARD [31].
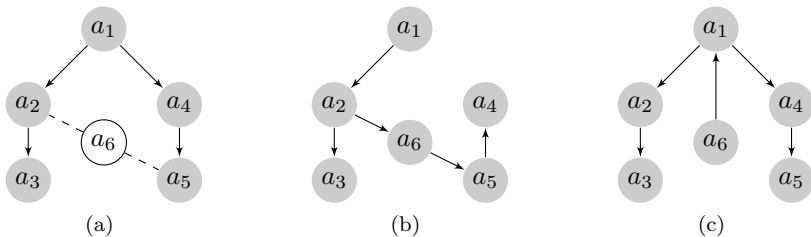
Figure 1.5: a_6, whose interaction graph neighbors are a_2 and a_5, requests to join the existing hierarchy in (a). If DFS is simply re-run the hierarchy in (b) results. Note that the parents of both a_4 and a_5 change. The optimal hierarchy in terms of minimal depth and edits—which could not have been produced by a DFS traversal—is in (c).

A valid hierarchy inherently has the property that each pair of neighboring agents in the interaction graph are either ancestors or descendants of each other in the hierarchy. This ensures that no interaction will necessarily occur between agents in disjoint subtrees. Therefore, interactions in disjoint subtrees may occur in parallel.

It is relatively trivial to prove that a simple depth-first traversal of the interaction graph will produce a valid hierarchy. Distributed algorithms for performing such a DFS traversal are known [32, 33]. Heuristics for guiding the DFS traversal for multiagent problem solving have also been proposed [34]. A general problem with DFS-based approaches, though, is that they will often produce sub-optimal hierarchies (*i.e.*, trees that are unnecessarily deep). For example, the hierarchy in Figure 1.5b might have been generated using a DFS traversal, however, the best-case hierarchy in Figure 1.5c could not have been generated using DFS.

A decentralized algorithm for creating valid pseudotree hierarchies has been proposed [29], however, its efficiency relies upon *a priori* knowledge about the maximum block size of the interaction graph, and it is also unclear how it might be extended to dynamic hierarchies. Some DCR algorithms construct a DFS-based pseudotree as the problem is being solved [32, 18, 35], and yet another has been proposed to handle a relaxed version of pseudotrees [36], however, it is likewise unclear how these algorithms might be extended to handle the intricacies of concurrency imposed by dynamic hierarchies. A number of algorithms based on asynchronous backtracking have been developed that dynamically reorder the agents within the hierarchy as the problem is being solved [28, 37], but this approach has only been explored in terms of static DCR problems and it is unclear how it might be extended to problems in which agents can dynamically be added and removed.

DFS-based algorithms are relatively inexpensive (most require only a lin-

ear number of rounds), so an argument might be made that DFS could simply be re-run every time the tree changes, possibly through the use of a self-stabilizing algorithm. In certain instances, however, such an approach might cause a large disparity between the original hierarchy and the hierarchy resulting after the perturbation, as pictured in Figure 1.5. Continuing the example of a business, the marketing department should not necessarily have to change its managerial structure every time a new accountant is hired in the billing department. An approach with a minimal number of edits to the existing hierarchy is therefore desirable. One way to ensure a constant number of edits is to simply add new agents as a parent of the root of the hierarchy. The problem with this method, however, is that if many new agents are added then the hierarchy will tend toward a chain, which is the worst case topology in terms of parallelism. What is ultimately desirable, then, is an approach that both minimizes edit distance between successive hierarchies and also minimizes tree depth.

1.2   Overview of the Proposed Approach

A thesis of this dissertation is that the aforementioned motivating scenarios, and others like them, can all be reduced to connectivity problems—problems in which each vertex requires to be connected to some subset of the other vertices. Such problems occur when a group of agents need to efficiently communicate with each other subject to a set of constraints on their topology. The idea of reducing such problems to the connectivity domain is not new; it has been espoused by many in the approximation algorithms community. A detailed example of how and why these reductions occur is given below in §2.3 after the introduction of some requisite formalism. A primary contribution of this dissertation, then, is introduction of a novel generalized framework capable of automatically producing *distributed* approximation algorithms that can solve a large family of connectivity problems. In doing so we have a way of constructing and optimizing organizational structures in distributed systems (subject to prescribed constraints on the underlying topology).

The ability to distributedly construct, optimize, and maintain organizational structures and overlay networks with certain desirable properties has direct application in multiagent systems, including the fields of:

- **distributed problem solving** [16]: computation can be parallelized by constructing high degree overlay networks;

- **cooperative multiagent systems** [17]: Pareto-optimal orderings can be found through overlay networks;

- **distributed constraint reasoning** [30]: construction of pseudotrees

> (requisite for most provably optimal distributed constraint reasoning algorithms);

- **command and control** [38]: determining optimal control hierarchies;

- **mobile *ad hoc* networking** [15]: creating Steiner networks for efficient multicast;

- **sensor networks** [39, 40]: power management; and

- **multiagent manufacturing** [26]: distributed task allocation.

Once a solution is found, small perturbations in the problem may occur. For example, a node may either connect to or disconnect from the network. If the change is minor and/or localized, it stands that a new solution might be easily found without resorting to re-solve the problem from scratch. Additional contributions of this dissertation are the discovery of methods that allow for dynamic maintenance of the optimized overlay networks as the problem changes, when possible.

1.3 Evaluating Multiagent Systems

Many connectivity problems are **NP-Hard**, such as the famous Traveling Salesman Problem, Vertex Cover, *&c.*, meaning that finding optimal solutions to the problems is likely to be intractable. Efficient greedy, heuristic, and local search algorithms do exist to solve such problems, however, if/when a solution is found there is no way of knowing how close the solution is to optimal. Some distributed optimization algorithms—the Adopt family of algorithms, for example [19, 41]—provide a means of terminating when a solution is found that is within a given delta of optimal. This can be problematic, however, if the cost of the optimal solution is unknown. What is ultimately desirable is a theoretical bound on the distance from the optimal solution as a factor (*i.e.*, a relative performance guarantee). In approximation algorithm parlance, an algorithm that is $\varepsilon\text{-}OPT$ produces solutions that are no worse than ε times the cost of the optimal solution. Note that this is different than the notion of "k-optimality" in the distributed constraint reasoning literature, wherein the term refers to a solution in which no subset of k or fewer agents working together can improve the cost of the overall solution [42].

Development of complexity theory has been one of the major achievements of computer science, as it allows for evaluation and comparison of the efficiency of algorithms in a scalable way [43]. Complexity theory proposes evaluation of algorithms in terms of both the number of times the *most expensive operation* is performed and the *amount of data* that are stored in memory. These quantities are taken as a function of the size of the problem. While such metrics do not reveal how much actual time is required
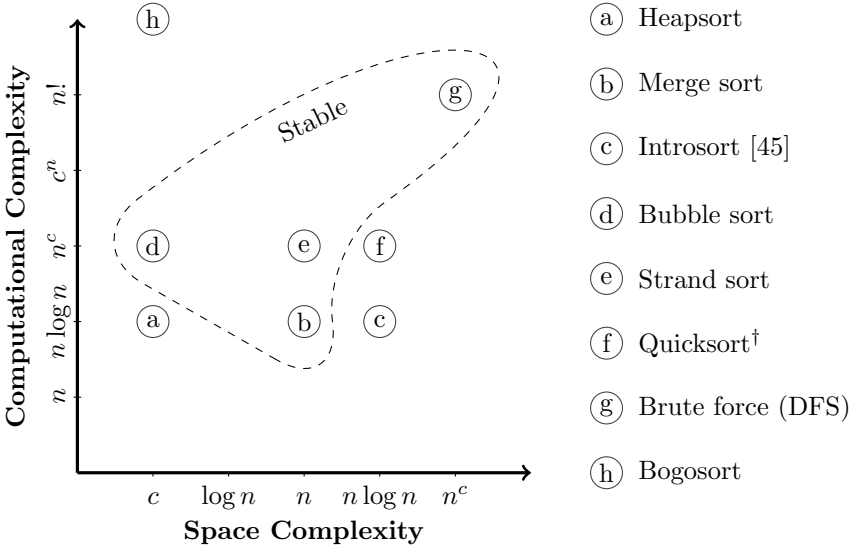
† Assuming that memory pointers require logarithmic space.

Figure 1.6: Worst-case time and space complexities of a number of sorting algorithms.

for a computation on a certain problem instance, they do allow for interpolating how techniques scale as the problem size increases. The assumption that computation speed and memory will double every couple years makes a polynomial factor in the cost irrelevant in the long term [43, 44]. As an example of complexity of computation, Figure 1.6 plots a number of common sorting algorithms with respect to their asymptotic worst-case computational and memory complexities. Sometimes the efficiency of algorithms is at the expense of another metric, such as the stability of the sort (*i.e.*, whether or not the relative ordering of equally weighted entries is preserved). Nonetheless, *a priori* analysis can inform a systems engineer as to which algorithm should be employed in a given domain. Complexity theory can even be used to devise non-uniform algorithms that dynamically choose the best sorting technique at runtime given properties of the input [45].

One of the primary challenges is in creating efficient distributed *asynchronous* algorithms; these types of algorithms are difficult to design since each computing device does not necessarily know the state of the others at any point in time. *Synchronous* distributed algorithms, on the other hand, have the disadvantage of being forced to execute at the speed of the slowest computing device, and require a globally synchronized clock [46, 18]. Asyn-

chronous algorithms have non-deterministic program flow dependent on random fluctuations in communications latency, bandwidth and message loss; a single distributed algorithm may perform drastically differently in one network setting than in another. It is therefore unclear how the metrics used in traditional complexity theory might be extended to distributed algorithms.

In the context of distributed algorithms, communications can be measured on a third axis as in Figure 1.7. For example, the Distributed Stochastic Algorithm (DSA) [47] requires very little memory at each node and very little local computation, but has a relatively high worst-case communications overhead (possibly sending an exponential number of messages in asynchronous networks). The Adopt algorithm [19] also has a high communications overhead and even higher memory and local computation complexities, however, Adopt has guaranteed optimality and bounded approximation, both of which DSA lacks. The Distributed Pseudotree Optimization Procedure (DPOP) [20] has the highest memory and computation complexity, but very low communications overhead. Such analysis *may* be sufficient to inform a systems engineer as to which algorithm might be best suited for a given domain. Probabilistic methods (notably from Queuing Theory [48]) are sometimes used to compare distributed algorithms, but more often than not discrete event simulation is required.

Virtually all literature on measuring communication complexity with respect to the asymptotic complexity of distributed algorithms consists of defining metrics that project the communication axis onto the computation axis [51, 52, 53, 54]. Doing so, however, does not capture the multivariate nature of communication. For example, a message sent over the Internet from one side of the world to the other will always[1] have at least 66 milliseconds of latency[2]. A sensor network may have very low latency but may be constrained by bandwidth/power. Therefore, despite transmitting an equivalent number of messages, a single distributed algorithm that is deployed on the Internet may have very different asymptotic properties than the same algorithm deployed on a sensor network; properties that cannot be solely captured in the computation of the algorithm (*e.g.*, power usage from bandwidth). Assuming local computation is tractable (*e.g.*, polynomial time), then the time required for a distributed algorithm to reach quiescence (*i.e.*, to terminate) can be measured by bounding the time required to transmit the longest causal chain of messages that needs to be sent [51, 53]. Therefore, in contrast to these other measures, it is of utmost importance to minimize the number of communication rounds in distributed algorithms.

In *cooperative multiagent systems* agents are non-adversarial insofar as

---

[1]Assuming correctness of the Special Theory of Relativity, correctness of the Principle of Causality, and lack of subterranean networks.

[2]Taking the diameter of the Earth as 12,756 km and the speed of light as 299,792,458 m/s.
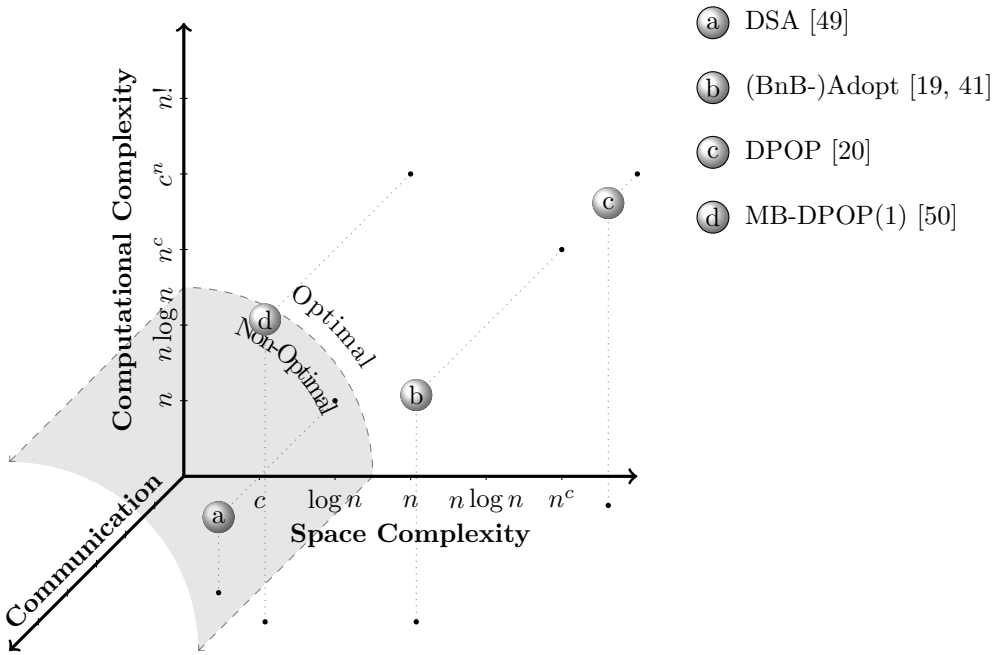
Figure 1.7: Asymptotic complexity bounds for three distributed constraint optimization algorithms.

their goal is to optimize some global objective [17]. In such systems it may be more efficient to have agents disseminate the global state using a *gossip* algorithm [55, 56, 57], for if every agent has a reasonable belief as to the global state then each can locally employ a centralized approximation algorithm to solve the problem. A naïve gossip algorithm can disseminate global state to the entire network in a number of communication rounds equal to the diameter of the network, which will in the worst case be $O(n)$ rounds. Therefore, if all agents in the system are trusted and have hardware sufficient to solve the global problem centrally, then there is little sense in deploying a distributed algorithm that runs in $\omega(n)$ rounds.

It is therefore a primary goal of this dissertation to create algorithms that balance the resource usage of memory, computation, and messaging.

1.4   Contributions

This dissertation shows that a large family of multiagent organization problems can be solved efficiently in a distributed manner. Many of these problems are **NP**-HARD and are therefore intractable in centralized, sequential computation. We show that, in allowing distribution and exploiting locality in the primal-dual schema, speedups are achievable. In fact, we show that distributed algorithms can approximate a solution in linear time and, under certain well defined conditions, can even quiesce in polylogarithmic or even logarithmic time. This is a bit surprising because many of the problems soluble to our approach are known to be **P**-COMPLETE.

These claims are supported by distributed algorithms recently discovered by Sozio [58], Sadeh [59], *& al.* These approaches, however, lack the desired generality and approximation/messaging bounds, the attainability of which is demonstrated herein.

The primary novel contribution of this dissertation is therefore a generalized distributed algorithm that can solve constrained forest problems with a constant optimization bound in no worse than linear communication rounds. We introduce such an algorithm called the *Generalized Distributed Constrained Forest Algorithm* for *Constrained Multidirectional Search*. We show that the algorithm is correct and complete (*i.e.*, it is guaranteed to find a feasible solution if one exists). We also provide a series of examples of how to instantiate this framework for specific problems, including Steiner network problems, art gallery/dominating set problems, and location design & vehicle routing problems.

Strong bounds on the convergence of the algorithm alone is not sufficient, however. We therefore prove that if the edge weights of the input graph are mapped to a metric space that conforms to a specific set of constraints, then the solutions produced by our algorithm are guaranteed to be $2\text{-}OPT$. We show that this requirement is necessary to achieve the speedup from

concurrency. If the input graph is not weighted in a sufficient metric space for the theoretical guarantees to hold, then we show that there exists an ε such that the solution our algorithm discovers is with high probability ε-optimal.

The ultimate problem we are working toward solving is that of *dynamic* multiagent organization. Ideally, the algorithm should not have to completely re-optimize subsequent to every perturbation of the problem. Therefore, we identify a family of events from which our algorithm can recover *faster* than having to re-optimize from scratch. There are also instances when the topological constraints on the desired forest are too expressive to be captured by our current primal-dual model. In some cases the instantaneous approximation bound on the solution is not as important as the *stability* of the solution. For example, for highly dynamic problems it may be better to have an organizational structure that is relatively fixed, as opposed to a structure that is in constant flux as it tries to keep pace with the topological constraints of the ever changing problem. For such cases, we introduce an algorithmic extension called Multiagent Organization with Bounded Edit Distance (Mobed).

Chapter 2

Optimization Using the Primal-Dual Schema

This chapter covers the background mathematical constructs on top of which the algorithms that are later introduced in this dissertation are based.

2.1 Approximation Algorithms

A common procedure for approximating solutions to hard integer programming problems is illustrated in Figure 2.1. First, the hard optimization problem is formulated as an integer program (IP) whose integrality constraints are relaxed to produce a continuous optimization problem. The continuous problem is then solved and converted back to a feasible solution to (IP). The difficulty is that solving the continuous relaxation can be very expensive for problems with many constraints. Let us consider the Steiner network problem as an example. Given a subset of a network's vertices $T \subseteq V$, the Steiner network problem can be represented as the following integer program [60]:

$$\text{minimize } \sum_{e \in E} w(e)x_e$$

$$\text{subject to:} \tag{IP}$$

$$x(\delta(S)) \geq f(S), \quad \forall S \subset V : S \neq \emptyset$$

$$x_e \in \{0,1\}, \quad \quad \forall e \in E,$$

where each variable x_e is an indicator as to whether the edge e is a member of the final Steiner network, $w(e)$ is the weight of edge e, $\delta(S)$ is the set of edges having exactly one endpoint in S, $x(F) \mapsto \sum_{e \in F} x_e$, and $f : 2^V \to \{0,1\}$ is a function such that $f(S) = 1$ if and only if $\emptyset \neq S \cap T \neq T$. A
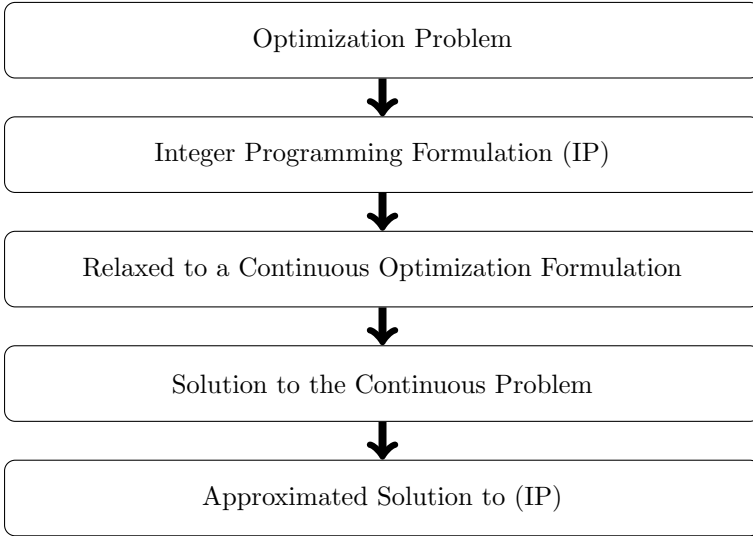
Figure 2.1: Standard procedure for approximating solutions to hard integer programming problems.

continuous optimization relaxation for this integer program is the following linear program:

$$\text{minimize } \sum_{e \in E} w(e)x_e$$
$$\text{subject to:} \qquad\qquad\qquad\qquad\qquad\qquad\quad (\text{LP})$$
$$x(\delta(S)) \geq f(S), \quad \forall S \subset V : S \neq \emptyset$$
$$x_e \geq 0, \qquad\qquad\qquad \forall e \in E.$$

The difficulty arises due to the fact that the $\forall S \subset V$ quantification expands to an exponential number of constraints. Much work has been done to overcome this apparent obstacle, resulting in the realization that not all of the constraints need be evaluated. The process by which this can be accomplished is reviewed in the following sections.

2.2   The Primal-Dual Schema

The idea behind the primal-dual schema is that the dual formulation of (LP)—the continuous optimization problem (a.k.a. the *primal*)—can be used to guide the solution of (LP) [61]. The dual formulation for the Steiner net-

work problem is

$$\text{maximize} \sum_{S \subset V} f(S) y_S$$

subject to:

$$\sum_{S: e \in \delta(S)} y_S \leq w_e, \qquad \forall e \in E \tag{D}$$

$$y_S \geq 0, \qquad \forall S \subset V : S \neq \emptyset.$$

It is a folklore result that the solution to the dual is a lower bound on the solution to the primal. This property is called *weak duality*. Furthermore, the optimal solution to the dual will have the same value as the optimal solution to the primal. This property is called *strong duality*. Finally, the *complementary slackness* property ensures that a primal variable can have a positive value only if its associated dual constraint is tight [62]. These properties conspire to produce a general method for sequentially approximating hard integer programming problems, which is often called the primal-dual strategy or schema. This general method is given in Algorithm 1.

Algorithm 1 The general primal-dual schema, as adapted from [63].

1: **procedure** PRIMAL-DUAL(IP)
2:     Let (CO) be the continuous optimization relaxation of (IP).
3:     Let (D) be the dual to (CO).
4:     Initialize vectors $x = 0$ and $y = 0$ which are, respectively, the solutions for (CO) and (D). /* Note that y will initially be dual feasible, but x may be primal infeasible. */
5:     **while** x is primal infeasible **do**
6:         While maintaining dual feasibility, deterministically increase the dual values y_i until one dual variable becomes tight (*i.e.*, that variable cannot be increased any more without breaking a dual constraint).
7:         For a subset of the tight dual constraints, increase the primal variable corresponding to them by an integral amount.
8:     **end while**
9:     The cost of the dual solution is used as a lower bound on OPT.
10: **end procedure**

The first algorithm using the primal-dual schema was devised to solve the assignment problem and is due to Kuhn [64], who named it the "Hungarian Method". The method was later dubbed *primal-dual* by Dantzig, Ford, and Fulkerson [65]. Although they did not originally state their work as using the primal-dual method, Bar-Yehuda and Even proposed the first approximation algorithm based on the schema in solving the weighted vertex cover problem [66].

Proposing *distributed* algorithms using the primal-dual schema has been the subject of study of a number of recent results [67, 59]. Problems that have been studied using this schema include Steiner problems [68], point-to-point connectivity problems [69], distributed scheduling [70], vertex cover [71, 72], facility location [58], and k-connectivity [59]. Some of these approaches provide theoretical runtime bounds and others provide theoretical approximation bounds, however, almost none provide both. The only approach that does provide both approximation and runtime bounds is due to Sadeh [59], however, the runtime bound is superlinear. What these approaches seem to miss is the following observation: Pushing up the dual variables until they become tight (*i.e.*, lines 6 and 7 of Algorithm 1) appears to require only local information. This dissertation shows that, with proper bookkeeping, the entire **while** loop in Algorithm 1 can be executed in parallel between agents. In doing so, it is possible to get good bounds on *both* runtime *and* optimality. Creating distributed algorithms that realize this claim is the primary contribution of this dissertation.

2.3   Generalizing the Schema: Proper Functions

One of the early generalized applications of the primal-dual technique to approximation algorithms was proposed by Goemans and Williamson [73, 14]. Recall that in the formulation of the Steiner network problem (*q.v.* §2.1), the primal constraints are defined using an indicator function f. For the case of the Steiner network problem, f is defined such that $f(S) = 1$ if and only if $\emptyset \neq S \cap T \neq T$. The intuition behind this function is that a partially constructed portion of the network, $S \subseteq V$, is *active* (*i.e.*, it needs to continue growing) if the network does not yet contain all necessary vertices in T. Goemans and Williamson had the brilliant insight that by simple modification to this f function the same paradigm can be used to solve a number of different connectivity problems [73]. For example, let T be the set of depot vertices and let R be the set of non-depot vertices; then setting $f(S) \mapsto 1$ if and only if $S \cap T = \emptyset$ will solve the Location Design and Routing Problem. Examples of other functions that solve preexisting connectivity problems are given in Table 2.1.

A function on the powerset of a set of vertices, $f : 2^V \to \{0, 1\}$, is said to be *proper* if the following are true:

Null Property: $f(\emptyset) = 0$;

Symmetry Property: $\forall S \subseteq V : f(S) = f(V \setminus S)$; and

Disjointness Property: $\forall A, B \subseteq V : (A \cap B = \emptyset) \implies f(A \cup B) \leq \max\{f(A), f(B)\}$.

NAME	PROBLEM	$f(S) = 1$ iff...
Minimum-weight perfect matching	Find a minimum-cost set of non-adjacent edges that cover all vertices.	$|S|$ is odd.
T-join [73]	Given an even subset T of vertices, find a minimum-cost set of edges that has odd degree at vertices in T and even degree at vertices not in T.	$|S \cap T|$ is odd.
Minimum spanning tree/forest	Find a minimum weight forest that maximizes connectivity between vertices.	$\exists u \in S, v \notin S : u \rightsquigarrow v \in G.$
Generalized Steiner tree	Find a minimum-cost forest that connects all vertices in T_i for $i = 1,\ldots,p$.	$\exists i \in \{1,\ldots,p\} : \emptyset \neq S \cap T_i \neq T_i.$
Point-to-point connection	Given a set $C = \{c_1,\ldots,c_p\}$ of sources and a set $D = \{d_1,\ldots,d_p\}$ of destinations in a graph $G = \langle V,E \rangle$, find a minimum-cost set F of edges such that each source-destination pair is connected in F.	$|S \cap C| \neq |S \cap D|.$
Partitioning (w/triangle inequality)	Find a minimum-cost collection of vertex-disjoint trees, paths, or cycles that cover all vertices.	$S \neq 0 (\text{mod } k).$
Location design/routing	Select depots among a subset D of vertices of a graph $G = \langle V,E \rangle$ and cover all vertices in V with a set of cycles, each containing a selected depot, while minimizing the sum of the fixed costs of opening depots and the sum of the costs of the edges in the cycles.	$\emptyset \neq S \subseteq V.$

Table 2.1: Constrained forest problems and their associated indicator functions.

If f is proper then Goemans and Williamson's algorithm has two additional properties: the algorithm will run in polynomial time, and the solution it produces will be 2-*OPT* (*i.e.*, the cost of the solution will be no more than two times the cost of the optimal solution). Goemans and Williamson named the class of problems representable as proper functions as *constrained forest problems*. Many constrained forest problems are **NP**-HARD, hence the motivation to find an approximate solution in polynomial time.

The space of functions amenable to solution via the primal-dual schema has also been expanded to include other families. For example, both *well spaced functions* [60] and *supermodular functions* [74] have been investigated. A function $f : 2^V \rightarrow \{0, \rho_1, \rho_2, \ldots, \rho_k\}$ is said to be *well spaced* if $\forall i \in \{1, 2, \ldots, k-1\} : \rho_{i+1} \geq |A|\rho_i$, where A is the set of active vertices. A function $f : 2^V \rightarrow \mathbb{Z}$ is said to be *weakly supermodular* if $f(V) = 0$ and for every $A, B \subseteq V$ at least one of the following holds:

- $f(A) + f(B) \leq f(A \setminus B) + f(B \setminus A)$

- $f(A) + f(B) \leq f(A \cap B) + f(A \cup B)$.

Note that proper functions are weakly supermodular; an algorithm for solving constrained forest problems defined by weakly supermodular functions was first given in [75] and has an approximation factor of $2 \sum_{i=1}^{|A|} \frac{1}{i}$. This was later improved to a factor of 2 by Jain in [74].

2.4   Conclusions

While the primal-dual schema has been known for many years, it did not reach its current level of awareness until the approximation algorithms community took hold of it in the 1990s. Thanks to the discoveries of Aggarwal [60], Goemans & Williamson [75], Vazirani [61], *& pl. al.*, it is now known that the schema works very well for bounded approximation. They were able to realize that not all of the exponential constraints in the primal and exponential variables in the dual need to be represented, let alone computed, in order to approximate a solution with bounded optimality in polynomial time. Furthermore, a large family of problems defined by proper functions and their extensions are amenable to the schema [14].

Of the few distributed approaches for solving constrained forest problems that do exist, some provide theoretical runtime bounds and others provide theoretical approximation bounds, however, almost none provide both and absolutely none have sublinear runtime bounds. A major contribution of this thesis is the discovery that further efficiency improvements can be achieved by reconstructing the schema in a distributed manner to exploit the relative

locality of the interactions in the algorithm. We will show that for a large family of problems we can maintain bounded approximation in a *linear*—and in some cases even polylogarithmic—number of phases.

In the next chapter a new distributed algorithm for solving constrained forest problems is introduced based upon the primal-dual schema. The algorithm is guaranteed to run in a linear number of communication rounds and, in certain well defined circumstances, logarithmic communication rounds. Furthermore, if the problem is encoded properly, the solution our algorithm discovers is guaranteed to be no worse than 2 times the cost of optimal. In Chapter 4 it will be shown that—even without proper encoding of the problem—the expected value of the approximation bound will be constant for a large family of edge weight distributions. In Chapter 5, examples are given as to how this framework can be instantiated (*e.g.*, furnished with distributed protocols) for specific problems and their associated proper functions. Also in that chapter the approach is empirically evaluated for several real-world domains. Finally, in Chapter 6 we provide algorithmic extensions that can handle dynamic changes of the underlying structure.

Chapter 3

The Generalized Distributed Constrained Forest Algorithm

Given a weighted graph with specified start and goal vertices, a *bidirectional graph search* performs two simultaneous searches: one initiated from the start vertex and the other from the goal vertex. When there is an overlap in the two searches' fringes, a path between the start and goal is discovered. Furthermore, if the goal-test function is modified such that the algorithm terminates when both searches expand the same node and an optimal search algorithm is used (*e.g.*, A*), then the resulting path will be of minimum length. Now consider the generalization of this problem in which the search is further constrained by specifying an arbitrary number of intermediate vertices that must be interconnected through some path. We call such intermediate vertices *terminals*. This generalization captures a number of different problems—many of which are **NP**-HARD—including path planning with waypoints, finding a minimum length tour (*vi&.* the traveling salesman problem), and the Steiner tree problem [60], which are collectively called *constrained forest problems*.

The idea of multidirectional search has existed for quite some time in the context of unconstrained search [76], however, there is very little in the literature on multidirectional *constrained* search. An algorithm was recently discovered for the related problem of distributedly clearing the *entire* search space [77]. There have also been a number of resent results in the context of multiagent path planning, such as the AA* algorithm employed in the AGENTFLY framework [78], however, these approaches are primarily concerned with online planning and distributed deconfliction. Similar speedups in graph search have also recently been achieved through disk-

based search [79]. A family of distributed best-first search algorithms also exists [19], however, this is in the context of distributed constraint reasoning and not general graph search.

We define *multidirectional graph search* as the process of finding a tree in the search space that connects all of the terminals by simultaneously performing a search emanating from each terminal. Given one intelligent agent per terminal, this approach is inherently distributable, with potential for significant speedups from concurrency. The difficulty is in proving that

1. the concurrent searches will not induce a cycle in the output;

2. the resulting tree will have an upper bound on its weight;

3. the solution technique is resilient to the fact that each agent only has local information and no shared memory; and

4. the overhead of these invariants will not outweigh the potential speedup from concurrency.

This chapter introduces the algorithm and prove these properties. After the introduction and definition of some requisite formalism, property 1 is proven below in Corollary 1 of §3.1.3. Property 2 is proven in Proposition 3, property 3 will be apparent from the definition of the method itself (Algorithm 2), and property 4 is proven by showing that the algorithm requires at most linear—and sometimes only logarithmic—messaging rounds in Proposition 5 of §3.2.

3.1   Multidirectional Graph Search for Constrained Forests

The basic mechanism of the algorithm is quite simple: Each terminal concurrently performs a best-first search using a special potential function to prioritize the fringe nodes. During each round the node of minimum potential is expanded from each search's fringe, thus adding an edge to the final forest. When two of the concurrent searches expand the same node, then they merge their remaining fringes and continue as a single search. When all of the searches have merged together, it implies that the forest spans all of the terminals—meaning that the forest is a feasible solution—so the algorithm terminates. An overview of the entire process is given in Algorithm 2.

The remainder of this section provides the notation and descriptions required to formally define and model the algorithm. This is used at the end of the section to prove correctness and completeness, and will be expanded later in the chapter to provide formal bounds on the runtime and performance of the algorithm.

Algorithm 2 The multidirectional graph search algorithm.

1: **procedure** MULTIDIRECTIONAL-GRAPH-SEARCH(T, v, w, δ)

Require: T is the set of terminals. $v \in T$ is the terminal running this instance of the search algorithm. w is a function that maps edges to their associated weight in the metric space $\left[\tilde{\omega}, \frac{3}{2}\tilde{\omega}\right] \in \mathbb{Q}$. δ is a successor function such that $\delta(S)$ is the set of edges having exactly one endpoint in S.

Ensure: $H = \langle \tilde{V}, \tilde{E} \rangle$ is the resulting forest.

2:      $\tilde{V} \leftarrow \{v\}$ /* The initial solution has just our vertex... */

3:      $\tilde{E} \leftarrow \emptyset$ /* ...and no edges */

4:      $F \leftarrow \delta(\{v\})$ /* The fringe of our search, initialized to v's incident edges */

5:      $\forall u \in V : g(v) \leftarrow 0$ /* Initialize the path-cost function, implicitly setting $y_S \leftarrow 0$ for all $S \subset V$ */

6:      **while** $(\tilde{V} \cap T \neq T) \wedge (F \neq \emptyset)$ **do**   /* while H does not contain all terminals and the fringe is not empty */

7:          Find an edge in $e = \langle v, u \rangle \in F$ such that $\varepsilon = w(e) - g(u) - g(v)$ is minimized.

8:          **if** u either is being or already was expanded by another search **then**

9:             Union $\tilde{V}$, $\tilde{E}$, F, and g with the respective data structures of the search that already expanded u and then merge our execution with that search.

10:             **if** The other search also expanded the edge $\langle v, u \rangle$ this round **then**

11:                $\varepsilon \leftarrow \frac{\varepsilon}{2}$

12:             **end if**

13:          **end if**

14:          **for all** $k \in \tilde{V} : k$ is incident to an edge in the fringe **do**

15:             $g(k) \leftarrow g(k) + \varepsilon$ /* Implicitly set $y_{\tilde{V}} \leftarrow y_{\tilde{V}} + \varepsilon$ */

16:          **end for**

17:          $F \leftarrow (F \setminus \{e\}) \cup \delta(\{u\})$ /* Remove e and add the edges incident to u */

18:          $\tilde{V} \leftarrow \tilde{V} \cup \{u\}$

19:          $\tilde{E} \leftarrow \tilde{E} \cup \{e\}$

20:      **end while**

21: **end procedure**

3.1.1   A Primal-Dual Formulation

In order to prove the various properties of the algorithm, it is useful to cast the search problem as an equivalent optimization problem. In fact, the underlying mechanism of the algorithm is that it casts the optimization problem as an integer program whose integrality constraints are relaxed to produce a continuous optimization problem (a.k.a. the *primal*). The dual formulation of the primal is then used to guide the solution of the primal. The solution is then converted back to a feasible solution of the integer program. This is a well known general method for sequentially approximating hard integer programming problems, which is often called the *primal-dual strategy* or *schema* [61].

Recall that the *complementary slackness* property (*q.v.* §2.2) ensures that a primal variable can have a positive value only if its associated dual constraint is tight. This property allows us to use the dual to guide the solution of the primal. In a more tangible sense, it is what informs us as to the potential function that should be used to prioritize the fringe nodes in the best-first search.

Let $f : 2^V \to \{0, 1\}$ be an auxiliary function such that $f(S) = 1$ if and only if the set S contains *some* terminals but not *all* terminals:

$$f(S) \mapsto \begin{cases} 1 & \text{if } \emptyset \neq S \cap T \neq T, \\ 0 & \text{otherwise.} \end{cases}$$

Note that f is *proper* (*q.v.* §2.3). Recall from §2.1 that the optimization problem of finding a minimum weight tree that spans the terminals can then be captured as the following integer program [60]:

$$\begin{aligned}
\text{minimize } & \sum_{e \in E} w(e) x_e \\
\text{subject to:} & \quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad \text{(IP)} \\
& x(\delta(S)) \geq f(S), \quad \forall S \subset V : S \neq \emptyset \\
& x_e \in \{0, 1\}, \quad\quad\quad\quad \forall e \in E,
\end{aligned}$$

where each variable x_e is an indicator as to whether the edge e is a member of the final tree, V is the set of the graph's nodes, $\delta(S)$ is the set of edges having exactly one endpoint in S, and $x(F) \mapsto \sum_{e \in F} x_e, \forall F \subseteq E$. Therefore, any forest $H = \langle \tilde{V}, \tilde{E} \rangle \subseteq G$ will be a feasible solution to the problem if $\left(f(\tilde{V}) = 0 \right) \implies \left(\tilde{V} \cap T = T \right).$

Let (LP) denote the linear programming relaxation of (IP) obtained by

replacing the integrality restriction with $x_e \geq 0$. The dual of (LP) is

$$\text{maximize} \sum_{S \subset V} f(S) y_S$$

$$\text{subject to:}$$

$$\sum_{S:e \in \delta(S)} y_S \leq w(e), \qquad \forall e \in E \tag{D}$$

$$y_S \geq 0, \qquad \forall S \subset V : S \neq \emptyset.$$

An edge is *tight* if $w(e) = \sum_{S:e \in \delta(S)} y_S$. Let Z_{LP}^* be the cost of the optimal solution to (LP) and let Z_{IP}^* be the cost of the optimal solution to (IP). It is easy to see that $Z_{\text{LP}}^* \leq Z_{\text{IP}}^*$.

The way the algorithm's multidirectional search corresponds to solving the primal-dual optimization problem is as follows. Since the algorithm starts off with an empty tree, the initial solution, H, is dual feasible but not necessarily primal feasible. By choosing an edge from the fringe that minimizes the potential function (*cf.* line 7 of Algorithm 2), we are essentially choosing a dual constraint to become tight. By the complementary slackness property, this means that an associated primal variable (*i.e.*, the edge) can become a part of the final tree (line 19). The path-cost update (line 15) is essentially closing the duality gap, which implicitly pushes up the values of the dual variables y_S for all $S \subseteq V$ that are currently involved in a search. The potential function is what ensures that H remains dual feasible (this is proven below in Proposition 2). Finally, the while loop (line 6) ensures that H is primal feasible.

3.1.2 The Distributed Model

This section serves to define a model of distributed computation that captures the primal-dual optimization scheme on which the algorithm is based. This distributed model is then used in the following section to provide theoretical bounds on the optimality of the algorithm.

We assume that the communications network provides guaranteed delivery and ordering of messages, however, there may be arbitrary latency (*i.e.*, the network is asynchronous [46]). We further assume that all agents are honest and correct and there are no malicious interlocutors, and thus need not consider the problem of Byzantine failure. There is one agent per terminal. The agents are non-adversarial insofar as their primary goal is to find a feasible solution to the search problem. The collective is therefore a *cooperative multiagent system* [17]. Agents' perceptions of the graph are consistent, possibly through the use of a distributed consensus algorithm, however, each agent only requires local knowledge of its neighbors in the graph. Each vertex has a unique identifier with a globally agreed ordering. This ordering can

be used to construct a total ordering over the edges (*e.g.*, by combining the unique identifiers of the incident vertices).

The algorithm is round-based, with each round corresponding to a single iteration of the while loop on line 6 of Algorithm 2. The rounds proceed asynchronously between each of the searches. Therefore, as the searches merge throughout the execution of the algorithm, the rounds naturally become synchronized.

Let $H_t = \langle \tilde{V}_t, \tilde{E}_t \rangle$ be the partially constructed tree at the beginning of round t (*i.e.*, the tth iteration of the while loop on line 6 of Algorithm 2). Let $\mathcal{C}_t$ be the set of connected components in H_t. For sake of brevity and simplicity, let $\mu_t : V \to \mathcal{C}_t$ be a function mapping vertices to their associated connected component during round t; therefore, $\mu_t(v) \mapsto C_i \implies v \in C_i (\in \mathcal{C}_t)$. A connected component $C_i \subseteq V$ such that $f(C_i) = 1$ (*i.e.*, C_i contains at least one terminal but not *all* of the terminals) is still actively performing its search; such components are therefore called *active*. The fringe of a connected component's search is therefore the set of edges in the cut-set of the component. Let $g_t : V \to \mathbb{Q}$ be a mapping of vertices to a rational number during round t. In the context of search, these values represent an estimate on the path-cost of each vertex, however, they also represent an upper bound on the duality gap in the associated optimization problem.

Let $J_t : V \times V \to \{0, 1\}$ be a binary relation defining which edges will be added to the tree during round t. Each active component will choose to add the edge in its fringe that has minimal duality gap. Therefore, $J_t(u, v) = 1$ if and only if $f(\mu_t(u)) = 1$ and

$$\langle u, v \rangle = \underset{\langle i,j \rangle \in \delta(\mu_t(u))}{\arg\min} \; w(\langle i, j \rangle) - g_t(i) - g_t(j). \tag{3.1}$$

Ties in the minimization are broken based upon the ordering of the edges. Let J^+ denote the transitive closure of J.

Note that J does not necessarily commute: $(\exists t : J_t(u, v)) \;\not\!\!\!\implies\; (\exists s : J_s(v, u))$. In the case when $J_t(u, v) = J_t(v, u)$, the union between $\mu_t(u)$ and $\mu_t(v)$ is said to be *mutual*. Also note that as long as there exists a feasible solution to (IP) then the minimization ensures that each search must have exactly one edge in the fringe that becomes tight each round:

$$\forall C \in \mathcal{C}_t : f(C) = \sum_{\langle u,v \rangle \in \delta(C)} J_t(u, v).$$

H_t is the partially constructed tree during round t, initialized to $H_0 = \langle V, \emptyset \rangle$. The partial tree is updated each round with the set of all edges that became tight during the round:

$$H_{t+1} = H_t \cup \{\langle u, v \rangle \in E : J_t(u, v) \vee J_t(v, u)\}.$$

For a set $S \subseteq V$, let y_S be the dual variable associated with S. Initially all such variables are set to zero. Note that in actuality these variables need not be made part of an implementation of the algorithm; they exist solely for the purpose of proving properties of the algorithm. These dual variables are implicitly updated as follows:

$$y_S \leftarrow \begin{cases} \frac{w(\langle i,j \rangle) - g_t(i) - g_t(j)}{1 + J_t(j,i)} & \text{if } \exists i \in S \in \mathcal{C}_t, j \notin S : J_t(i,j), \\ 0 & \text{otherwise.} \end{cases} \tag{3.2}$$

The g values are initialized such that $\forall v \in V : g_0(v) = 0$. They are updated each round such that

$$g_{t+1}(v) = g_t(v) + y_{\mu_t(v)}. \tag{3.3}$$

The value $g_t(v)$ can therefore be interpreted as the amount of slack remaining in the dual variables during round t before an edge incident to vertex v becomes tight.

Let τ be the number of rounds required for the algorithm to reach quiescence. Therefore, τ is the earliest round during which there are no active components: $\tau = \min_{t \in \mathbb{N}_0} (\forall C \in \mathcal{C}_t : f(C) = 0)$.

3.1.3 Correctness Proofs

Lemma 1. *Any cycle in the intersection graph[1] of H_{t+1} formed from $\mathcal{C}_t$ must consist solely of edges along the cuts between active components.*

Proof. Assume, on the contrary, that there exists a cycle containing an edge that is incident to at least one inactive component. Let $\langle u, v \rangle$ be such an edge and assume $\mu_t(v)$ is inactive; then (3.1) implies that v's connected component has no outgoing edges,

$$\forall i \in \mu_t(v) : (\neg \exists j \in V : J_t(i,j)),$$

which contradicts the fact that $\langle u, v \rangle$ is in a cycle. □

The *potential cost* of an edge is the fractional quantity associated with ε on line 7 of Algorithm 2.

Lemma 2. *Any cycle in the intersection graph of H_{t+1} formed from $\mathcal{C}_t$ must consist of edges of equal potential cost.*

---

[1] An *intersection graph* is formed from a family of sets $\mathcal{C} = \{C_1, C_2, C_3, \ldots\}$ by creating one super-vertex v_i for each set C_i and connecting it to any other vertex v_j by an edge whenever v_i and v_j's corresponding sets have a nonempty intersection. This produces the edge set $\{\langle v_i, v_j \rangle : \mu(v_i) \cap \mu(v_j) \neq \emptyset\}$.

Proof. Let $e_1 = \langle u_1, v_1 \rangle$ be an edge in a cycle; then (3.1) implies that all edges in a cycle must be cuts between existing connected components. Therefore, $\mu_t(u_1) \neq \mu_t(v_1)$. Furthermore, there must be another edge in the cycle, $e_2 = \langle u_2, v_2 \rangle$, such that $\mu_t(v_2) = \mu_t(u_1)$. It must also be true that $J_t(u_1, v_1) = J_t(u_2, v_2) = J_t^+(u_1, v_2) = 1$. By Lemma 1 all components in the cycle are active. Therefore, applying (3.1) gives

$$w(e_1) - g_t(u_1) - g_t(v_1) \leq w(e_2) - g_t(u_2) - g_t(v_2).$$

In general, this inequality will hold for the incoming and outgoing edges of any connected component in the cycle. Therefore, by transitivity,

$$
\begin{aligned}
w(e_1) - g_t(u_1) - g_t(v_1) &\leq w(e_2) - g_t(u_2) - g_t(v_2) \\
&\leq w(e_1) - g_t(u_1) - g_t(v_1),
\end{aligned}
$$

implying that

$$w(e_1) - g_t(u_1) - g_t(v_1) = w(e_2) - g_t(u_2) - g_t(v_2).$$

$\square$

Proposition 1. *The intersection graph (q.v. footnote 1 on page 31) of H_{t+1} formed from $\mathcal{C}_t$ is acyclic.*

Proof. Assume, on the contrary, that there is a round t during which a cycle of length ℓ is formed. Since the graph is simple, $\ell > 1$. By Lemma 2, all of the edges in the cycle must be of equal potential cost. Therefore, each connected component will have had a tie between two fringe edges which must have been broken using the edge ordering. Therefore, either $\ell = 1$ or there are two edges with the same unique identifier, both of which are contradictions. $\square$

Claim 1. *If all edge weights are coprime, then Proposition 1 will hold even if the unique identifier ordering assumption does not.*

Corollary 1. *$H_0, \ldots, H_\tau$ are all acyclic.*

Proof. Since $H_0 = \langle V, \emptyset \rangle$, the base case is acyclic. Induction over Proposition 1 then proves the corollary. $\square$

Lemma 3. *Let t' be the round during which an edge $e = \langle u, v \rangle$ is added to the spanning forest. Then e will not be in the cut of any component in a subsequent round: $\forall t > t', C \in \mathcal{C}_t : e \notin \delta(C)$.*

Proof. $\mu_{t'+1}(u) = \mu_{t'+1}(v) = \mu_{t'}(u) \cup \mu_{t'}(v)$. Therefore, in all rounds subsequent to t' both endpoints of e are in the same component and thus cannot be in the fringe. $\square$

Proposition 2. *The vector y is a feasible solution to* (D) *and has the property*

$$\sum_{e \in H_\tau} w(e) \leq \sum_{e \in H_\tau} \sum_{S: e \in \delta(S)} y_S.$$

Proof. The fact that y is a feasible solution to (D) is a straightforward result of the fact that y is initially zero and is updated according to (3.2). Let t be the round during which an edge $e = \langle u, v \rangle \in H_\tau$ was added to the forest. It follows from the definition of (3.3) that

$$\left(g_t(u) = \sum_{i=0}^{t-1} y_{\mu_i(u)} \right) \bigwedge \left(g_t(v) = \sum_{i=0}^{t-1} y_{\mu_i(v)} \right).$$

Furthermore, at the beginning of round t the potential for e is $\varepsilon = w(e) - g_t(u) - g_t(v)$. Once e is added to H_t, the dual variables $y_{\mu_t(u)}$ and $y_{\mu_t(v)}$ are updated according to (3.2). Then there are three possible cases:

1. $f(\mu_t(u)) = f(\mu_t(v)) = J_t(u, v) = J_t(v, u) = 1$;

2. $f(\mu_t(u)) = f(\mu_t(v)) = J_t(u, v) + J_t(v, u) = 1$; or

3. $f(\mu_t(u)) + f(\mu_t(v)) = 1$.

In case 1 (handled on line 11 of Algorithm 2),

$$y_{\mu_t(u)} + y_{\mu_t(v)} = \frac{\varepsilon}{1 + J_t(v, u)} + \frac{\varepsilon}{1 + J_t(u, v)} = \varepsilon$$

$$\implies w(e) = \sum_{i=0}^{t} \left(y_{\mu_i(u)} + y_{\mu_i(v)} \right). \quad (3.4)$$

For case 2, assume without loss of generality that $J_t(u, v) = 1$ and $J_t(v, u) = 0$. For case 3, assume without loss of generality that $f(\mu_t(u)) = 1$ and $f(\mu_t(v)) = 0$. Then for both of these cases note that $y_{\mu_t(u)} = \frac{\varepsilon}{1 + J_t(v, u)} = \varepsilon$, implying

$$w(e) = y_{\mu_t(u)} + \sum_{i=0}^{t-1} \left(y_{\mu_i(u)} + y_{\mu_i(v)} \right). \quad (3.5)$$

Lemma 7 implies that the summations in (3.4) and (3.5) comprise all sets that cut e, thus completing the proof. $\qquad\square$

Lemma 4 (Theorem 3.6 of [14])**.** *For any $t \leq \tau$, let I be the intersection graph of the final spanning forest H_τ formed from $\mathcal{C}_t$. Remove all isolated vertices in I that correspond to components in $\mathcal{C}_t$ that are inactive. Then no leaf in I corresponds to an inactive component.*

Proof sketch. The contrary leads to the necessity that at least one edge incident to an inactive component cannot be a part of H_τ, which is a contradiction. $\square$

Proposition 3. *If there exists an $\tilde{\omega}$ such that all edge weights are in the range $[\tilde{\omega}, \frac{3}{2}\tilde{\omega}]$ then the cost of the final tree H_τ is bounded above by $\left(2 - \frac{2}{|V|}\right) Z_{IP}^*$.*

Proof. Without loss of generality, assume $y_S > 0 \implies f(S) = 1$. This property ensures that $\sum_{S \subset V} y_S \leq Z_{LP}^*$. Since it is clear that $Z_{LP}^* \leq Z_{IP}^*$, we then have $\sum_{S \subset V} y_S \leq Z_{LP}^* \leq Z_{IP}^*$. Proposition 2 ensures that the weight of H_τ is

$$\sum_{e \in H_\tau} w(e) \leq \sum_{e \in H_\tau} \sum_{S:e \in \delta(S)} y_S = \sum_{S \subset V} y_S |H_\tau \cap \delta(S)|.$$

To prove this theorem we will show by induction over the construction of H_τ that

$$\sum_{S \subset V} y_S |H_\tau \cap \delta(S)| \leq \left(2 - \frac{2}{|R|}\right) \sum_{S \subset V} y_S. \tag{3.6}$$

The base case certainly holds at round zero since all y_S are initialized to zero. Let A be the set of edges added to the spanning forest during an arbitrary round t. For each edge $e = \langle u, v \rangle \in A$, let ε_e denote the potential value associated with that edge: $\varepsilon_e = w(e) - g_t(u) - g_t(v)$. Now sort A according to descending potential value, such that $e_i \in A$ is the edge with the ith largest potential. At the end of a round t, the left-hand side of (3.6) will increase by at most

$$\sum_{C \in \mathcal{C}_t : f(C)=1} y_C |H_\tau \cap \delta(C)| = \sum_{C \in \mathcal{C}_t : f(C)=1} \sum_{\langle u,v \rangle \in A : u \in C} \frac{J_t(u,v)\varepsilon_e}{1 + J_t(v,u)} |H_\tau \cap \delta(C)|. \tag{3.7}$$

If we can prove that this increase is bounded above by the increase of the right-hand side, namely

$$\left(2 - \frac{2}{|V|}\right) \sum_{i=1}^{|A|} i \times \varepsilon_{e_i}, \tag{3.8}$$

then we will be done.

First, observe that (3.7) can be bounded above by

$$\left(\max_{e \in A} \varepsilon_e\right) \sum_{C \in \mathcal{C}_t : f(C)=1} \sum_{\langle u,v \rangle \in A : u \in C} J_t(u,v) |H_\tau \cap \delta(C)|. \tag{3.9}$$

Next, observe that (3.8) can be bounded below by

$$\left(\min_{e \in A} \varepsilon_e\right) \left(2 - \frac{2}{|V|}\right) |A| \left(\frac{|A|}{2} + \frac{1}{2}\right). \tag{3.10}$$

Now let I be the intersection graph of the final spanning forest H_τ formed from $\mathcal{C}_t$. Remove all isolated vertices in I that correspond to inactive components in $\mathcal{C}_t$. Notice that I is a forest, and by Lemma 4 no leaf in I corresponds to an inactive component. Let N_a be the set of vertices in I that correspond to active components: $N_a = \{C \in \mathcal{C}_t : f(c) = 1\}$, and let N_i be the set of vertices in I corresponding to inactive components. The degree of a vertex v in I corresponding to component C, denoted d_v, must be $|\{e \in \delta(C) : e \in H_\tau\}|$. Then the summation of (3.9) can be rewritten as

$$\sum_{v \in N_a} d_v = \sum_{v \in N_a \cup N_i} d_v - \sum_{v \in N_i} d_v \leq (2|N_a| - 2).$$

This inequality holds since I is a forest with at most $|N_a| + |N_i| - 1$ edges, and since each vertex corresponding to a guarded component has degree at least 2. Substituting this result back into (3.9) we have

$$(3.7) \leq (3.9) \leq \left(\max_{e \in A} \varepsilon_e\right)\left(2 - \frac{2}{|V|}\right)|A|,$$

since the number of active components is always no more than $|V|$. Therefore, $(3.7) \leq (3.10) \leq (3.8)$ if during every round the following invariant holds:

$$\max_{e \in A} \varepsilon_e \leq \left(\frac{|A|}{2} + \frac{1}{2}\right)\min_{e \in A} \varepsilon_e,$$

which is clearly true because all of the edge weights are in $\left[\tilde{\omega}, \frac{3}{2}\tilde{\omega}\right]$. Hence the theorem is proven. $\qquad\square$

By using induction over the growth of a connected component, it is possible to identify an alternative set of conditions under which the algorithm is guaranteed to produce 2-optimal solutions that do not rely on assumptions about the distribution of edge weights. Roughly speaking, if we can bound the number of edges that will later be added to an active component then we can also prove 2-optimality. In order to prove this, let $d : \mathbb{N}_0 \times 2^V \to \mathbb{N}_0$ be an auxiliary function such that $d(t, S \subseteq V)$ returns the number of edges incident to the vertices in S that will become tight in rounds later than t:

$$d(t, S) \mapsto \left|\left\{\langle x, y\rangle \in \bigcup_{v \in S} \delta_t(v) : \left(\exists t' : t < t' \leq \tau \wedge \left(J_s(x, y) \vee J_s(y, x)\right)\right)\right\}\right|.$$

Proposition 4. *Algorithm 2 is 2-optimal if* $\forall e = \langle u, v \rangle \in \tilde{E}$:

$$\left(\exists t : \underbrace{J_t(u,v) = 1 \wedge J_t(v,u) = 0}_{e \text{ was not added mutually}} \wedge \underbrace{d(t, \{u\}) \leq 1}_{} \right)$$

(*at most one additional edge incident to u becomes tight in a later round*)

$$\bigvee \left(\exists t : \underbrace{J_t(u,v) = J_t(v,u)}_{e \text{ was added mutually}} \wedge \underbrace{d(t, \{u, v\}) \leq 3}_{} \right) .$$

(*at most three additional edges incident to u and v become tight in a later round*)

Proof. We still need to ultimately prove the following invariant:

$$\sum_{e \in H_\tau} w(e) \leq 2 \sum_{S \subset V} y_S.$$

Now let us consider the addition of a single edge $e = \langle u, v \rangle$ of potential $\varepsilon = w(e) - g_t(u) - g_t(v)$ during an arbitrary round t. Without loss of generality assume that $J_t(u, v) = 1$, but $J_t(v, u)$ is arbitrary. The increase of the right-hand side of the invariant is always ε. Now, for sake of analysis, assume that the constrained forest $H_t : t \in \{0, 1, \ldots, \tau\}$ is actually represented as a directed graph, where each edge $\langle i, j \rangle \in H_t$ implies that i's component chose to add the edge to j during round t:

$$\langle i, j \rangle \in H_t \implies J_t(i, j).$$

Therefore, if an edge choice is mutual then $\langle i, j \rangle$ and $\langle j, i \rangle$ will both be in H_t. Now let I be the intersection graph of H_τ formed from $\mathcal{C}_t$. For every component $C \in \mathcal{C}_t$, let i_C denote the vertex in I corresponding to C. Then the increase of the left-hand side of the invariant can be exactly represented as

$$\begin{cases} \varepsilon \times \deg^+(i_{\mu_t(u)}), & \text{if } J_t(u,v) \neq J_t(v,u) \\ \frac{\varepsilon}{2} \times \left(\deg^+(i_{\mu_t(u)}) + \deg^+(i_{\mu_t(v)}) \right), & \text{otherwise.} \end{cases}$$

This result can be interpreted as follows: In order to maintain the invariant (and thereby prove 2-optimality)...

1. ...if the addition of e is not mutual then there can be at most one other edge in $\delta_t(u)$ that becomes tight in a later round; or

2. ...if the addition of e is mutual then there can be at most three other edges in $\delta_t(u) \cup \delta_t(v)$ that become tight in a later round.

These requirements are satisfied by the conditions of the proposition. $\square$

3.2 Efficiency of the Algorithm

Now that we have established the correctness and approximation bounds of the algorithm, this section analyzes its efficiency with respect to the number of communication rounds required for it to reach quiescence.

For sake of analysis, let us first assume that each line in Algorithm 2 (*qq.v.* page 27) can be executed in a constant number of communication rounds; this assumption will be relaxed later in §3.2.2. Note that line 6 of Algorithm 2 contains the only loop of the algorithm whose body requires message passing. Therefore, the total number of communication rounds will be asymptotically equal to the number of iterations of the while loop, which we shall call *primary* communication rounds. Communication rounds that are required for executing the lines of the algorithm within the loop (*e.g.*, intermediate rounds required for a connected component to choose which fringe edge it will make tight) are called *secondary* communication rounds.

3.2.1 Primary Communication Rounds

It is easy to see that the number of primary rounds required for the algorithm to reach quiescence, τ, is equal to the diameter of the search space, which is in turn $O(n)$, since every acyclic subgraph has $O(n)$ edges and the algorithm adds at least one edge per round. This upper bound can in fact be tightened for many common cases, which we shall now demonstrate. Let $A_f(t) = A(t)$ be an upper bound on the number of active components at the beginning of round t. Similarly, let $L_f(t) = L(t)$ be an upper bound on the total number of components at the beginning of round t. Clearly,

$$
\begin{aligned}
A(t) &\geq |\{C \in \mathcal{C}_t : f(C) = 1\}|, \text{ and} \\
L(t) &\geq |\mathcal{C}_t| \geq A(t).
\end{aligned}
$$

In general, every active component will union with another component during each round. Regardless of whether such a component chooses to union with an active or inactive component, the total number of components will decrease by one half the number of active components. Therefore $L(t) = L(t-1) - A(t-1)/2$. Now let us consider the extrema for the change in the number of active components. If all active components choose to union with other active components and all unions are mutual (*q.v.* the definition of mutuality on page 30), then we have $A(t) = A(t-1)/2$. On the other hand, if as many active components union with inactive components as possible, then $A(t) \leq \min\left(A(t-1), L(t-1) - A(t-1)\right)$. Therefore, assuming mutual

unions, the general recurrences for $A(t)$ and $L(t)$ are:

$$A(t) = \max\left(\frac{A(t-1)}{2}, \min\left(A(t-1), L(t-1) - A(t-1)\right)\right), \quad (3.11)$$

$$L(t) = L(t-1) - \frac{A(t-1)}{2}.$$

Let n be the total number of vertices, $n = |V|$, and let $\alpha \le n$ be the number of terminals:

$$\alpha = |\{v \in V : f(\{v\}) = 1\}|. \quad (3.12)$$

The initial conditions for the recurrences are clearly

$$A(0) = |\{C \in \mathcal{C}_0 : f(C) = 1\}| = (3.12) = \alpha,$$
$$L(0) = |\mathcal{C}_0| = n.$$

Claim 2. *$A(t-1)/2$ will always dominate the maximization in* (3.11).

Validation of this claim will be given in the proof of the following proposition.

Proposition 5. *The algorithm will terminate after $\tau = O(\log n)$ iterations of the main loop (line 6 of Algorithm 2) if $\alpha \ge \frac{n}{2}$ and all component unions are mutual.*

Proof. This follows from the fact that the algorithm will terminate once all components are inactive:

$$\forall t \in \mathbb{N}_0 : A(t) = 0 \implies t \ge \tau.$$

Therefore, the burden of this proof is to show that the $A(t)$ recurrence will converge exponentially, implying that $\tau = O(\log n)$.

If Claim 2 holds, then it is clear that the $A(t)$ recurrence will converge exponentially:

$$A(t) = \frac{A(0)}{2^t},$$

$$L(t) = n - \sum_{i=0}^{t} \frac{A(0)}{2^i}.$$

Let $k = \frac{A(0)}{n}$ and observe that the conditions of this proposition ensure $k \ge \frac{1}{2}$. Substituting $k \times n$ for $A(0)$ ensures that the minimization in $A(t)$ will always

evaluate to $L(t-1) - A(t-1)$ because

$$\forall t \in \mathbb{N}_0 : A(t) \geq L(t) - A(t)$$

$$\iff n\frac{k}{2^t} \geq n\left(1 - \left(\sum_{i=0}^{t} \frac{k}{2^i}\right) - \frac{k}{2^t}\right)$$

$$\iff \frac{2k}{2^t} \geq 1 - \sum_{i=0}^{t} \frac{k}{2^i}$$

$$\iff k \geq \frac{2^t}{1 + 2^{t+1}},$$

which is true because $2^t/(1 + 2^{t+1})$ is bounded above by $\frac{1}{2}$. Therefore, provided Claim 2 holds, (3.11) can be simplified to

$$A(t) = \max\left(\frac{A(t-1)}{2}, L(t-1) - A(t-1)\right).$$

Claim 2 obviously holds for the base case of $t = 1$ because $A(0)/2 = k \times n$ is bounded below by $L(0) - A(0) = n - \frac{n}{2}$. Therefore, Claim 2 will hold as long as

$$\frac{A(t)}{2} \geq L(t) - A(t).$$

This equates to

$$k \geq 2^{t+1}\left(1 - \left(\sum_{i=0}^{t} \frac{k}{2^i}\right) - \frac{k}{2^t}\right)$$

$$\geq \frac{2 \times 4^t}{2^t + 4^{t+1}},$$

which must be true because $(2 \times 4^t)/(2^t + 4^{t+1})$ is bounded above by $\frac{1}{2}$. $\quad\square$

3.2.2 Secondary Communication Rounds

In some communications networks—such as a wired local area network (LAN)—the assumption that there is a constant number of secondary communication rounds for each primary round is valid. In fact, it will be valid for any network in which the cost of sending a message between any pair of agents is constant. Routing may be necessary in some networks like MANETs, in which case the cost of sending a message between agents is a function of the number of routing hops that separate them. Analyzing the total number of rounds in such cases is the subject of this section.

If network routing is necessary, the number of messaging rounds required to send a message between any pair of agents in the same connected component is equal to the diameter of the component. Therefore, the number of

rounds required to choose a fringe edge (line 7 of Algorithm 2) would be on the order of the diameter of the active connected component. During primary round t, the diameter of every connected component is $O(t)$. Therefore, the number of secondary rounds between primary round t and primary round $t+1$ will be $O(t)$.

The worst case therefore occurs when the problem instance contains only two terminals, in which case the algorithm will require $O(n)$ primary rounds. The total number of rounds (*viz.*, the sum of both primary and secondary rounds) will therefore be bounded above by

$$\sum_{i=1}^{n} i = O(n^2). \tag{3.13}$$

Despite the fact that this is $O(n^2)$, the summation converges to that upper bound very slowly. To measure the rate of the convergence we shall use a variation of d'Alembert's ratio test. Given a sequence $\{x_k\}$ that converges to a constant L,

$$\lim_{k \to \infty} x_k = L,$$

we take the following limit

$$\lim_{k \to \infty} \frac{|x_{k+1} - L|}{|x_k - L|} = r,$$

and say that r is the *rate of convergence*. If $r = 0$ then the sequence converges to L superlinearly. If $0 < r < 1$ then the sequence converges linearly. If $r = 1$ then the sequence converges sublinearly. Applying this test to the ratio of the the sum in (3.13) and its asymptotic bound of n^2, we see that $L = \frac{1}{2}$:

$$\lim_{n \to \infty} \frac{\frac{1}{2}(n^2 - n)}{n^2} = \frac{1}{2}.$$

Taking the limit of the ratio,

$$\lim_{k \to \infty} \frac{\left| \frac{(k+1)^2 - k - 1}{2(k+1)^2} - \frac{1}{2} \right|}{\left| \frac{k^2 - k}{2k^2} - \frac{1}{2} \right|} = \lim_{k \to \infty} \left| \frac{k}{k+1} \right| = 1,$$

it is clear that the sum in (3.13) converges to its bound of $O(n^2)$ sublinearly with respect to n.

In the best case, the number of terminals in the problem instance is greater than or equal to $\lfloor \frac{n}{2} \rfloor$ and the majority of edge additions are mutual. From Proposition 5 we know that this will result in $O(\log n)$ primary rounds. Therefore, the total number of rounds (primary plus secondary) will be

$$\sum_{i=1}^{\log n} i = \frac{\log(n)\,(\log(n) + 1)}{2} = O(\log^2(n)),$$

which is polylogarithmic. Applying the same rate of convergence test as above:

$$\lim_{n \to \infty} \frac{\frac{1}{2}\left(\log(n)\left(\log(n)+1\right)\right)}{n^2} = L = \frac{1}{2},$$

$$\text{and } \lim_{k \to \infty} \left| \frac{\log^2(k+1) + \log(k+1) - 1}{\log^2(k) + \log(k) - 1} \right| = r = 1.$$

In the best case in terms of the number of terminals, we can likewise conclude that the convergence of the number of rounds to its upper bound of $O\left(\log^2(n)\right)$ is also sublinear.

3.2.3 Time-Approximation Tradeoff

Elkin recently discovered unconditional lower bounds on the time-approximation tradeoff in distributedly solving the minimum spanning tree (MST) problem [80]. Recall from Table 2.1 that the MST problem is a constrained forest problem. In fact, the problem is a matroid and thus our approach is always guaranteed to find the optimal solution. Since the MST problem is in a sense a best case scenario for our approach, Elkin's lower bounds provide a benchmark on how close our approach achieves the bound. Elkin's results are transcribed in Table 3.1.

As we can see from Elkin's results, in the general case when the diameter of the graph, Λ, is equal to n, the time-approximation tradeoff is $\tau^2 \times \varepsilon = \Omega\left(\frac{n}{B}\right)$, where B is the number of bytes sent per message. In the worst case, our algorithm will have to transmit a message with a payload containing all of the graph's edges (*e.g.*, if a fringe vertex is neighboring all other vertices). This message will be of size $O(\log|E|)$ bytes. Given that our algorithm's approximation bound, ε, equals 2, we have:

$$2\tau^2 = \Omega\left(\frac{n}{\log|E|}\right)$$
$$\Downarrow$$
$$\tau = \Omega\left(\sqrt{\frac{n}{\log\binom{n}{2}}}\right)$$
$$= \Omega\left(\log n\right).$$

Therefore, our runtime bound of $\tau = O(\log n)$ (*q.v.* Proposition 5) actually achieves the lower bound.

Extending our notion of subdividing rounds espoused in the previous sections, let us consider further subdividing rounds such that only a fractional

Λ	Lower Bound on the Exact Computation	Lower Bound on the Time-Approximation Tradeoff
$n^d : 0 < d < \frac{1}{2}$	$\Omega\left(\sqrt{\dfrac{n}{B}}\right)$	$\tau^2 \times \varepsilon = \Omega\left(\dfrac{n}{B}\right)$
$\Theta(\log n)$	$\Omega\left(\sqrt{\dfrac{n}{B \log n}}\right)$	$\tau^2 \times \varepsilon = \Omega\left(\dfrac{n}{B \log n}\right)$
Constant (at least 3)	$\Omega\left(\left(\dfrac{n}{B}\right)^{\frac{1}{2} - \frac{1}{2\Lambda-2}}\right)$	$\tau^{2+\frac{2}{\Lambda-2}} \times \varepsilon = \Omega\left(\dfrac{n}{B \times \Lambda}\right)$

Table 3.1: A summary of lower bounds on the MST problem restricted to n-vertex graphs of diameter at most Λ, where B is the number of bytes sent per message and ε is the constant of approximation. This table is excerpted from [80, *cf.* Table 1] and is attributable to Elkin.

amount of dual variable slack is pushed up during each sub-round. Borrowing Elkin's notation (at the expense of overloading our own), let ω_{max} be the ratio between the maximal and the minimal weight of an edge in the input graph [80, *cf.* page 332]. Divide each primary round into $\left\lceil \frac{2}{3}\omega_{max} \right\rceil$ sub-rounds. During each sub-round increase the non-tight dual variables by a value of $\frac{3}{2}$. Therefore, during sub-round $i \in \left\{1, 2, \ldots, \left\lceil \frac{2}{3}\omega_{max} \right\rceil \right\}$, only edges of weight in the range

$$\left(\tilde{\omega} + \frac{3}{2}(i-1), \tilde{\omega} + \frac{3}{2}i \right] \tag{3.14}$$

will become tight. Since $\forall i \in \mathbb{N} : |(3.14)| \leq \frac{3}{2}$, the original 2-optimality invariant from Proposition 3 holds, even if ω_{max} is greater than $\frac{3}{2}$.

As long as there exists some constant c such that $\omega_{max} \leq \frac{3}{2}\log^c(n)$, then the round splitting will only increase the total number of rounds by a factor of $O(\log^c(n))$, which retains polylogarithmic runtime. This in effect gives a tradeoff on runtime: If we have a finite *a priori* upper bound on ω_{max}, then we can perform the round splitting with $\left\lceil \frac{2}{3}\omega_{max} \right\rceil$ sub-rounds. The better the bound on ω_{max} the lower the c parameter, and thereby the fewer sub-divided rounds that are necessary. If a constant c does exist, then the total number of rounds will increase by a factor of

$$O\left(\log^{\left(\frac{\log\left(\frac{2}{3}\omega_{max} \right)}{\log\log n} \right)}(n) \right).$$

3.2.4 Local Efficiency

The only data structures employed in Algorithm 2 are those of $\tilde{V}$, $\tilde{E}$, and F. $\tilde{V}$ can contain at most all of the vertices, so $|\tilde{V}| = o(n)$. $\tilde{E}$ and F can contain at most all of the edges, so $|\tilde{E}| = o(|E|)$ and $|F| = o(|E|)$. Therefore, the maximum memory usage at any node executing the algorithm is

$$O(|E|) = O\left(\frac{n^2 - n}{2} \right).$$

In terms of local computation, we showed in §3.2.1 that the number of iterations of the while loop of the algorithm is asymptotically sublinear with respect to the number of vertices. It is also clear that the operations in lines 2–5 can be implemented in sublinear time. The most expensive local operations in the body of the while loop are lines 7, 9, 14, and 17.

Line 7 In our analysis of memory complexity, we established that

$$|F| = O\left(\frac{n^2 - n}{2} \right),$$

which will thereby also be the complexity of finding the minimum element in F.

Line 9 The most expensive operation on this line is performing the symmetric difference of the searches' fringes for their merger. The symmetric difference of two sets can be performed in asymptotic linear time with respect to the size of the larger set. Since $|F| = O(|E|)$, this operation can likewise be performed in $O\left(\frac{n^2-n}{2}\right)$ time.

Line 14 $|\tilde{V}| \leq |V|$, so this operation will run in $O(n)$ time.

Line 17 u can be incident to at most $|V| - 1$ edges, so this operation will run in $O(n)$ time.

Therefore, the most expensive operation inside of the main while loop runs in $O\left(\frac{n^2-n}{2}\right)$ time. Since the while loop will iterate at most n times, the worst-case local computation of the algorithm is bounded above by

$$O\left(n \times m\right) = O\left(\frac{n^3 - n^2}{2}\right),$$

where m is the number of edges in the search space. This bound is clearly polynomial.

3.3   Conclusions

This chapter has introduced a new distributed multidirectional graph search algorithm for constrained forest problems based on the primal-dual schema. We have shown that each node in the search space only needs to know the status of the fringe of its search in order to make a decision, the locality of which inherently allows for the distribution of the algorithm. In Corollary 1 we showed that the distribution will not induce a cycle in the final solutions, the result of which is strengthened by Proposition 2's assurance that the solutions of the algorithm are feasible. Proposition 3 proves that a constant approximation bound is achievable and, furthermore, that we can sustain that bound for graphs with well behaved edge weights. Finally, in §3.2.3 we showed that our runtime bounds meet the theoretical lower bound, and gave an algorithmic extension for 2-approximating constrained forest problems of arbitrary edge distribution if a finite upper bound on $\omega_{\max}$ is known.

There are, however, a few more issues to address with respect to distributed multidirectional graph search. First, there is the question of how the algorithm performs on graphs in which the edge weights do not map to the metric space required for the theoretical approximation guarantees to hold and there is no *a priori* finite upper bound on $\omega_{\max}$ that would allow

for round splitting. In Chapter 4 it will be shown that—even without the required embedding of edge weights and in the absence of the round splitting technique—the expected value of the approximation bound will be constant for a large family of edge weight distributions. Secondly, no examples have yet been given as to how this framework can be instantiated (*e.g.*, furnished with distributed protocols) for specific problems and their associated proper functions. Finally, it is unclear how this approach might fare empirically in real-world domains. These last two topics are subjects of Chapter 5.

Chapter 4

Bad Things Rarely Happen to Good Graphs

The previous chapter proved that multidirectional graph search can be used to distributedly discover constrained forests. Furthermore, if the edge weights are known to be distributed in a proper metric space then the solutions are guaranteed to be 2-optimal. But what if the distribution of edge weights is unknown and/or does not satisfy the requirements for the theoretical 2-optimality bound to hold? This chapter answers the previous question by investigating the expected value of the worst case behavior of constrained forest algorithms. In order to do this, we shall first need to introduce the problem using statistical notation. On how this statistical problem relates to the previous approximation algorithms problem will then be expounded.

Let $\mathbf{X} = [X_1, X_2, \ldots, X_n]^{\mathrm{T}}$ be a vector random variable drawn from a known non-negative[1] distribution with cumulative distribution function $F(x)$ and probability density function $f(x)$. Let $X_{(n,k)}$ be the distribution of the kth order statistic of $\mathbf{X}$ (*i.e.*, the kth smallest element of the vector). Given $m, \ell \in \{1, 2, \ldots, n\}$, let Z be the ratio distribution defined by the sum of the ℓ largest order statistics divided by the m smallest order statistics:

$$Z = \frac{\displaystyle\sum_{i=n-\ell+1}^{n} X_{(n,i)}}{\displaystyle\sum_{k=1}^{m} X_{(n,k)}}.$$

What can be said about Z? What are its probability density function (PDF), cumulative distribution function (CDF), and expected value?

---

[1] By "non-negative" we mean that the distribution is truncated in the range $[0, \infty)$ such that $F(0) = 0$.

Before proceeding in addressing these questions, let us first discuss why this problem is relevant to approximation algorithms. Consider a combinatorial optimization problem on a finite structure, such as a graph. We want to find a subset of the edges of the graph that minimize some objective function subject to a set of constraints. In this context, $\mathbf{X}$ can be thought of as the set of edge weights, assuming the edge weights are independent and identically distributed random variables drawn from the distribution $F(x)$. Now let us assume that we have a lower bound on the number of edges in the optimal solution. This bound is m. For example, if we are trying to find a minimum spanning tree and we know that the graph is connected, then the optimal solution will have a number of edges exactly equal to the number of vertices minus one. As another example, in the Steiner network problem with α terminals, m must be greater than or equal to $\left\lfloor \frac{\alpha}{2} \right\rfloor$. Therefore, the cost of the optimal solution will have a probability distribution bounded below by the sum of the m smallest order statistics of $\mathbf{X}$. This is the denominator of Z. Now consider ℓ as an upper bound on the number of edges chosen by our algorithm. If our algorithm is finding a forest, then ℓ is clearly bounded above by the number of vertices minus one. How bad would it be if our algorithm had the *worst* behavior: choosing a solution containing the ℓ heaviest edges? The cost of this solution would be equal to the sum of the ℓ largest order statistics of $\mathbf{X}$, which constitutes the nominator of Z. Therefore, Z is a very loose lower bound on the probability distribution of the bound of approximation of this worst case algorithm. The faster the CDF of Z converges to 1 then the lower the expected approximation bound of the algorithm.

In the remainder of this chapter we analyze the asymptotic behavior of Z for a number of common probability distributions of X. Using the central limit theorem, we generalize these results by discovering that, for large ℓ and m, the expected value of Z is $\frac{\ell}{m}$, regardless of the distribution of X. We conclude by examining some surprising consequences of this general result.

4.1 Distributions of Trimmed Sums

In order to define the probability distribution of Z we will need to know the distribution of the sum of consecutive order statistics (what are also often called "trimmed sums"). The problem is that there is no known generalized closed form for this distribution. We can, however, devise a recursive definition.

Csörgő and Simons discovered a recurrence relation for the CDF and PDF of the distribution of the sum of the m smallest order statistics of non-negative *integer-valued* random variables [81]. In the following, we generalize their approach by devising a recurrence relation for the PDF that is applicable to *any* non-negative distribution (not just integer-valued ones).

We are interested in finding the probability that the sum of the m smallest

order statistics equals a given value: $P\left[\sum_{i=1}^{m} X_{(n,i)} = x\right]$. The recurrence is quite simple: The probability that the sum of the m smallest order statistics equals x is equal to the integral over all possible values of the mth smallest order statistic multiplied by the probability that the smaller $m-1$ order statistics sum to the remaining value in x. The PDF for a specific order statistic (encompassing the base case of $m = 1$) is well defined for most distributions. Therefore, the probability that the smallest m order statistics of a sample of size n sums to x is

$$P\left[\sum_{i=1}^{m} X_{(n,i)} = x\right] =$$

$$\int_0^x P[X_{(n,m)} = s_1] \int_0^{x-s_1} P[X_{(n,m-1)} = s_2] \int_0^{x-s_1-s_2} P[X_{(n,m-3)} = s_3]$$

$$\cdots \int_0^{x-\sum_{i=1}^{m-1} s_i} P[X_{(n,1)} = s_m]\, ds_m \cdots ds_3\, ds_2\, ds_1,$$

which reduces to the following recurrence:

$$P\left[\sum_{i=1}^{m} X_{(n,i)} = x\right] =$$

$$\begin{cases} P\left[X_{(n,1)} = x\right] & \text{if } m = 1, \\ \int_0^x P\left[X_{(n,m)} = s\right] P\left[\sum_{i=1}^{m-1} X_{(n,i)} = x - s\right] ds & \text{otherwise.} \end{cases} \tag{4.1}$$

A similar recurrence follows for the sum of the ℓ largest order statistics:

$$P\left[\sum_{i=n-\ell+1}^{n} X_{(n,i)} = x\right] =$$

$$\begin{cases} P\left[X_{(n,n)} = x\right] & \text{if } \ell = 1, \\ \int_0^x P\left[X_{(n,\ell)} = s\right] P\left[\sum_{i=n-\ell+2}^{n} X_{(n,i)} = x - s\right] ds & \text{otherwise.} \end{cases} \tag{4.2}$$

While there does not appear to be a general closed form for this recurrence, it is possible to analyze for specific distributions.

First, let us consider the continuous uniform distribution on the range $[a, b]$ where $0 \leq a \leq b$. Call a random variable sampled from this distribution $U(a, b)$. It is well known that the PDF for the kth order statistic for a sample of size n of the continuous uniform distribution falls under a β-distribution with parameters k and $n - k + 1$:

$$P\left[U(a, b)_{(n,k)} = x\right] = \frac{x^{k-1}(1 - x)^{n-k}\Gamma(n + 1)}{\Gamma(k)\Gamma(n - k + 1)}. \tag{4.3}$$

Figure 4.1: CDF for the distribution of the sum of the m smallest order statistics of a sample of size 10 from the standard uniform distribution.

Remarkably, this PDF is dependent on neither a nor b. This implies that for any $0 \le a \le b$ and $0 \le c \le d$ then

$$\sum_{i=1}^{m} U(a,b)_{(n,i)} \; \overset{d}{=} \; \sum_{j=1}^{m} U(c,d)_{(n,j)}. \tag{4.4}$$

Therefore, we consider the standard uniform distribution $U(0,1)$, the results of which can be generalized to any other continuous uniform distribution by virtue of (4.4). Plugging (4.3) into the recurrence of (4.1) allows us to numerically evaluate the trimmed sum of order statistics. The CDF for the sum of order statistics of $U(0,1)$ on a sample of size 10 is given in Figures 4.1 and 4.2. It is clear that the CDF converges to 1 with a very high gradient, especially for small m. This implies that, for uniform distributions, the distribution of the first moment (*i.e.*, expected value) of the sum of the m smallest order statistics is relatively invariant.

By the central limit theorem, for large enough m and ℓ the distribution of the sum of order statistics will fall under a normal distribution, regardless of the underlying distribution of X. The sum of the smallest m order statistics will asymptotically have the mean $m\,\mathrm{E}[X]$ and variance $\mathrm{Var}(X)m^{-1}$.
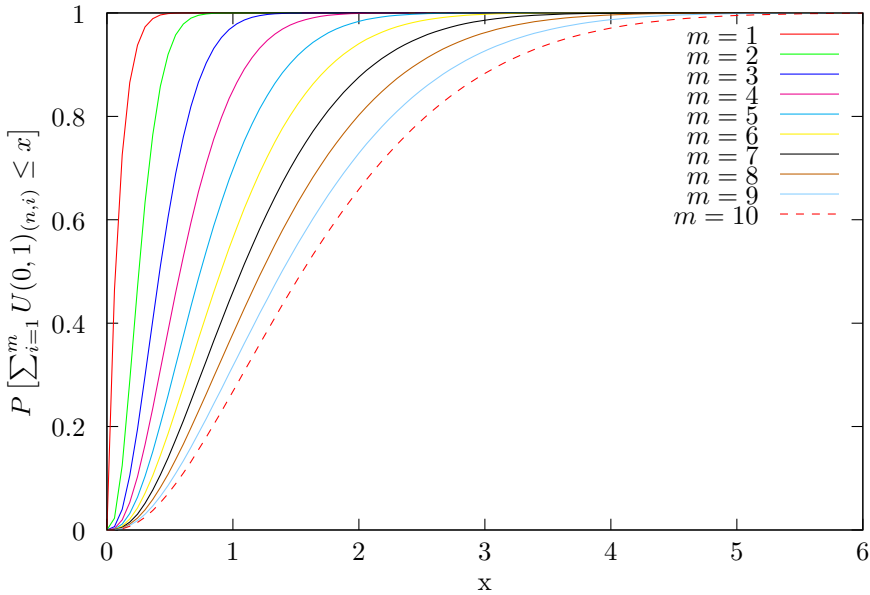
Figure 4.2: CDF for the distribution of the sum of the ℓ largest order statistics of a sample of size 10 from the standard uniform distribution.

Likewise, the largest ℓ order statistics will have mean $\ell \, \mathrm{E}[X]$ and variance $\mathrm{Var}(X)\ell^{-1}$. For the continuous uniform distribution, this asymptotically translates to the following normal distributions:

$$P\left[\sum_{i=1}^{m} U(a,b)_{(n,i)} \leq x\right] \stackrel{d}{=} \mathcal{N}\left(\frac{m(a+b)}{2}, \frac{(b-a)^2}{12m}\right) \qquad (4.5)$$

$$P\left[\sum_{i=n-\ell+1}^{n} U(a,b)_{(n,i)} \leq x\right] \stackrel{d}{=} \mathcal{N}\left(\frac{\ell(a+b)}{2}, \frac{(b-a)^2}{12\ell}\right). \qquad (4.6)$$

Therefore, the expected value of the sum of the order statistics of every continuous uniform distribution has very low variance. This means that for continuous uniform distributions the quotient of the expected values of the nominator and denominator in Z should give a relatively unbiased estimation of the true expected value of Z, which itself should have very low variance.

4.2 The Exponential Distribution

In many respects, the exponential distribution is the worst case as its memoryless property ensures the Z distribution will have unbounded outliers. This distribution is of specific import given its manifestation in many real-world systems. We therefore want to examine how quickly the exponential distribution's trimmed sums' CDF converges to 1.

Nagaraja recently showed that the sum of the ℓ largest order statistics of a random variable X taken from the standard exponential distribution has the following PDF [82]:

$$P\left[\sum_{i=n-\ell+1}^{n} X_{(n,i)} = x\right] =$$

$$n\binom{n-1}{\ell-2}\sum_{i=1}^{\ell-1}\binom{\ell-2}{i-1}(-1)^{\ell-i-1}\exp\left(-\frac{n-i+1}{n-\ell+1}x\right)$$

$$\times \frac{1}{(n-\ell+1)!}\int_0^x \exp\left(\frac{\ell-i}{n-\ell+1}j\right)j^{n-\ell}dj,$$

the expected value of which is

$$\mathrm{E}\left[\sum_{i=n-\ell+1}^{n} X_{(n,i)}\right] = \sum_{i=n-\ell+1}^{n}\sum_{j=1}^{i}\frac{1}{n-j+1}.$$

We can see from Figure 4.3 that this distribution also converges at a very high gradient.
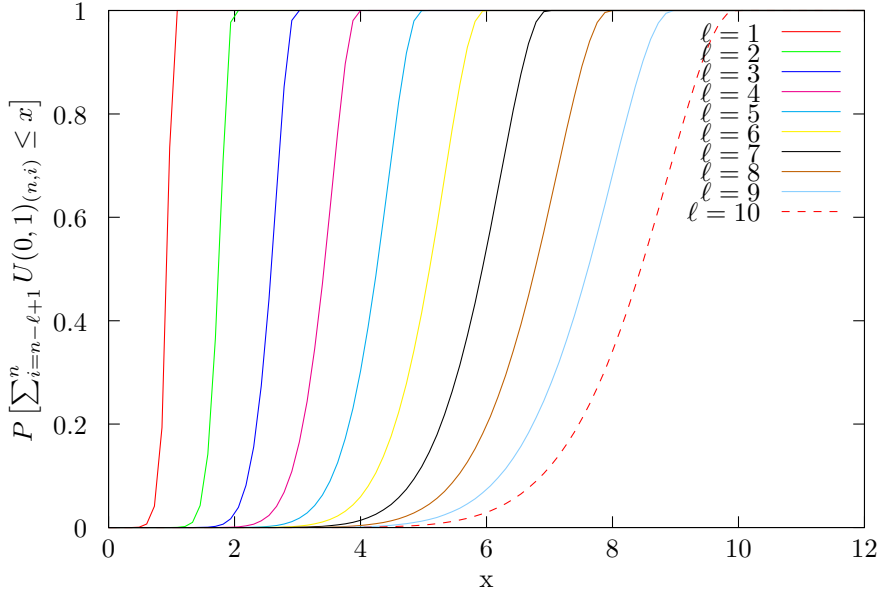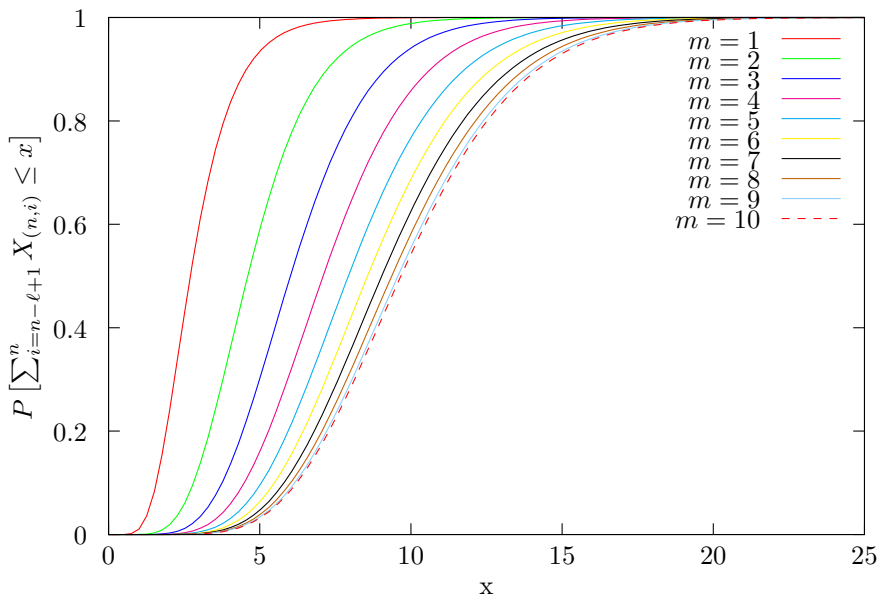
Figure 4.3: CDF for the distribution of the sum of the ℓ largest order statistics of a sample of size 10 from the standard exponential distribution.

Therefore, for the standard exponential distribution with a sample of size n, the expected value for Z is

$$
\begin{aligned}
E[Z] &= \frac{\mathrm{E}\left[\sum_{i=n-\ell+1}^{n} X_{(n,i)}\right]}{\mathrm{E}\left[\sum_{i=1}^{n} X_{(n,i)}\right] - \mathrm{E}\left[\sum_{i=m+1}^{n} X_{(n,i)}\right]} \\[2mm]
&= \frac{\sum_{i=n-\ell+1}^{n} \sum_{j=1}^{i} \frac{1}{n-j+1}}{\sum_{i=1}^{n} \sum_{j=1}^{i} \frac{1}{n-j+1} - \sum_{i=m+1}^{n} \sum_{j=1}^{i} \frac{1}{n-j+1}} \\[2mm]
&= \frac{\sum_{i=n-\ell+1}^{n} \sum_{j=1}^{i} \frac{1}{n-j+1}}{\sum_{i=1}^{m} \sum_{j=1}^{i} \frac{1}{n-j+1}} \\[2mm]
&= \frac{\sum_{i=1}^{\ell} \sum_{j=1}^{n-\ell+i} \frac{1}{n-j+1}}{\sum_{i=1}^{m} \sum_{j=1}^{i} \frac{1}{n-j+1}} \\[2mm]
&= \frac{\sum_{i=1}^{\ell} \Psi(-n) - \Psi(i-\ell)}{\sum_{j=1}^{m} \Psi(-n) - \Psi(j-n)} \\[2mm]
&= \frac{\ell}{m}\Psi(-n) - \frac{\sum_{i=1}^{\ell} \Psi(i-\ell)}{\sum_{j=1}^{m} \Psi(j-n)} \\[2mm]
&= \frac{\ell}{m} H_{-n-1} - \frac{\sum_{i=1}^{\ell} H_{i-\ell-1}}{\sum_{j=1}^{m} H_{j-n-1}}, \tag{4.7}
\end{aligned}
$$

where H_n is the nth harmonic number and Ψ is the digamma function:

$$
\Psi(x) = \frac{d}{dx} \log \Gamma(x).
$$

The summations in (4.7) can be further reduced to the rather cumbersome closed form of

$$
\frac{\ell \sin(\pi n) \sin(\pi \ell) \Big(\Psi(\ell) n - \Psi(n) n - n - 1 \Big)}{nm \sin(\pi n) \sin(\pi \ell) \Big(\Psi(m-n) - \Psi(-n) \Big)}
$$

$$
+ \frac{\ell \pi n \Big(\sin(\pi n) \cos(\pi \ell) - \cos(\pi n) \sin(\pi \ell) \Big)}{nm \sin(\pi n) \sin(\pi \ell) \Big(\Psi(m-n) - \Psi(-n) \Big)}.
$$

Therefore, the trimmed sum of order statistics of the exponential distribution is well behaved, similar to the continuous uniform distribution.

4.3 Normal Distributions

Discovering expressions for the distributions of trimmed sums of normally distributed random variables is still an open problem. We can, however,

Figure 4.4: CDF for the distribution of the sum of the m smallest order statistics of a sample of size 10 from the standard normal distribution, calculated from a Monte Carlo simulation.

plot such distributions using Monte Carlo simulation. First, a vector of size 10 was randomly populated with variates drawn from the standard normal distribution $\mathcal{N}(\mu = 0, \sigma = 1)$. The vector is then sorted and the sums of the m smallest and ℓ largest elements is taken. This process was repeated 32000 times, the results of which are given in Figures 4.4 and 4.5.

As we can see, the normal distribution exhibits the same quick convergence as the uniform and exponential distributions, albeit skewed by the fact that the normal distribution is not positive. Therefore, let us consider the standard normal distribution truncated on the interval $[0, 1]$:

$$P\left[\mathcal{N}_{[0,1]}(0, 1) \leq x\right] =$$

$$\frac{\int_0^x P\left[\mathcal{N}(0, 1) = t\right] dt}{P\left[\mathcal{N}(0, 1) = 1\right] - P\left[\mathcal{N}(0, 1) \leq 0\right]} = \begin{cases} 0, & x < 0 \\ 1, & x > 1 \\ \dfrac{\operatorname{erf}\left(\frac{x\sqrt{2}}{2}\right)}{\operatorname{erf}\left(\frac{\sqrt{2}}{2}\right)}, & \text{otherwise.} \end{cases} \quad (4.8)$$

Since (4.8) is continuous and monotone, the Newton-Raphson method can

Figure 4.5: CDF for the distribution of the sum of the ℓ largest order statistics of a sample of size 10 from the standard normal distribution, calculated from a Monte Carlo simulation.

Figure 4.6: CDF for the distribution of the sum of the m smallest order statistics of a sample of size 10 from the standard normal distribution truncated in the range $[0, 1]$, calculated from a Monte Carlo simulation.

be used to quickly calculate the inverse CDF to arbitrary precision. This, in turn, allows for production of a random variate for the truncated standard normal distribution using inverse transform sampling. The variate can then be used for Monte Carlo simulation, as described above. The results for the truncated standard normal distribution are given in Figures 4.6 and 4.7, and exhibit a similarly fast convergence.

4.4 The Expected Value of Z

We have thus far established that the expected value of the sum of the order statistics of every continuous uniform, standard exponential, standard normal, and truncated standard normal distribution has very low variance. In fact, by the central limit theorem,

$$\lim_{n \to \infty} \max_{m=1\ldots n} \mathrm{Var}\left(\mathrm{E}\left[\sum_{k=1}^{m} X_{(n,k)} \right] \right) = \lim_{n \to \infty} \mathrm{Var}\left(\mathrm{E}\left[\sum_{k=1}^{m} X_{(n,1)} \right] \right) = \frac{1}{12}.$$
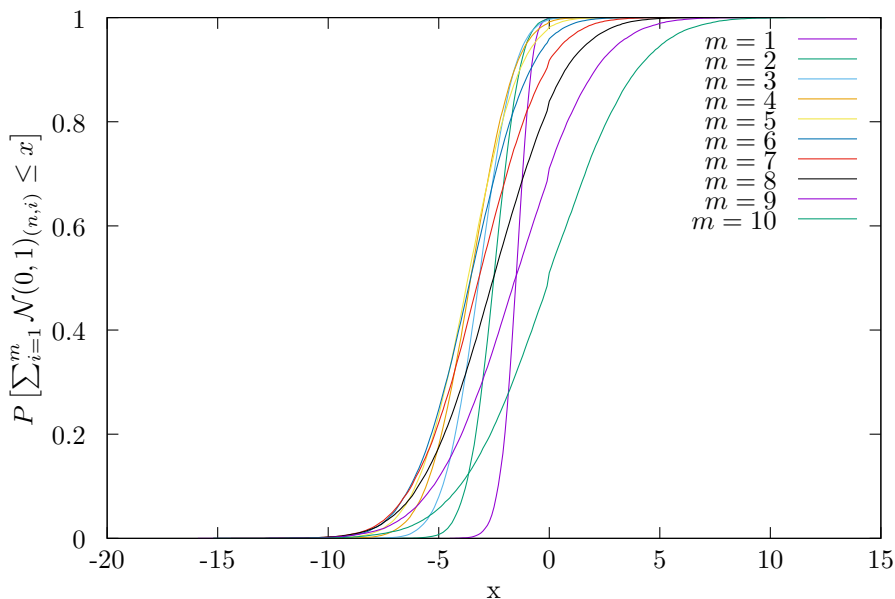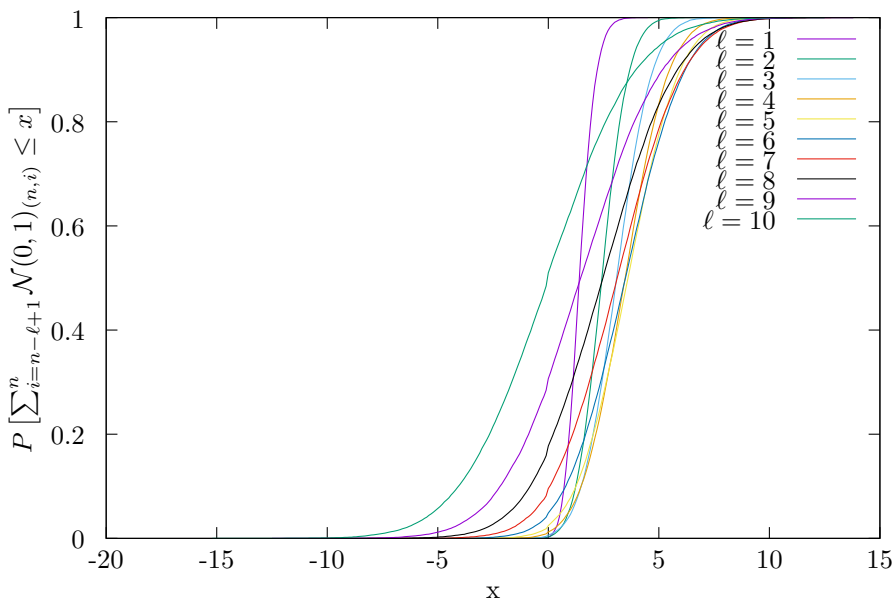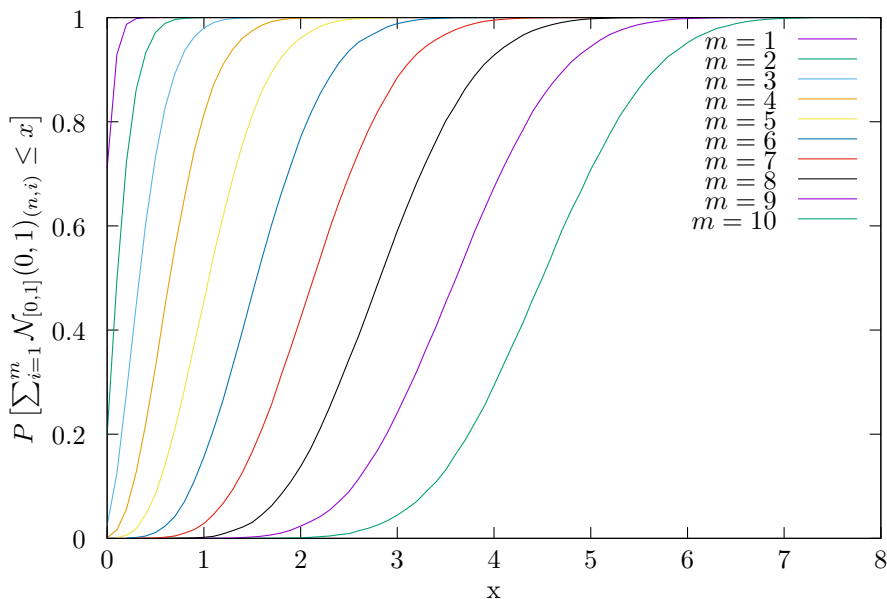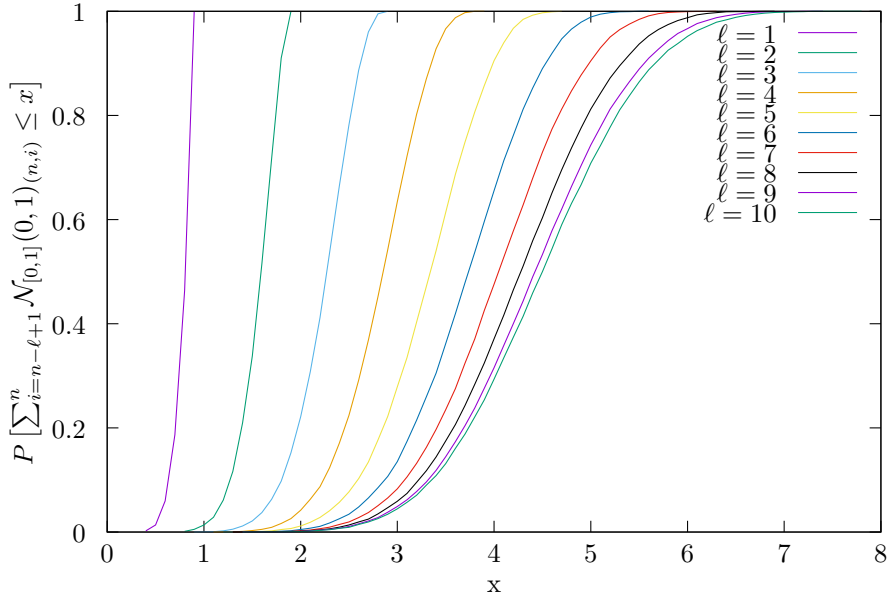
Figure 4.7: CDF for the distribution of the sum of the ℓ largest order statistics of a sample of size 10 from the truncated standard normal distribution truncated in the range $[0, 1]$, calculated from a Monte Carlo simulation.

Therefore, the quotient of the expected values of the nominator and denominator in Z should give a relatively unbiased estimation of the true expected value of Z, which itself should have very low variance.

By (4.5) and (4.6), for large enough m and ℓ the expected value for Z will be

$$\mathrm{E}[Z] = \frac{\frac{1}{2}\ell(a+b)}{\frac{1}{2}m(a+b)} = \frac{\ell}{m}.$$

In the context of the approximation algorithm example, if the edge weights are drawn from any uniform distribution then, for large enough m, any sufficiently large, randomly chosen subset of ℓ edges will with high probability be $\frac{\ell}{m}$ times optimal, regardless of the objective function. Furthermore, if $\ell = O(m)$ then the solution is with high probability a constant factor of optimal.

In fact, this property holds in general:

$$P\left[\sum_{i=1}^{m} X_{(n,i)} \leq x\right] \overset{d}{=} \mathcal{N}\left(m\,\mathrm{E}[X], \frac{\mathrm{Var}(X)}{m}\right)$$

$$P\left[\sum_{i=n-\ell+1}^{n} X_{(n,i)} \leq x\right] \overset{d}{=} \mathcal{N}\left(\ell\,\mathrm{E}[X], \frac{\mathrm{Var}(X)}{\ell}\right),$$

and

$$\mathrm{E}[Z] = \frac{\ell\,\mathrm{E}[X]}{m\,\mathrm{E}[X]} = \frac{\ell}{m}. \tag{4.9}$$

This is a rather surprising result. This means that if we know the size of the optimal solution is bounded below by m, then *any* randomly chosen solution of size at most ℓ will, on average, be $\frac{\ell}{m}$ times optimal. Consider the Steiner network problem as an example. If there are α terminals, then we know that the optimal solution must have at least $\left\lfloor \frac{\alpha}{2} \right\rfloor$ edges. Any feasible solution to the problem is going to be an acyclic graph, which will have at most $n-1$ edges. Therefore, *any* randomly chosen feasible solution to the Steiner network problem will be, on average, $\frac{2n-2}{\alpha}$ times optimal. If every vertex is a terminal, then any randomly chosen feasible solution is with high probability 2-Optimal. Equation (4.9) has even stronger implications for problems like minimum spanning tree. In that case, we know that (assuming the graph is connected) the optimal solution has exactly $n-1$ edges, and any feasible solution will also have exactly $n-1$ edges. Therefore, any randomly chosen feasible solution to the minimum spanning tree problem will with high probability be a constant factor of optimal.

In conclusion, this chapter has motivated the fact that—even if the conditions of the theoretical guarantees of 2-optimality (*qq.v.* Propositions 3 and 4) are not met—the solutions produced by the distributed multiagent graph search algorithm are with high probability 2-optimal.

Chapter 5

Solving Constrained Forest Problems

Given a subset of the vertices of a graph, called *terminals*, the *constrained forest problem* asks to find a minimum weight forest spanning the terminals subject to a set of topological requirements. These requirements are often represented using *proper functions* (*qq.v.* §2.3). Many constrained forest problems are **NP**-HARD. There were a number of groundbreaking results in the 1990s which culminated in a very elegant and efficient centralized approximation algorithm for constrained forest problems using the primal-dual schema [60, 73]. This opened the door for solving many constrained forest problems in the sequential computation model.

In this chapter, we show that our generalized distributed constrained forest algorithm based on multidirectional graph search can likewise be applied to a number of constrained forest problems. First, we continue our analysis of Steiner network problems by empirically testing how close the algorithm comes to the theoretical bounds described in Chapter 3. Later in this chapter we give two additional examples of our framework's application: the location design and routing problem and art gallery problems. For each we give examples of the protocols that can be employed to fully distribute the algorithm, along with more precise analysis of the number of rounds and quality of the solution. Finally, we give empirical results for a number of cases in which the problem and/or domain does not necessarily satisfy the conditions required for the aforementioned theoretical bounds to hold.
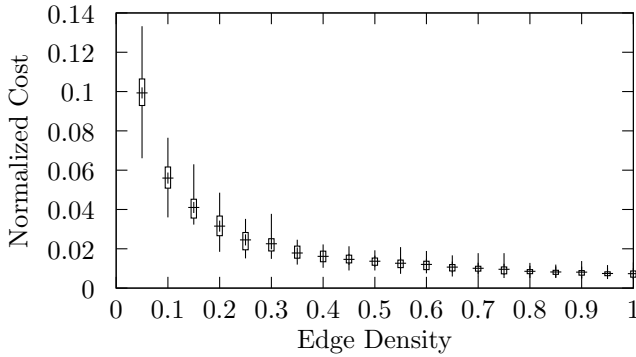
Figure 5.1: Normalized cost of the solutions discovered for 200 node random graphs with 200 terminals and varying edge density. Each data point is the distribution over 32 random graphs. Boxes surround the middle two quartiles. The mean of each distribution is depicted as "+". A normalized cost of 0.0 means that the optimal solution was discovered, whereas a cost of 1.0 means that the costliest possible solution was discovered.

5.1 Steiner Network Problems

In order to test the average case performance of the generalized distributed constrained forest algorithm, a series of graphs on 200 vertices of varying edge density were randomly generated using the Erdős-Rényi model $G(n, p)$ with $n = 200$ vertices and an edge density $p \in [0.05, 1.0]$). For each edge density p, 32 random graphs were generated, the edges of which were weighted according to a uniform distribution. The algorithm was then run with best case concurrency (*i.e.*, one agent per vertex). The optimality results for these experiments are given in Figure 5.1. This can be interpreted as implying that the algorithm produces solutions that are very close to the optimal solution, especially when the search space has a high branching factor. Given that the percentage of terminals was very high, the number of messaging rounds required for the algorithm to reach quiescence was only 3.93 with a standard deviation of 0.70.

Next, we investigated the algorithm's worst case performance: when the edge density (*i.e.*, branching factor) is low and the number of agents is limited. We therefore fixed the edge density at 0.1 and re-ran the experiments with a varying number of terminals. The results of these experiments are in Figure 5.2. We see that even in the absolute worst case (*i.e.*, with a very sparse graph and few agents to perform the concurrent search) the number of messaging rounds required by the algorithm is only about 12, and this quickly converges to the previous best case of about four rounds as the number of
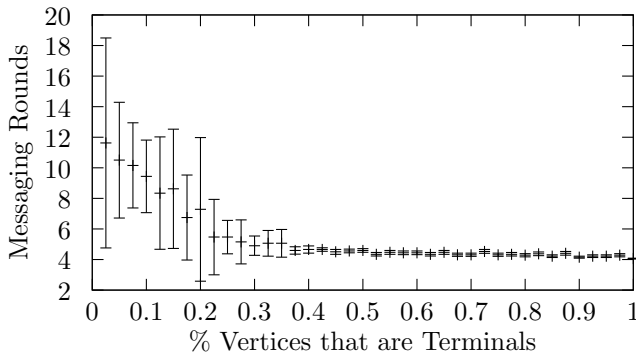
Figure 5.2: Average number of messaging rounds required for the algorithm to reach quiescence for 200 node random graphs with an edge density of 0.1. Error bars represent variance.

terminals increases.

The intuition behind the empirical runtime results is that the algorithm will require approximately as many rounds as the diameter of the intersection graph (*q.v.* footnote 1 on page 31) of G formed from the set of terminals. In the worst case this is clearly $O(n)$, however, if the number of terminals is high then the diameter is likely to be very small. Furthermore, if the graph is known to have scale-free properties (which are very common in many relevant domains), then the diameter of G will be logarithmic, thus the algorithm will run in $O(\log n)$ time.

Finally, we investigated the average case approximation bounds of the algorithm. The number of vertices was set at 20, being a size small enough such that calculation of the optimal solution is tractable on modern hardware. We could then compare the approximated solution of the proposed algorithm to the cost of optimal. These results are given in Figure 5.3. It is clear from the results that the average case approximation bound is actually better than the theoretical upper bound of 2; on average the algorithm produces solutions about 1.3 times the cost of optimal.

5.2 Location Design & Vehicle Routing Problems

Recall from §1.1.1 that the Location Design and Routing problem asks to find a subset of "depot" nodes and a spanning forest of a graph such that every connected component in the forest contains at least one depot [3].

The sequential variant of this problem has been thoroughly studied in the
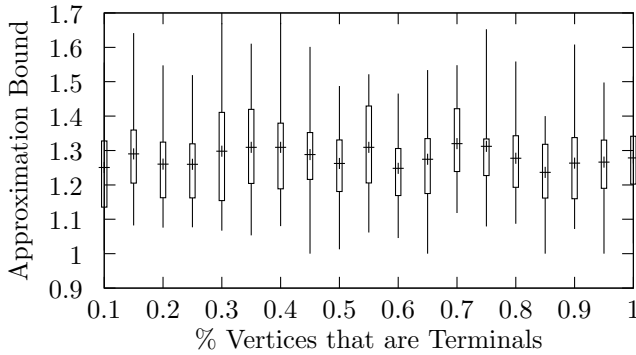
Figure 5.3: Solution quality of the algorithm for a number of random graphs on 20 vertices with varying number of terminals. The *y*-axis is the performance guarantee of the approximation (a value of 1.0 means the optimal solution was found). Each column is the distribution over 32 random graphs, depicted using a similar scheme as that of Figure 5.1.

literature [3, 73, 5, 7], culminating in the discovery that the problem submits to bounded approximation in polynomial time. There are therefore three primary motivations for developing a distributed approximated solution to the problem:

1. the problem itself is naturally distributed—there may not be an obvious central node in which to perform the optimization;

2. local properties of the problem seem to allow for speedups from distributed processing; and

3. in certain environments, such as sensor networks, hardware restrictions might necessitate decentralization in order to save memory.

Although there is very little in the literature on the parallelization and distribution of this specific problem, the related problem of finding a minimum spanning forest has been widely studied and is known to be soluble in logarithmic time with a linear number of processors [83, 84]. In fact, finding a minimum spanning forest is a special case of the location design and routing problem in which depot opening costs are very large (*viz.*, greater than the diameter of the graph). The converse, however, is not true: There is no known trivial reduction from the location design and routing problem to the spanning forest problem.

In the remainder of this section we introduce our algorithm for solving the distributed location design and routing problem. In §5.2.1 formalizes

the problem such that it can be mapped to an equivalent constrained forest problem and §5.2.2 then introduces a parallel version of the algorithm. Next, §5.2.3 proves a series of propositions about the algorithm, including its correctness, completeness, and approximation & runtime bounds. Finally, in §5.2.4, we show that the parallel algorithm can be distributed in an asynchronous network.

5.2.1   Problem Formalization

Given a graph $G = \langle V, E \rangle$ with both vertex and edge weights $w : V \cup E \to \mathbb{Q}$, the *Location Design and Routing Problem* asks to find a subset of "depot" vertices $D \subseteq V$ and a spanning forest $F \subseteq E$ such that each connected component in F contains at least one depot. The cost of opening a depot at a vertex is modeled using the vertex weights; for example, the cost of opening a depot at vertex $v \in V$ is $w(v)$. Therefore, we want to minimize the weight of D and F:

$$\text{minimize} \ \left(\sum_{d \in D} w(d) \right) + \left(\sum_{e \in F} w(e) \right)$$

$$\text{subject to:}$$

$$\left(\exists d \in D : v \text{ is connected to } d \text{ in F} \right), \quad \forall v \in V$$

$$D \subseteq V,$$

$$F \subseteq E.$$

For sake of analysis, the representation of this optimization problem can be simplified by recasting it as a variation of a constrained forest. Therefore, let us augment the graph with special depot vertices d_i associated with each original vertex v_i. Next, add an edge from each vertex to its associated depot vertex, weighted with the cost of opening a depot at that vertex. The new overall set of edges is the original set of edges unioned with the set of new depot edges. Let $R = \{v_1, \ldots, v_n\}$ be the set of original (input) vertices and let $T = \{d_1, \ldots, d_n\}$ be the set of new special depot vertices with the new overall set of vertices $V = R \cup T$. A vertex v_i will be a part of D (*i.e.*, it will be chosen to become a depot) if the edge from it to its associated depot vertex is a part of the final spanning forest: $\langle v_i, d_i \rangle \in F \implies v_i \in D$. The optimization problem can now be rewritten such that it amounts to finding an acyclic subset of the new edges that connects each $v \in R$ to at least one $d \in T$. Since this subset is a forest that spans R we will hereafter refer to it as "the spanning forest". Note, however, that the spanning forest does not necessarily span all of T. An example of this augmented graph is given in Figure 5.4a.

Assume that $w(e)$ now denotes the edge weights. The new optimization problem on the augmented graph can be captured as the following integer

(a) Augmented distribution network.

(b) The optimal solution.

Figure 5.4: (a) gives the augmented graph corresponding to the distribution network in Figure 1.2a with a uniform depot opening cost of 100. Original vertices are ● and the special depot vertices are ○. (b) is the optimal solution to the augmented network, corresponding to the solution in Figure 1.2b.

program:

$$\text{minimize } \sum_{e \in E} w(e) x_e$$

subject to: $\qquad\qquad\qquad\qquad\qquad\qquad$ (LDRP-IP)

$$x(\delta(S)) \geq |S \cap T|, \quad \forall S \subset V : S \neq \emptyset$$
$$x_e \in \{0,1\}, \qquad\qquad \forall e \in E,$$

where each variable x_e is an indicator as to whether the edge e is a member of the final spanning forest, $\delta(S)$ is the set of edges having exactly one endpoint in S, and $x(F) \mapsto \sum_{e \in F} x_e$. Therefore, any forest $F \subseteq E$ will be a feasible solution to the problem if every connected component S of the forest has at least one depot. Let (LP) denote the linear programming relaxation of (LDRP-IP) obtained by relaxing the integrality restriction on the variables to $x_e \geq 0$. The dual of (LP) is

$$\text{maximize } \sum_{S \subset V} \chi \left(\overline{S \cap T} \right) y_S$$

subject to: $\qquad\qquad\qquad\qquad\qquad\qquad$ (LDRP-D)

$$\sum_{S : e \in \delta(S)} y_S \leq w(e), \qquad\qquad \forall e \in E$$
$$y_S \geq 0, \qquad\qquad \forall S \subset V : S \neq \emptyset,$$

where $\chi : X \to \{0,1\}$ is an indicator function: $\chi(X) \mapsto 1 \iff X \neq \emptyset$. An edge is *tight* if $w(e) = \sum_{S : e \in \delta(S)} y_S$. Let $Z^*_{\text{LDRP-LP}}$ be the cost of the

optimal solution to (LP) and let $Z^*_{\text{LDRP-IP}}$ be the cost of the optimal solution to (LDRP-IP). It is a folklore result that $Z^*_{\text{LDRP-LP}} \leq Z^*_{\text{LDRP-IP}}$.

5.2.2 Parallel Computation Model

The basic mechanism of the algorithm follows that of constrained multidirectional graph search. We start off with an empty forest; each vertex is a member of its own connected component. Every round, each depot-less component greedily chooses to add one of its cut edges to the forest, merging with the component on the other end of the edge. When a component merges with another component containing a depot, the new component after the union stops actively growing. When all components contain depots the algorithm terminates.

For simplicity, the algorithm is first introduced as a parallel algorithm according to the concurrent read exclusive write (CREW) parallel random access machine (PRAM) model. The remainder of this section provides the notation and mathematics required to formally define and model the algorithm. This will later be used in §5.2.3 to provide formal bounds on the runtime and performance of the algorithm, and also to prove correctness and completeness. We will later show in §5.2.4 that this same algorithm can be distributed in an asynchronous network.

Let F_t be the partially constructed spanning forest at the beginning of round t. Let $\mathcal{C}_t$ be the set of connected components in F_t. For sake of brevity and simplicity, let $\mu_t : V \to \mathcal{C}_t$ be a function mapping vertices to their associated connected component during round t; therefore, $\mu_t(v) \mapsto C_i \implies v \in C_i \in \mathcal{C}_t$. A vertex that is incident to at least one edge in the cut of its connected component is said to be in the *fringe*. Let $g_t : V \to \mathbb{R}$ be a mapping of vertices to a real number during round t. These values represent the amount of slack remaining in the dual variables associated with a vertex. An example of this notation is given in Figure 5.5.

As it was the case earlier in Chapter 3, let $J_t : V \times V \to \{0, 1\}$ be a binary relation defining which edges will become tight during round t. Each depot-less component will choose to add the edge in its fringe that has minimal weight and dual variable slack. Therefore, $J_t(u, v) = 1$ if and only if

$$\Big(\mu_t(u) \cap T = \emptyset \Big) \wedge \left(\langle u, v \rangle = \operatorname*{arg\,min}_{\langle i,j \rangle \in \delta(\mu_t(u))} w(\langle i, j \rangle) - g_t(i) - g_t(j) \right). \quad (5.1)$$

We assume that each edge has a unique identifier over which there is a total ordering. Ties in the minimization are broken based upon the ordering of the edges. Let J^+ denote the transitive closure of J. Note that J does not necessarily commute: $J(u, v) \not\Longrightarrow J(v, u)$. Also note that as long as there exists a feasible solution to (LDRP-IP) then the minimization ensures that

each depot-less connected component must have exactly one edge in the fringe that becomes tight each round:

$$\forall C \in \mathcal{C}_t, \ \exists \langle v, u \rangle \in \delta(C) : C \cap T = \emptyset \implies J_t(v, u) = 1.$$

We denote by F_t the partially constructed spanning forest during round t, initialized to $F_0 = \langle V, \emptyset \rangle$. The forest is updated each round with the set of all edges that became tight during the round:

$$F_{t+1} = F_t \cup \{\langle u, v \rangle \in E : J_t(u, v) \vee J_t(v, u)\}.$$

For a set $S \subseteq V$, let y_S be the dual variable associated with S. Initially all such variables are set to zero. Note that in actuality these variables need not be made part of an implementation of the algorithm; they exist solely for the purpose of theoretically proving properties of the algorithm [73]. These dual variables are implicitly updated as follows:

$$y_S \leftarrow \begin{cases} \frac{w(\langle i,j \rangle) - g_t(i) - g_t(j)}{1 + J_t(j,i)} & \text{if } \exists i \in S \in \mathcal{C}_t, j \notin S : J_t(i, j), \\ 0 & \text{otherwise.} \end{cases} \tag{5.2}$$

The g values are initialized such that $\forall v \in V : g_0(v) = 0$. They are updated each round such that

$$g_{t+1}(v) = g_t(v) + y_{\mu_t(v)}. \tag{5.3}$$

The value $g_t(v)$ can therefore be interpreted as the amount of slack remaining in the dual variables during round t before an edge incident to vertex v becomes tight.

Let τ be the number of rounds required for the algorithm to terminate—or, in the case of the distributed algorithm, the number of rounds required to reach quiescence. Therefore, τ is the earliest round during which there are no components without depots:

$$\tau = \min_{t \in \mathbb{N}_0} \left(\forall C \in \mathcal{C}_t : C \cap T \neq \emptyset \right). \tag{5.4}$$

The notation is now sufficient to introduce the parallel version of the algorithm, given in Algorithm 3. A snapshot of the algorithm's execution, along with an example of our notation, is given in Figure 5.5.

5.2.3 Analysis

The various performance guarantees of the algorithm are proven in this section. First, Lemmas 5 and 6 lead to Proposition 6 which implies that the solutions found by the algorithm are acyclic and thereby forests, implying that they are primal feasible. Proposition 7 states that the main loop (line 10

Algorithm 3 The parallel location design and routing algorithm.

1: **procedure** PARALLEL-LOCATION-DESIGN(G, T, w)

Require: $G = \langle V, E \rangle$ is an undirected graph already augmented with the special depot vertices. $T \subset V$ is the set of "special" depot vertices. $w : E \to [1, \frac{3}{2}] \in \mathbb{Q}$ is a weight or cost function such that each edge $e \in E$ has an associated cost. $\mu_t : V \to 2^V$ is a convenience function mapping vertices to their connected component in $\mathcal{C}_t$.

Ensure: F' is the resulting spanning forest.

2:      $t \leftarrow 0$

3:      $F \leftarrow \emptyset$ /* Implicitly set $y_S = 0$ for all $S \subset V$ */

4:      **for all** $v \in V$ **do in parallel**

5:          $S \leftarrow \{v\}$

6:          $\mu_0(v) \leftarrow S$

7:          $\mathcal{C}_0 \leftarrow \mathcal{C}_0 \cup \{S\}$

8:          $g_0(v) \leftarrow 0$

9:      **end for**

10:      **while** $\exists C \in \mathcal{C}_t : C \cap T = \emptyset$ **do**

11:          $t \leftarrow t + 1$

12:          $\mathcal{C}_t \leftarrow \mathcal{C}_{t-1}$

13:          **for all** $C \in \mathcal{C}_{t-1}$ **do in parallel**

14:              $I(C) \leftarrow 0$ /* I is a temporary map */

15:              $K(C) \leftarrow \emptyset$ /* K is a temporary map */

16:          **end for**

17:          **for all** $C \in \{S \in \mathcal{C}_{t-1} : C \cap T = \emptyset\}$ **do in parallel**

18:              Find an edge $e = \langle u, v \rangle \in \delta(C)$ such that $u \in C$ and $\varepsilon = w(e) - g_{t-1}(v) - g_{t-1}(u)$ is minimized.

19:              $F \leftarrow F \cup \{e\}$

20:              $\mathcal{C}_t \leftarrow (\mathcal{C}_t \setminus (\{\mu_{t-1}(v)\} \cup \{\mu_{t-1}(u)\})) \cup \{\mu_{t-1}(v) \cup \mu_{t-1}(u)\}$

21:              $\mu_t(u) \leftarrow \mu_{t-1}(v) \cup \mu_{t-1}(u)$

22:              $K(C) \leftarrow \mu_{t-1}(v)$

23:              $I(C) \leftarrow \varepsilon$

24:          **end for**

25:          **for all** $v \in V$ **do in parallel**

26:              **if** $K(K(\mu_{t-1}(v))) = \mu_{t-1}(v)$ **then**

27:                  $I(\mu_{t-1}(v)) \leftarrow \frac{I(\mu_{t-1}(v))}{2}$

28:              **end if**

29:              $g_t(v) \leftarrow g_{t-1}(v) + I(\mu_{t-1}(v))$

30:          **end for**

31:          $A \leftarrow$ the keys of I sorted by descending ε value.   /* Implicitly used for analysis; need not be implemented */

32:          **for** $i \leftarrow 1$ **to** $|A|$ **do** /* Implicit */

33:              **for** $j \leftarrow 1$ **to** i **do** /* Implicit */

34:                  $y_{A[j]} \leftarrow y_{A[j]} + I(A[i])$ /* Implicit */

35:              **end for**

36:          **end for**

37:      **end while**

38:      $F' \leftarrow \{e \in F : \text{For some connected component } N \text{ of } \langle V, F \setminus \{e\} \rangle \text{ it is true that } N \cap T = \emptyset\}$

39: **end procedure**
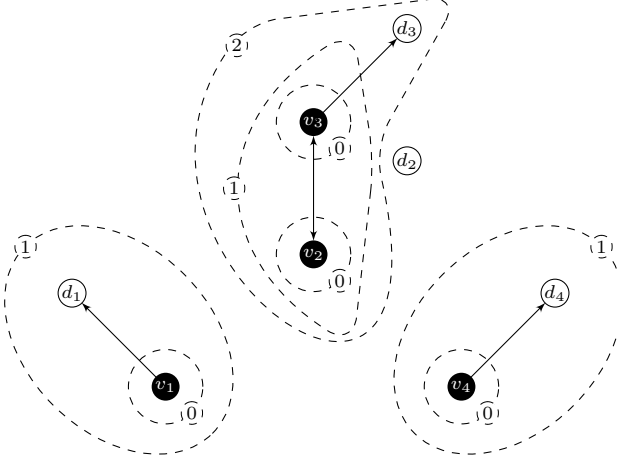
$$t=0 \quad \begin{cases} \forall S \subset V : y_S = 0 \\ \mathcal{C}_0 = \{\{v_1\}, \{v_2\}, \{v_3\}, \{v_4\}, \{d_1\}, \{d_2\}, \{d_3\}, \{d_4\}\} \\ \forall i \in \{1,2,3,4\} : g_0(v_i) = g_0(d_i) = 0 \end{cases}$$

$$t=1 \quad \begin{cases} J_1(v_1, d_1) = J_1(v_2, v_3) = J_1(v_3, v_2) = J_1(v_4, d_4) = 1 \\ g_1(v_1) = g_1(v_4) = g_1(v_2) = g_1(v_3) = 100 \\ y_{\{v_1\}} = y_{\{v_4\}} = 100, \ y_{\{v_2\}} = y_{\{v_3\}} = 50 \\ \mathcal{C}_1 = \{\{v_1, d_1\}, \{v_2, v_3\}, \{v_4, d_4\}, \{d_2\}, \{d_3\}\} \end{cases}$$

$$t=2 \quad \begin{cases} J_2(v_3, d_3) = 1 \\ y_{\{v_2, v_3\}} = 100 \\ \mathcal{C}_2 = \{\{v_1, d_1\}, \{v_2, v_3, d_3\}, \{v_4, d_4\}, \{d_2\}\} \end{cases}$$

$\tau = 2 \qquad$ because every $C \in \mathcal{C}_2$ contains a depot.

Figure 5.5: Snapshots for three rounds of the algorithm solving the augmented distribution network problem of Figure 5.4a. Connected components are visualized as dashed regions. The $\overset{\frown}{\#}$ labels denote the round number during which a connected component was created. Directions on the edges represent the component that chose to make that edge tight. In other words, $\bullet \!\rightarrow\! \circ \implies J(\bullet, \circ)$.

of Algorithm 3) will have a logarithmic number of iterations. Claim 5 leads to Proposition 8 which states that the solutions found by the algorithm are dual feasible. Finally, Lemma 7 leads to Proposition 9 which states that as long as the edge weights are embedded in the proper metric space then the algorithm is a 2-approximation.

Lemma 5. *Any cycle in the intersection graph (q.v. footnote 1 on page 31) of F_{t+1} formed from $\mathcal{C}_t$ must consist solely of edges along the cuts between depot-less components.*

Proof. Assume, on the contrary, that there exists a cycle containing an edge that is incident to at least one component containing a depot. Let $\langle u, v \rangle$ be such an edge and assume $\mu_t(v)$ contains at least one depot. (5.1) implies that v's connected component has no outgoing edges,

$$\forall i \in \mu_t(v) : (\neg \exists j \in V : J_t(i, j)),$$

which contradicts the fact that $\langle u, v \rangle$ is in a cycle. $\qquad \square$

The "potential cost" of an edge is the fractional quantity associated with ε on line 18 of Algorithm 3.

Lemma 6. *Any cycle in the intersection graph of F_{t+1} formed from $\mathcal{C}_t$ must consist of edges of equal potential cost.*

Proof. Let $e_1 = \langle u_1, v_1 \rangle$ be an edge in a cycle. (5.1) implies that all edges in a cycle must be cuts between existing connected components. Therefore, $\mu_t(u_1) \neq \mu_t(v_1)$. Furthermore, there must be another edge in the cycle, $e_2 = \langle u_2, v_2 \rangle$, such that $\mu_t(v_2) = \mu_t(u_1)$. It must also be true that $J_t(u_1, v_1) = J_t(u_2, v_2) = J_t^+(u_1, v_2) = 1$. By Lemma 5 there are no depots in any of the components in the cycle. Therefore, applying (5.1) gives

$$w(e_1) - g_t(u_1) - g_t(v_1) \leq w(e_2) - g_t(u_2) - g_t(v_2).$$

In general, this inequality will hold for the incoming and outgoing edges of any connected component in the cycle. Therefore, by transitivity,

$$
\begin{aligned}
w(e_1) - g_t(u_1) - g_t(v_1) &\leq w(e_2) - g_t(u_2) - g_t(v_2) \\
&\leq w(e_1) - g_t(u_1) - g_t(v_1),
\end{aligned}
$$

implying that

$$w(e_1) - g_t(u_1) - g_t(v_1) = w(e_2) - g_t(u_2) - g_t(v_2). \qquad \square$$

Proposition 6. *The intersection graph of F_{t+1} formed from $\mathcal{C}_t$ is acyclic.*

Proposition 7.

Proof. Assume, on the contrary, that there is a round t during which a cycle of length ℓ is formed. Since the graph is simple, $\ell > 1$. By Lemma 6, all of the edges in the cycle must be of equal potential cost. Therefore, each connected component will have had a tie between two fringe edges which must have been broken using the edge ordering. Therefore, either $\ell = 1$ or there are two edges with the same unique identifier, both of which are contradictions.   □

Claim 3. *If all edge weights are coprime, then Proposition 6 will hold even if the unique identifier ordering assumption does not.*

Corollary 2. $F_0, \ldots, F_\tau$ *are all acyclic.*

Proof. Since $F_0 = \langle V, \emptyset \rangle$, the base case is acyclic. Induction over Proposition 6 then proves the corollary.   □

Let $A_f(t) = A(t)$ be an upper bound on the number of depot-less components at the beginning of round t. Similarly, let $L_f(t) = L(t)$ be an upper bound on the total number of components at the beginning of round t. Clearly,

$$A(t) \geq |\{C \in \mathcal{C}_t : C \cap T = \emptyset\}|, \text{ and}$$
$$L(t) \geq |\mathcal{C}_t| \geq A(t).$$

In general, every depot-less component will union with another component during every round. Regardless of whether such a component chooses to union with a component containing a depot or one that is depot-less, the total number of components will decrease by one half the number of depot-less components. Therefore $L(t) = L(t-1) - A(t-1)/2$. Now let us consider the extrema for the change in the number of depot-less components. If all depot-less components choose to union with other depot-less components then we have $A(t) = A(t-1)/2$. On the other hand, if as many depot-less components union with components containing depots as possible, then $A(t) \leq \min(A(t-1), L(t-1) - A(t-1))$. Therefore, the general recurrences for $A(t)$ and $L(t)$ are:

$$A(t) = \max\left(\frac{A(t-1)}{2}, \min\left(A(t-1), L(t-1) - A(t-1)\right)\right), \quad (5.5)$$

$$L(t) = L(t-1) - \frac{A(t-1)}{2}.$$

The initial conditions for the recurrences are clearly

$$A(0) = |\{C \in \mathcal{C}_0 : C \cap T = \emptyset\}| = |R|,$$
$$L(0) = |\mathcal{C}_0| = |R| + |T| = 2|R|.$$

Claim 4. $A(t-1)/2$ *will always dominate the maximization in* (5.5).

Validation of this claim will be given in the proof of the following proposition.

The algorithm will terminate after a logarithmic number of rounds (i.e., iterations of the main loop on line 10 of Algorithm 3): $\tau = O(\log n)$.

Proof. This follows from the fact that the algorithm will terminate once the number of depot-less components is zero:

$$\forall t \in \mathbb{N}_0 : A(t) = 0 \implies t \geq \tau.$$

Therefore, the burden of this proof is to show that, the $A(t)$ recurrence will converge exponentially, implying that $\tau = O(\log n)$.

If Claim 4 holds, then it is clear that the $A(t)$ recurrence will converge exponentially:

$$A(t) = \frac{A(0)}{2^t}$$

$$L(t) = 2|R| - \sum_{i=0}^{t} \frac{A(0)}{2^i}.$$

Let $k = \frac{A(0)}{2|R|}$ and observe that $k = \frac{1}{2}$. Substituting $2k|R|$ for $A(0)$ ensures that the minimization in $A(t)$ will always evaluate to $L(t-1) - A(t-1)$ because

$$
\begin{aligned}
\forall t \in \mathbb{N}_0 : A(t) &\geq L(t) - A(t) \\
\iff 2|R|\frac{k}{2^t} &\geq 2|R|\left(1 - \left(\sum_{i=0}^{t}\frac{k}{2^i}\right) - \frac{k}{2^t}\right) \\
\iff \frac{2k}{2^t} &\geq 1 - \sum_{i=0}^{t}\frac{k}{2^i} \\
\iff k &\geq \frac{2^t}{1 + 2^{t+1}},
\end{aligned}
$$

which is true because $2^t/(1 + 2^{t+1})$ is bounded above by $\frac{1}{2}$. Therefore, provided Claim 4 holds, (5.5) can be simplified to

$$A(t) = \max\left(\frac{A(t-1)}{2}, L(t-1) - A(t-1)\right).$$

Claim 4 obviously holds for the base case of $t = 1$ because $A(0)/2 = 2k|R|$ is bounded below by $L(0) - A(0) = 2|R| - \frac{2|R|}{2}$. Therefore, Claim 4 will hold as long as

$$\frac{A(t)}{2} \geq L(t) - A(t).$$

This equates to

$$k \geq 2^{t+1}\left(1 - \left(\sum_{i=0}^{t}\frac{k}{2^i}\right) - \frac{k}{2^t}\right)$$
$$\geq \frac{2 \times 4^t}{2^t + 4^{t+1}},$$

which must be true because $(2 \times 4^t)/(2^t + 4^{t+1})$ is bounded above by $\frac{1}{2}$. □

Claim 5. *Let t' be the round during which an edge $e = \langle u, v \rangle$ is added to the spanning forest. Then e will not be in the cut of any component in a subsequent round:*
$$\forall t > t', C \in \mathcal{C}_t : e \notin \delta(C).$$

Proof. $\mu_{t'+1}(u) = \mu_{t'+1}(v) = \mu_{t'}(u) \cup \mu_{t'}(v)$. Therefore, in all rounds subsequent to t' both endpoints of e are in the same component and therefore cannot be in the fringe. □

Proposition 8. *The vector y is a feasible solution to* (LDRP-D) *and has the property*
$$\sum_{e \in F_\tau} w(e) \leq \sum_{e \in F_\tau} \sum_{S : e \in \delta(S)} y_S.$$

Proof. The fact that y is a feasible solution to (LDRP-D) is a consequence of the fact that y is initially zero and is updated according to (5.2). Let t be the round during which an edge $e = \langle u, v \rangle \in F_\tau$ was added to the forest. From (5.3), note that

$$\left(g_t(u) = \sum_{i=0}^{t-1} y_{\mu_i(u)}\right) \bigwedge \left(g_t(v) = \sum_{i=0}^{t-1} y_{\mu_i(v)}\right).$$

Furthermore, at the beginning of round t the potential for e is

$$\varepsilon = w(e) - g_t(u) - g_t(v).$$

Once e is added to F_t, the dual variables $y_{\mu_t(u)}$ and $y_{\mu_t(v)}$ are updated according to (5.2). Then there are three possible cases:

1. both u and v's components are depot-less and e is added mutually: $\mu_t(u) \cap T = \mu_t(v) \cap T = \emptyset$ and $J_t(u, v) = J_t(v, u)$;

2. both u and v's components are depot-less and e is not added mutually: $\mu_t(u) \cap T = \mu_t(v) \cap T = \emptyset$ and $J_t(u, v) \neq J_t(v, u)$; or

3. u's component has a depot and v's does not, or *vice versa*: $|\mu_t(u) \cap T| + |\mu_t(v) \cap T| = 1$.

In case 1,

$$y_{\mu_t(u)} + y_{\mu_t(v)} = \frac{\varepsilon}{1 + J_t(v, u)} + \frac{\varepsilon}{1 + J_t(u, v)} = \varepsilon,$$

implying that

$$w(e) = \sum_{i=0}^{t} \left(y_{\mu_i(u)} + y_{\mu_i(v)} \right). \tag{5.6}$$

For case 2, assume without loss of generality that $J_t(u, v) = 1$ and $J_t(v, u) = 0$. For case 3, assume without loss of generality that $\mu_t(u) \cap T = \emptyset$ and $\mu_t(v) \cap T \neq \emptyset$. Then for both of these cases note that

$$y_{\mu_t(u)} = \frac{\varepsilon}{1 + J_t(v, u)} = \varepsilon,$$

implying that

$$w(e) = y_{\mu_t(u)} + \sum_{i=0}^{t-1} \left(y_{\mu_i(u)} + y_{\mu_i(v)} \right). \tag{5.7}$$

Claim 5 implies that the summations in (5.6) and (5.7) comprise all sets that cut e, thus completing the proof. $\qquad\square$

Lemma 7 (Williamson, *& al.* [14, Theorem 3.6]). *Let H be the intersection graph of the final spanning forest F_τ formed from $\mathcal{C}_t$. Remove all isolated vertices in H that correspond to components in $\mathcal{C}_t$ that have depots. Then no leaf in H corresponds to a component containing a depot.*

Proof. This is a transcription of the proof, reproduced here for completeness using our notation in the specific domain of the Location Design and Routing Problem. Assume the contrary: Let v be a leaf, let C_v be its associated component containing a depot, let e be the edge incident to v, and let $C \subseteq V$ be the component of F which contains C_v. Let N and $C \setminus N$ be the two components formed by removing edge e from the edges of component C. Without loss of generality, say that $C_v \subseteq N$. The set $N \setminus C_v$ is partitioned by some of the components of the current round; call these $C_1, \ldots, C_k$. Since vertex v is a leaf, no edge in F_τ connects C_v to any C_i. Thus by the construction of F_τ, $\forall i \in \{1, \ldots, k\} : C_i$ contains at least one depot. Since C_v also contains at least one depot, it follows that N must too. Clearly, if two components S and B both contain depots and $B \subseteq S$, then the component $S \setminus B$ must also contain a depot. Since we know that C contains a depot then $N \setminus C$ must as well, and thus by the construction of F_τ, $e \notin F_\tau$, which is a contradiction. $\qquad\square$

We have thus far proven that Algorithm 3 produces a feasible solution to both (LDRP-IP) and (LDRP-D) in a logarithmic number of rounds. It therefore only remains to bound the quality of the solution, which we shall do now. To prove that the algorithm is a 2-approximation, we use the same

technique of defining an invariant over the weights of the edges added to the forest as in the generalized algorithm.

The cost of the final spanning forest F_τ is bounded above by $\left(2 - \dfrac{2}{|R|}\right) Z^*_{LDRP\text{-}IP}$

if the normalized edge weights are in the range $\left[1, \frac{3}{2}\right]$.

Proof sketch. The basic intuition of our result is that the average degree of a vertex in a forest of at most n vertices is at most $2 - \frac{2}{n}$. *Cf.* the proof of Proposition 3 on page 34.

$\square$

This section has served to prove that the parallel version of the algorithm is correct, complete, and maintains the runtime and approximation bounds we claim. In the next section we will show that it can be trivially extended to a distributed algorithm.

5.2.4 Distributing the Algorithm

We make the same assumptions on the communications network as those of multidirectional graph search given in §3.1.2. Namely, we assume that the communications network provides guaranteed delivery of messages, however, there may be arbitrary latency (*i.e.*, the network is asynchronous [46]). Without loss of generality, we assume that there is one intelligent agent per vertex in the graph. We further assume that all agents are honest and correct and thus need not consider the problem of Byzantine failure [85]. The agents are non-adversarial insofar as their primary goal is to find a feasible solution to the location design and routing problem. The collective is therefore what is called a *cooperative multiagent system* [17]. Agents' perceptions of the graph being optimized (*e.g.*, the network topology) are consistent, possibly through the use of a distributed consensus algorithm [46]. Each agent/vertex has a unique identifier with a globally agreed ordering. This ordering can be used to construct a total ordering over the edges (*e.g.*, by combining the unique identifiers of the incident vertices).

The proposed distributed algorithm is round-based, with each round corresponding to a single iteration of the main loop (line 10) of the parallel algorithm. The rounds proceed asynchronously between connected components. Therefore, as the connected components grow throughout the execution of the algorithm, the rounds naturally become synchronized.

The distributed version of the algorithm, given in Algorithm 4, uses simple synchronous message passing in place of the shared memory of the parallel algorithm. The distributed algorithm is proven deadlock-free in the following proposition.

Proposition 10. *Algorithm 4 is deadlock-free.*

Proof. The only blocking operations in the algorithm occur on lines 12, 15, 22, 27, and 38.

Waiting for an `Update` message on line 12 will clearly not deadlock since `UpdateRequest` messages are sent on the line previous, and the procedure on line 1 immediately sends an `Update` message in reply to requests.

Similarly, line 14 ensures that line 15 will not deadlock.

The fact that the procedure on line 7 contains no blocking operations ensures that any `Union` message will be immediately replied with an `Ack` message. Therefore, line 22 will not deadlock.

Any vertices that choose to make an incident edge tight will block on line 27 unless either the choice of edge was mutual or the connected component on the other end of the edge contains a depot. Let $\langle v, u \rangle$ be such an edge. Note that $J(v, u) = 1$ and $J(u, v) = 0$. In such a case, v will be added by u to I upon receipt of v's `Union` message on line 14. Now, by a similar argument to the proof of Lemma 6, note that there must be some x and y such that $J^+(u, y)$ and $J(y, x) = 1$ and either $J(x, y) = 1$ or x's component already contains a depot (otherwise there would be a cycle). Therefore, u will not deadlock on line 27 and will eventually send an `Adding` message to v on line 32.

Finally, the main loop invariant on line 10 ensures that only depot-less components block on line 12. Furthermore, (5.1) ensures that there must be exactly one vertex in each depot-less component that chooses to add exactly one edge during each round. Therefore, line 38 will not deadlock. $\qquad\square$

Assuming all messages can be both unicast and broadcast in a constant number of messaging rounds then, by the same argument as in the proof of Proposition 7, the main loop of the distributed algorithm on line 10 will run in a logarithmic number of iterations and thereby will have a logarithmic number of messaging rounds. If this assumption does not hold—for example, if *ad hoc* routing is required—then the algorithm can be trivially extended to support the Broadcast-Message function itself. To do this, the algorithm will use the partially constructed spanning trees within each connected component for multicast.

The most expensive operations in the distributed algorithm are

1. determining the fringe edge with minimal potential; and

2. merging two connected components once an edge between them becomes tight.

Should efficient broadcast be unavailable, one way of implementing these operations is to have broadcast messages convergecasted up the partially constructed spanning tree in the component. This method was used to solve a similar problem in [68]. The root of the tree (*e.g.*, the vertex that was added

Algorithm 4 The distributed location design and routing algorithm. Message handlers are given in Algorithm 5.

1: **procedure** Distributed-Location-Design(v, κ)
Require: κ is the cost of opening a depot at v.
2:     $C \leftarrow \emptyset$ /* The other fringe vertices in our component. */
3:     $N \leftarrow \delta(\{v\}) \cup \{\langle v, d_v \rangle\}$ /* The neighborhood of v along with the special depot vertex d_v */
4:     $F \leftarrow \emptyset$ /* The spanning forest of our component. */
5:     $w(d_v) \leftarrow \kappa$
6:     **for all** $i \in \delta(C) \cup \{v\}$ **do**
7:         $g(i) \leftarrow 0$
8:     **end for**
9:     $I \leftarrow \emptyset$
10:    **while** F does not contain a depot **do**
11:        Broadcast-Message(`UpdateRequest`) to all $u \in N$
12:        Block until we have received and handled all `Update` messages from N.
13:        Find an edge $e = \langle v, u \rangle \in N$ such that $u \notin C$ and $\varepsilon = w(e) - g(v) - g(u)$ is minimized.
14:        Broadcast-Message(`Potential`$\langle \varepsilon \rangle$) to all $c \in C$
15:        Listen for all broadcast `Potential` messages from the fringe
16:        **if** ε is the smallest in the fringe and ties are broken in our favor **then**
17:            $N \leftarrow N \setminus \{u\}$
18:            **if** $u = d_v$ **then**
19:                $C_m \leftarrow \{u\}$
20:            **else**
21:                Send-Message(`Union`$\langle e \rangle$) to u
22:                Wait for an `Ack`$\langle m, C_m \rangle$ message from u
23:                **if** $m = $ `Mutual` **then**   /* u also chose to make edge e tight */
24:                    $\varepsilon \leftarrow \frac{\varepsilon}{2}$
25:                **else**
26:                    **if** $m = $ `Not-Mutual` **then**   /* this means u does not yet have a depot */
27:                        Block until we have received and handled an `Adding`$\langle e_a, \varepsilon_a, C_a \rangle$ message from u
28:                    **end if**
29:                    $C_m \leftarrow C_a$
30:                **end if**
31:            **end if**
32:            Broadcast-Message(`Adding`$\langle e, \varepsilon, C_m \rangle$) to all $c \in C \cup I$
33:            $I \leftarrow \emptyset$
34:            $C \leftarrow C \cup C_m$
35:            $g(v) \leftarrow g(v) + \varepsilon$
36:            $F \leftarrow F \cup \{e\}$
37:        **else**
38:            Block until we have received and handled an `Adding` message from another fringe member
39:        **end if**
40:    **end while**
41: **end procedure**

Algorithm 5 Message handlers for Algorithm 4.

 1: **procedure** HANDLE-UPDATE-REQUEST-MESSAGE(`UpdateRequest` sent by u)
 2:     SEND-MESSAGE(`Update`$\langle v, g(v) \rangle$) to u
 3: **end procedure**
 4: **procedure** HANDLE-UPDATE-MESSAGE(`Update`$\langle v_u, g_u \rangle$ sent by u)
 5:     $g(v_u) \leftarrow g_u$
 6: **end procedure**
 7: **procedure** HANDLE-UNION-MESSAGE(`Union`$\langle e_u \rangle$ sent by u)
 8:     **if** $e = e_u$ **then**
 9:         SEND-MESSAGE(`Ack`$\langle$Mutual$, C \rangle$)
10:     **else if** F already contains a depot **then**
11:         SEND-MESSAGE(`Ack`$\langle$Has-Depot$, \emptyset \rangle$)
12:     **else**
13:         SEND-MESSAGE(`Ack`$\langle$Not-Mutual$, \emptyset \rangle$)
14:         $I \leftarrow I \cup \{u\}$
15:     **end if**
16: **end procedure**
17: **procedure** HANDLE-ADDING-MESSAGE(`Adding`$\langle e_a, \varepsilon_a, C_a \rangle$)
18:     $F \leftarrow F \cup \{e_a\}$
19:     $g(v) \leftarrow g(v) + \varepsilon_a$
20:     $C \leftarrow C \cup C_a$
21: **end procedure**

---

the earliest and is of highest unique identifier) can then perform the operation and unicast the result back down to the relevant fringe member(s).

This section has applied the technique of distributed multidirectional graph search to the location design and routing problem. The algorithm guaranteed to run in a logarithmic number communication rounds. Provided that the weight of the heaviest edge added in a round is no more than 150% of the lightest edge, the algorithm is guaranteed to produce a solution whose cost is no worse than two times optimal. This invariant can be maintained in general by embedding the true edge weights into $[1, \frac{3}{2}] \in \mathbb{Q}$.

5.3   Art Gallery Problems

Art gallery problems generally ask to find the minimum number of guards required to observe the interior of a polygonal area [8]. Over the past thirty years since their proposition, these problems have been thoroughly studied by the computational geometry community. Interest in art gallery problems has seen a recent resurgence given their application to a number of areas of multiagent systems. For example, many robotics, sensor network, wireless networking, and surveillance problems can be mapped to variants of the art gallery problem. Since such problems can be naturally distributed, a logical approach is to apply the multiagent paradigm (*i.e.*, each guard is an agent).

As a motivating scenario, consider a wireless sensor network such as the one pictured in Figure 1.3. Since one goal of the network is to maximize

survivability, it may be desirable to conserve battery power by having as few sensors active as necessary, especially for sensors with wide overlapping fields of view. The problem is then to find a minimum subset of sensors that need to remain active in order to provide a desirable level of coverage. As another scenario, consider a group of mobile robots each equipped with a wireless access point. The objective of the robots is to maximally cover an area with the wireless network. As the robots are traveling between waypoints, though, it is highly likely that there will be a large amount of overlap in the coverage. Therefore, in order to save power, the robots might want to choose a maximum subset of robots that can lower their transmit power while still retaining coverage. The difficulty in each of these scenarios is for the agents to collectively find the solution without centralizing computation. Centralization is infeasible either due to lack of resources (*i.e.*, no single agent has powerful enough hardware to solve the global problem) or due to lack of time (*i.e.*, centralizing the problem will take at least a linear number of messaging rounds). These problems are **NP**-HARD and can be modeled as art gallery problems.

Solving art gallery problems using multiagent systems is not a new idea. In Lass, *& al.* [86], we applied the multiagent coordination paradigm of Distributed Constraint Optimization (DisCOP) to a variant of the problem in which a fixed number of robotic guards must patrol a polygonal area. The difficulty with using DisCOP, however, is that all known algorithms that provide a constant bound on the quality of the solution will in the worst case be exponential in either messaging or memory [87]. Whereas DisCOP is a general problem solving paradigm, Ganguli, *& al.*, developed a multiagent algorithm specifically for solving art gallery problems [88]. This algorithm has several desirable properties including optimality, however, there is no theoretical bound on runtime.

The dominating set problem is a generalization of the art gallery problem that asks to find a minimum subset of the vertices in a graph such that every vertex not in the subset has at least one member of the subset in its neighborhood (*viz.*, related to the *hitting set problem*). This problem is also **NP**-COMPLETE [43]. The dominating set problem has been widely studied in the wireless networking community given its applications to *ad hoc* routing [89] and efficient multicast [15]. The majority of the proposed distributed algorithms for the dominating set problem, however, do not have bounds on both runtime and solution quality. In the wireless networking community much emphasis is placed on devising algorithms with a constant number of communication rounds. Ruan, *& al.*, propose a one-step greedy algorithm for approximating a solution to the dominating set problem [90], however, the performance ratio is a function of the degree distribution of the graph. Kuhn and Wattenhofer provide a more general result, producing an algorithm that has a variable approximation bound as a function of the number of commu-

nication rounds executed. Kuhn and Wattenhofer's approach, however, is likewise tied to the degree distribution of the graph. Finally, Huang, *& al.*, show that with a slightly higher message complexity a solution no worse than 12 times the cost of the optimal solution can be found [91]. This chapter introduces an algorithm based on multidirectional constrained graph search that exhibits a lower constant of approximation in a worst case linear—but often logarithmic—number of communication rounds.

This section introduces a novel distributed version of a multiagent approximation algorithm based on the primal-dual schema for solving the distributed art gallery and dominating set problems (§5.3.1). We show that this algorithm is correct and complete and bound its runtime with respect to communication rounds (§5.3.2). Next, we show through empirical analysis that the algorithm will produce solutions within a constant factor of optimal with high probability (§5.3.3). We then show that some well known variants of the problem can also be solved with the same algorithm and, under certain reasonable assumptions about the distribution of edge weights, the algorithm will produce a solution no worse than two times optimal (independent of the topology of the problem) (§5.3.4).

5.3.1 Distributed Dominating Sets

This section defines the necessary formalism for the distributed dominating set problem, which is equivalent to the original art gallery problem of finding a minimum set of vertices from which the entire polygon is visible. This formalism is later used to define our algorithm.

Given two vertices of a polygon u and v, u is said to be *visible* from v if the line segment between them is contained within the polygon, *& vice versa*. The exterior of the polygon is forbidden for visibility graph edges. Given the vertices of a polygon, V, the *Art Gallery Problem* asks to find a minimum subset of the vertices $D \subseteq V$ such that for every $v \in V$ there is at least one $d \in D$ that is visible. The *visibility graph* of a polygon is constructed by adding an edge between all pairs of vertices that are visible to each other. For example, see Figure 5.6b. The Art Gallery Problem therefore reduces to finding a dominating set of the vertices in the polygon's visibility graph. Given a visibility graph $G = \langle V, E \rangle$, the object is to find a $D \subseteq V$ of minimum cardinality such that each $v \notin D$ has at least one $d \in D$ in its neighborhood. An example is given in Figure 5.6, with an optimal solution depicted in Figure 5.6c.

The analysis of the dominating set problem can be simplified by representing it as a connectivity problem. Therefore, let us augment the visibility graph with one special *guard* vertex d_i for each original vertex v_i, as in Figure 5.7a. Next, add an edge from each vertex to its associated guard vertex with a weight of one. The new overall set of edges is the original set of
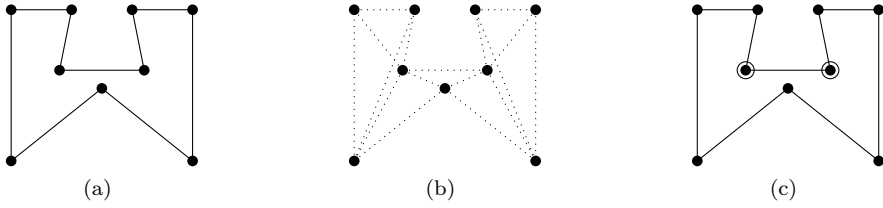
Figure 5.6: An art gallery, (a), with its associated visibility graph, (b), and an optimal placement of guards, (c). Guard placement is represented by ⦿.

edges from the visibility graph unioned with the set of new guard edges. All original edges from the visibility graph are given a weight of zero. Let $R = \{v_1, \ldots, v_n\}$ be the set of original vertices in the visibility graph and let $T = \{d_1, \ldots, d_n\}$ be the set of new special guard vertices with the new overall set of vertices $V = R \cup T$. Now the problem reduces to that of finding a minimum weight forest that spans R having the property that the length of the shortest path from any $v \in R$ to a $d \in T$ is no more than two edges. We will hereafter refer to this forest as "the spanning forest". Note, however, that the spanning forest does not necessarily span all of T.

In this new connectivity representation, a vertex v_i will be a part of D (*i.e.*, it will be chosen to become a guard) if the edge from it to its associated guard vertex is a part of the final spanning forest: $\langle v_i, d_i \rangle \in F \implies v_i \in D$. Let $f : 2^V \to \{0, 1\}$ be the function defining whether a connected component $S \subseteq V$ satisfies the requirement that each vertex is close to at least one guard. f is defined such that $f(S) = 1$ if and only if there exists an original vertex in S that is not within two edges distance of a guard in S: $f(S) = 1 \iff \exists u \in S \cap R \ \forall v \in S \cap T : |u \leadsto v| > 2$. Note that f is not proper. A component for which $f(S) = 1$ is said to be *unguarded*. The optimization problem on the augmented graph can be captured as the following integer program:

$$\text{minimize} \sum_{e \in E} w(e) x_e$$

subject to: (ART-IP)

$$x(\delta(S)) \geq f(S), \quad \forall S \subset V : S \neq \emptyset$$
$$x_e \in \{0, 1\}, \quad\quad\quad\quad \forall e \in E,$$

where each variable x_e is an indicator as to whether the edge e is a member of the final spanning forest, $\delta(S)$ is the set of edges having exactly one endpoint in S, and $x(F) \mapsto \sum_{e \in F} x_e$. Therefore, any forest $F \subseteq E$ will be a feasible solution to the problem if $f(S) = 0$ for every connected component S of the forest. Let (ART-LP) denote the linear programming relaxation of (ART-IP) obtained by replacing the integrality restriction with $x_e \geq 0$. The dual

of (ART-LP) is

$$\text{maximize} \sum_{S \subset V} f(S) y_S$$

subject to:

$$\sum_{S:e \in \delta(S)} y_S \leq w(e), \qquad \forall e \in E \tag{ART-D}$$

$$y_S \geq 0, \qquad \forall S \subset V : S \neq \emptyset.$$

Again, an edge is *tight* if $w(e) = \sum_{S:e \in \delta(S)} y_S$. Let $Z^*_{\text{ART-LP}}$ be the cost of the optimal solution to (ART-LP) and let $Z^*_{\text{ART-IP}}$ be the cost of the optimal solution to (ART-IP). Needless to say that $Z^*_{\text{ART-LP}} \leq Z^*_{\text{ART-IP}}$.

5.3.2   The Algorithm

The basic mechanism of the algorithm is that of constrained multidirectional graph search. We start off with an empty forest; each vertex is a member of its own connected component. Every round, each unguarded component greedily chooses to add one of its cut edges in the visibility graph to the forest, merging with the component on the other end of the edge. If the new component becomes guarded as a result of the merger then the new component stops actively growing. This has the effect of first finding a forest that spans the original visibility graph; then each connected component in the forest finds the minimum set of special vertices that is sufficient to be guarded. When all components are guarded the algorithm terminates. A technical sketch of the algorithm is given in Figure 5.7.

The remainder of this section provides the notation and mathematics required to formally define and model the algorithm. This will later be used to provide formal bounds on the runtime and performance of the algorithm, and also to prove correctness and completeness. We make the same assumptions on the communications network as in §5.2.4.

Let F_t be the partially constructed spanning forest at the beginning of round t. Let $\mathcal{C}_t$ be the set of connected components in F_t. For sake of brevity and simplicity, let $\mu_t : V \to \mathcal{C}_t$ be a function mapping vertices to their associated connected component during round t; therefore, $\mu_t(v) \mapsto C_i \implies v \in C_i (\in \mathcal{C}_t)$. A vertex that is incident to at least one edge in the cut of its connected component is said to be in the *fringe*. Let $g_t : V \to \mathbb{R}$ be a mapping of vertices to a real number during round t. These values represent the amount of slack remaining in the dual variables associated with a vertex.

Let $J_t : V \times V \to \{0, 1\}$ be a binary relation defining which edges will become tight during round t. Each unguarded component will choose to add the edge in its fringe that has minimal weight and dual variable slack.

(a) Augmented graph.

(b) Round 1.

(c)   Connectivity   after round 1.

(d) Round 2.

(e)   Connectivity   after round 2.

(f) Round 3.

(g) A guard is added.

(h) Round 4.

(i) A guard is added.

(j) Round 5.

(k) A guard is added.

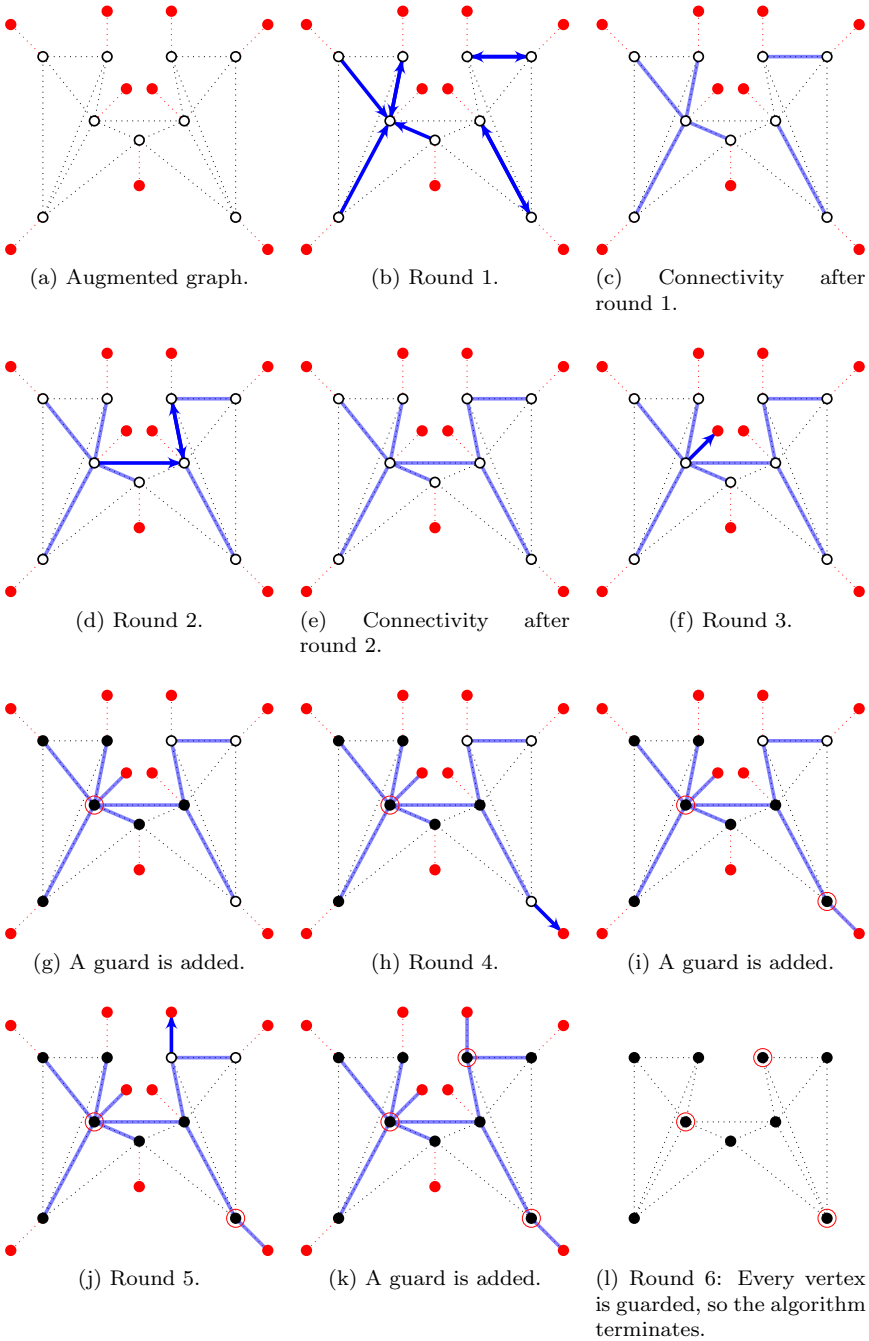(l) Round 6: Every vertex is guarded, so the algorithm terminates.

Figure 5.7: A sketch of the multidirectional constrained graph search algorithm solving the Art Gallery constrained forest problem.

Therefore, $J_t(u, v) = 1$ if and only if $f(\mu_t(u)) = 1$ and

$$\langle u, v \rangle = \operatorname*{arg\,min}_{\langle i,j \rangle \in \delta(\mu_t(u))} w(\langle i, j \rangle) - g_t(i) - g_t(j). \tag{5.8}$$

Ties in the minimization are broken based upon the ordering of the edges. Let J^+ denote the transitive closure of J. Note that J does not commute: $J(u, v) \not\Longrightarrow J(v, u)$. Also note that as long as there exists a feasible solution to (ART-IP) then the minimization ensures that each unguarded connected component must have exactly one edge in the fringe that becomes tight each round: $\forall C \in \mathcal{C}_t : f(C) = \sum_{\langle u,v \rangle \in \delta(C)} J_t(u, v)$.

F_t is the partially constructed spanning forest during round t, initialized to $F_0 = \langle V, \emptyset \rangle$. The forest is updated each round with the set of all edges that became tight during the round: $F_{t+1} = F_t \cup \{\langle u, v \rangle \in E : J_t(u, v) \vee J_t(v, u)\}$.

For a set $S \subseteq V$, let y_S be the dual variable associated with S. Initially all such variables are set to zero. Note that in actuality these variables need not be made part of an implementation of the algorithm; they exist solely for the purpose of proving properties of the algorithm [73]. These dual variables are implicitly updated as follows:

$$y_S \leftarrow \begin{cases} \frac{w(\langle i,j \rangle) - g_t(i) - g_t(j)}{1 + J_t(j,i)} & \text{if } \exists i \in S \in \mathcal{C}_t, j \notin S : J_t(i, j), \\ 0 & \text{otherwise.} \end{cases} \tag{5.9}$$

The g values are initialized such that $\forall v \in V : g_0(v) = 0$. They are updated each round such that

$$g_{t+1}(v) = g_t(v) + y_{\mu_t(v)}. \tag{5.10}$$

The value $g_t(v)$ can therefore be interpreted as the amount of slack remaining in the dual variables during round t before an edge incident to vertex v becomes tight.

Let τ be the number of rounds required for the algorithm to reach quiescence. Therefore, τ is the earliest round during which there are no unguarded components:

$$\tau = \operatorname*{arg\,min}_{t \in \mathbb{N}_0} \left(\forall C \in \mathcal{C}_t : f(C) = 0 \right). \tag{5.11}$$

The performance guarantees of the algorithm are proven in this section. First, Lemmas 8 and 9 lead to Proposition 11 which implies that any solution found by the algorithm is acyclic and thereby a forest, implying that it is primal feasible. Proposition 12 states that under certain common conditions the main loop (line 9 of Algorithm 6) will have a logarithmic number of iterations. Finally, Claim 7 leads to Proposition 13 which states any solution found by the algorithm is dual feasible.

Lemma 8. *Any cycle in the intersection graph (q.v. footnote 1 on page 31) of F_{t+1} formed from $\mathcal{C}_t$ must consist solely of edges along the cuts between unguarded components.*

Algorithm 6 The distributed art gallery/dominating set algorithm. Message handlers are defined in Algorithm 7.

1: **procedure** DISTRIBUTED-ART-GALLERY(v)
Require: v is the vertex associated with the location of this agent.
Ensure: v will become a guard if $\langle v, d_v \rangle \in F$.
2:     $C \leftarrow \emptyset$ /* The other fringe vertices in our component. */
3:     $N \leftarrow \delta(\{v\}) \cup \{\langle v, d_v \rangle\}$ /* The neighborhood of v along with the special guard vertex d_v */
4:     $F \leftarrow \emptyset$ /* The spanning forest of our component. */
5:     **for all** $i \in \delta(C) \cup \{v\}$ **do**
6:         $g(i) \leftarrow 0$
7:     **end for**
8:     $I \leftarrow \emptyset$
9:     **while** F is unguarded **do**
10:         BROADCAST-MESSAGE(`UpdateRequest`) to all $u \in N$
11:         Block until we have received and handled all `Update` messages from N.
12:         Find an edge $e = \langle v, u \rangle \in N$ such that $u \notin C$ and $\varepsilon = w(e) - g(v) - g(u)$ is minimized.
13:         BROADCAST-MESSAGE(`Potential`$\langle\varepsilon\rangle$) to all $c \in C$
14:         Listen for all broadcast `Potential` messages from the fringe.
15:         **if** ε is the smallest in the fringe and ties are broken in our favor **then**
16:             $N \leftarrow N \setminus \{u\}$
17:             **if** $u = d_v$ **then**
18:                 $C_m \leftarrow \{u\}$ /* v is to become a guard. */
19:             **else**
20:                 SEND-MESSAGE(`Union`$\langle e\rangle$) to u
21:                 Wait for an `Ack`$\langle m, C_m \rangle$ message from u
22:                 **if** $m = $ `Mutual` **then**   /* u also chose to make edge e tight */
23:                     $\varepsilon \leftarrow \frac{\varepsilon}{2}$
24:                 **else**
25:                     **if** $m = $ `Not-Mutual` **then**   /* this means u is not yet guarded */
26:                       Block until we have received and handled an `Adding`$\langle e_a, \varepsilon_a, C_a \rangle$ message from u
27:                     **end if**
28:                   $C_m \leftarrow C_a$
29:                 **end if**
30:             **end if**
31:             BROADCAST-MESSAGE(`Adding`$\langle e, \varepsilon, C_m \rangle$) to all $c \in C \cup I$
32:             $I \leftarrow \emptyset$
33:             $C \leftarrow C \cup C_m$
34:             $g(v) \leftarrow g(v) + \varepsilon$
35:             $F \leftarrow F \cup \{e\}$
36:         **else**
37:             Block until we have received and handled an `Adding` message from another fringe member
38:         **end if**
39:     **end while**
40: **end procedure**

Algorithm 7 Message handlers for Algorithm 6.

1: **procedure** HANDLE-UPDATE-REQUEST-MESSAGE(`UpdateRequest` sent by u)
2:     SEND-MESSAGE(`Update`$\langle v, g(v)\rangle$) to u
3: **end procedure**
4: **procedure** HANDLE-UPDATE-MESSAGE(`Update`$\langle v_u, g_u\rangle$ sent by u)
5:     $g(v_u) \leftarrow g_u$
6: **end procedure**
7: **procedure** HANDLE-UNION-MESSAGE(`Union`$\langle e_u\rangle$ sent by u)
8:     **if** $e = e_u$ **then**
9:         SEND-MESSAGE(`Ack`$\langle$`Mutual`$, C\rangle$)
10:     **else if** F is already guarded **then**
11:         SEND-MESSAGE(`Ack`$\langle$`Is-Guarded`$, \emptyset\rangle$)
12:     **else**
13:         SEND-MESSAGE(`Ack`$\langle$`Not-Mutual`$, \emptyset\rangle$)
14:         $I \leftarrow I \cup \{u\}$
15:     **end if**
16: **end procedure**
17: **procedure** HANDLE-ADDING-MESSAGE(`Adding`$\langle e_a, \varepsilon_a, C_a\rangle$)
18:     $F \leftarrow F \cup \{e_a\}$
19:     $g(v) \leftarrow g(v) + \varepsilon_a$
20:     $C \leftarrow C \cup C_a$
21: **end procedure**

Proof. Assume, on the contrary, that there exists a cycle containing an edge that is incident to at least one guarded component. Let $\langle u, v\rangle$ be such an edge and assume $\mu_t(v)$ is guarded. (5.8) implies that v's connected component has no outgoing edges,

$$\forall i \in \mu_t(v) : (\neg \exists j \in V : J_t(i, j)),$$

which contradicts the fact that $\langle u, v\rangle$ is in a cycle. $\qquad\square$

The *potential cost* of an edge is the fractional quantity associated with ε on line 12 of Algorithm 6.

Lemma 9. *Any cycle in the intersection graph of F_{t+1} formed from $\mathcal{C}_t$ must consist of edges of equal potential cost.*

Proof. Let $e_1 = \langle u_1, v_1\rangle$ be an edge in a cycle. (5.8) implies that all edges in a cycle must be cuts between existing connected components. Therefore, $\mu_t(u_1) \neq \mu_t(v_1)$. Furthermore, there must be another edge in the cycle, $e_2 = \langle u_2, v_2\rangle$, such that $\mu_t(v_2) = \mu_t(u_1)$. It must also be true that $J_t(u_1, v_1) = J_t(u_2, v_2) = J_t^+(u_1, v_1) = 1$. By Lemma 8 all components in the cycle are unguarded. Therefore, applying (5.8) gives

$$w(e_1) - g_t(u_1) - g_t(v_1) \leq w(e_2) - g_t(u_2) - g_t(v_2).$$

In general, this inequality will hold for the incoming and outgoing edges of

any connected component in the cycle. Therefore, by transitivity,

$$
\begin{aligned}
w(e_1) - g_t(u_1) - g_t(v_1) &\leq w(e_2) - g_t(u_2) - g_t(v_2) \\
&\leq w(e_1) - g_t(u_1) - g_t(v_1),
\end{aligned}
$$

implying that

$$
w(e_1) - g_t(u_1) - g_t(v_1) = w(e_2) - g_t(u_2) - g_t(v_2).
$$

$\square$

Proposition 11. *The intersection graph of F_{t+1} formed from $\mathcal{C}_t$ is acyclic.*

Proof. Assume, on the contrary, that there is a round t during which a cycle of length ℓ is formed. Since the graph is simple, $\ell > 1$. By Lemma 9, all of the edges in the cycle must be of equal potential cost. Therefore, each connected component will have had a tie between two fringe edges which must have been broken using the edge ordering. Therefore, either $\ell = 1$ or there are two edges with the same unique identifier, both of which are contradictions. $\square$

Corollary 3. *$F_0, \ldots, F_\tau$ are all acyclic.*

Proof. Since $F_0 = \langle V, \emptyset \rangle$, the base case is acyclic. Induction over Proposition 11 then proves the corollary. $\square$

It is easy to see that $\tau = $ the diameter of the visibility graph $= O(n)$ since every acyclic subgraph has $O(n)$ edges and the algorithm adds at least one edge per round. This upper bound can in fact be tightened for many common cases, which we shall now demonstrate. Let $A_f(t) = A(t)$ be an upper bound on the number of unguarded components at the beginning of round t. Similarly, let $L_f(t) = L(t)$ be an upper bound on the total number of components at the beginning of round t. Clearly,

$$
\begin{aligned}
A(t) &\geq |\{C \in \mathcal{C}_t : f(C) = 1\}|, \text{ and} \\
L(t) &\geq |\mathcal{C}_t| \geq A(t).
\end{aligned}
$$

In general, every unguarded component will union with another component during each round. Regardless of whether such a component chooses to union with a guarded or unguarded component, the total number of components will decrease by one half the number of unguarded components. Therefore $L(t) = L(t-1) - A(t-1)/2$. Now let us consider the extrema for the change in the number of unguarded components. If all unguarded components choose to union with other unguarded components and all unions are pairwise, then we have $A(t) = A(t-1)/2$. On the other hand, if as many unguarded components union with guarded components as possible, then $A(t) \leq \min(A(t-1), L(t-$

$1) - A(t-1)$). Therefore, assuming pairwise unions, the general recurrences for $A(t)$ and $L(t)$ are:

$$A(t) = \max\left(\frac{A(t-1)}{2}, \min\left(A(t-1), L(t-1) - A(t-1)\right)\right), \qquad (5.12)$$

$$L(t) = L(t-1) - \frac{A(t-1)}{2}.$$

The initial conditions for the recurrences are clearly

$$
\begin{aligned}
A(0) &= |\{C \in \mathcal{C}_0 : f(C) = 1\}| = |R|, \\
L(0) &= |\mathcal{C}_0| = |R| + |T| = 2|R|.
\end{aligned}
$$

Claim 6. *$A(t-1)/2$ will always dominate in the maximization in* (5.12).

Validation of this claim will be given in the proof of the following proposition.

Proposition 12. *The algorithm will terminate after a logarithmic number of rounds if all component unions are pairwise (i.e., iterations of the main loop on line 9 of Algorithm 6): $\tau = O(\log n)$.*

Proof. This follows from the fact that the algorithm will terminate once the number of unguarded components is zero:

$$\forall t \in \mathbb{N}_0 : A(t) = 0 \implies t \geq \tau.$$

Therefore, the burden of this proof is to show that, the $A(t)$ recurrence will converge exponentially, implying that $\tau = O(\log n)$.

If Claim 6 holds, then it is clear that the $A(t)$ recurrence will converge exponentially:

$$
\begin{aligned}
A(t) &= \frac{A(0)}{2^t}, \\
L(t) &= 2|R| - \sum_{i=0}^{t} \frac{A(0)}{2^i}.
\end{aligned}
$$

Let $k = \frac{A(0)}{2|R|}$ and observe that $k = \frac{1}{2}$. Substituting $2k|R|$ for $A(0)$ ensures that the minimization in $A(t)$ will always evaluate to $L(t-1) - A(t-1)$

because

$$\forall t \in \mathbb{N}_0 : A(t) \ \geq \ L(t) - A(t).$$

$$2|R|\frac{k}{2^t} \ \geq \ 2|R| \left(1 - \left(\sum_{i=0}^{t} \frac{k}{2^i} \right) - \frac{k}{2^t} \right)$$

$$\frac{2k}{2^t} \ \geq \ 1 - \sum_{i=0}^{t} \frac{k}{2^i}$$

$$k \ \geq \ \frac{2^t}{1 + 2^{t+1}},$$

which is true because $2^t/(1 + 2^{t+1})$ is bounded above by $\frac{1}{2}$. Therefore, provided Claim 6 holds, (5.12) can be simplified to

$$A(t) = \max \left(\frac{A(t-1)}{2}, L(t-1) - A(t-1) \right).$$

Claim 6 obviously holds for the base case of $t = 1$ because $A(0)/2 = 2k|R|$ is bounded below by $L(0) - A(0) = 2|R| - \frac{2|R|}{2}$. Therefore, Claim 6 will hold as long as

$$\frac{A(t)}{2} \geq L(t) - A(t).$$

This equates to

$$k \ \geq \ 2^{t+1} \left(1 - \left(\sum_{i=0}^{t} \frac{k}{2^i} \right) - \frac{k}{2^t} \right)$$

$$\geq \ \frac{2 \times 4^t}{2^t + 4^{t+1}},$$

which must be true because $(2 \times 4^t)/(2^t + 4^{t+1})$ is bounded above by $\frac{1}{2}$. $\quad\square$

Claim 7. *Let t' be the round during which an edge $e = \langle u, v \rangle$ is added to the spanning forest. Then e will not be in the cut of any component in a subsequent round: $\forall t > t', C \in \mathcal{C}_t : e \notin \delta(C)$.*

Proof. $\mu_{t'+1}(u) = \mu_{t'+1}(v) = \mu_{t'}(u) \cup \mu_{t'}(v)$. Therefore, in all rounds subsequent to t' both endpoints of e are in the same component and therefore cannot be in the fringe. $\quad\square$

Proposition 13. *The vector y is a feasible solution to* (ART-D) *and has the property*

$$\sum_{e \in F_\tau} w(e) \leq \sum_{e \in F_\tau} \sum_{S : e \in \delta(S)} y_S.$$

Proof. The fact that y is a feasible solution to (ART-D) is a straightforward result of the fact that y is initially zero and is updated according to (5.9). Let t be the round during which an edge $e = \langle u, v \rangle \in F_\tau$ was added to the forest. From (5.10), note that

$$\left(g_t(u) = \sum_{i=0}^{t-1} y_{\mu_i(u)} \right) \bigwedge \left(g_t(v) = \sum_{i=0}^{t-1} y_{\mu_i(v)} \right).$$

Furthermore, at the beginning of round t the potential for e is $\varepsilon = w(e) - g_t(u) - g_t(v)$. Once e is added to F_t, the dual variables $y_{\mu_t(u)}$ and $y_{\mu_t(v)}$ are updated according to (5.9). Then there are three possible cases:

1. $f(\mu_t(u)) = f(\mu_t(v)) = J_t(u,v) = J_t(v,u) = 1$;

2. $f(\mu_t(u)) = f(\mu_t(v)) = J_t(u,v) + J_t(v,u) = 1$; or

3. $f(\mu_t(u)) + f(\mu_t(v)) = 1$.

In case 1,

$$y_{\mu_t(u)} + y_{\mu_t(v)} = \frac{\varepsilon}{1 + J_t(v,u)} + \frac{\varepsilon}{1 + J_t(u,v)} = \varepsilon,$$

implying that

$$w(e) = \sum_{i=0}^{t} \left(y_{\mu_i(u)} + y_{\mu_i(v)} \right). \tag{5.13}$$

For case 2, assume without loss of generality that $J_t(u,v) = 1$ and $J_t(v,u) = 0$. For case 3, assume without loss of generality that $f(\mu_t(u)) = 1$ and $f(\mu_t(v)) = 1$. Then for both of these cases note that

$$y_{\mu_t(u)} = \frac{\varepsilon}{1 + J_t(v,u)} = \varepsilon,$$

implying that

$$w(e) = y_{\mu_t(u)} + \sum_{i=0}^{t-1} \left(y_{\mu_i(u)} + y_{\mu_i(v)} \right). \tag{5.14}$$

Claim 7 implies that the summations in (5.13) and (5.14) comprise all sets that cut e, thus completing the proof. $\square$

5.3.3 Empirical Analysis

We have thus far proven that Algorithm 6 produces a feasible solution to both (ART-IP) and (ART-D) in a linear—and often logarithmic—number of rounds. It therefore only remains to analyze the quality of the solution.

A series of n-gons were randomly generated by connecting n uniformly distributed vertices in the unit square of the Cartesian plane according to the
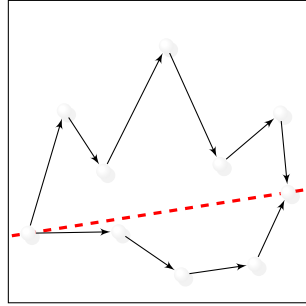
Figure 5.8: Illustration of the "Two Peasants" method of point set polygonization. The dashed line ("- - - -") divides the unit square into half-spaces (shaded "▬" and "▦") through the two extremal vertices in the x dimension. Arrows represent the addition of edges to the polygon, the direction of which indicate the order of their addition in the half-space.

folkloric "Two Peasants" method. This method works by dividing the plane into half-spaces via the line that passes through the two vertices of extremal value in the x dimension. The plane is then rotated such that the line between the extremal vertices is parallel to the x-axis. In each half-space, the vertices are connected to each other in order of increasing x value. This will result in a simple polygon. The method is depicted in Figure 5.8. 32 random polygons were created for each value of n. An agent was instantiated at each vertex of each randomly generated polygon and the algorithm run. The optimal dominating set was also calculated using an exhaustive sequential method.

Figure 5.9 presents the distribution of optimality as a function of polygon size. Values on the y-axis represent the constant of approximation; lower values are better, with 1.0 being the optimal solution. Boxes represent the second and third quartiles of each distribution. The overall mean constant of approximation is 3.13 with a standard deviation of 0.36. Therefore, we can say with high probability that the algorithm will produce a solution with a constant approximation bound regardless of the problem size.

5.3.4   Art Gallery Variants

In this section we will show that some variants of the art gallery problem can be solved using the same approach as described above. In fact, some harder problems can be approximated with a constant *theoretical* bound on solution quality. For example, one popular variant is what is dubbed the "Treasury Problem" [92], in which treasures dispersed in the polygon are what need to be guarded. As another variant, one might need to minimize the distance between a guard and that which he or she is guarding (*e.g.*, due to a limited
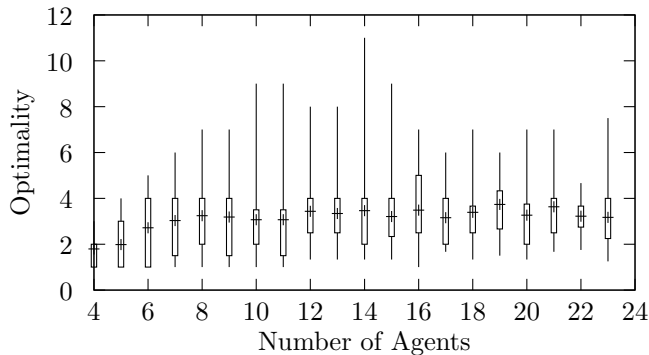
Figure 5.9: Solution quality of the algorithm for art gallery problems of various size. The x-axis is the size of the polygon (*i.e.*, the number of agents) and the y-axis is the constant of approximation. Each column is the distribution over 32 randomly generated polygons of a specific size. Boxes surround the middle two quartiles. The mean of each distribution is depicted as "+".

view distance of the sensor, or due to the mobility of a robot). This variant is in fact equivalent to the treasury problem in which each treasure has weighted importance [93].

In order to model this variant, we need only to embed the edge weights of the augmented graph into the proper metric space. As long as all of the edges in the augmented graph are weighted in a metric space with a normalized bijection in the range $\left[1, \frac{3}{2}\right]$ then we will show that the algorithm as defined above will produce a solution that is no more than a factor of $2 - \frac{2}{|R|}$ away from optimal. This can easily be done by parameterizing the relative cost between covering a vertex/treasure and the distance between a guard and a vertex/treasure. To prove this claim, we use a technique of defining an invariant over the weights of the edges added to the forest that can ultimately be bounded by the average vertex degree of the forest. The basic intuition of our result is that the average degree of a vertex in a forest of at most n vertices is at most $2 - \frac{2}{n}$. This technique is exactly the same as that first used in a proof due to Goemans and Williamson in [73, Theorem 3.6], in which they show that certain connectivity problems can be sequentially 2-approximated in polynomial time. Our result in fact generalizes that of Goemans and Williamson by proving that, with a slight change to the potential function (and thereby the invariant), the approximation guarantee can be maintained even if multiple edges are added per round (allowing for parallelism/distribution).

Lemma 10 (Williamson, *& al.* [14, Theorem 3.6])**.**
Let H be the intersection graph of the final spanning forest F_τ formed from

$\mathcal{C}_t$. *Remove all isolated vertices in H that correspond to components in $\mathcal{C}_t$ that are guarded. Then no leaf in H corresponds to a guarded component.*

Proof. This is a transcription of the proof, reproduced here for completeness using our notation in the specific domain of art gallery problems. Assume the contrary: Let v be a leaf, let C_v be its associated guarded component, let e be the edge incident to v, and let $C \subseteq V$ be the component of F which contains C_v. Let N and $C \setminus N$ be the two components formed by removing edge e from the edges of component C. Without loss of generality, say that $C_v \subseteq N$. The set $N \setminus C_v$ is partitioned by some of the components of the current round; call these $C_1, \ldots, C_k$. Since vertex v is a leaf, no edge in F_τ connects C_v to any C_i. Thus by the construction of F_τ, $\forall i \in \{1, \ldots, k\} : C_i$ is guarded. Since C_v is also guarded, it follows that N must be too. Clearly, if two components S and B are both guarded and $B \subseteq S$, then the component $S \setminus B$ must also be guarded. Since we know that C is guarded then $N \setminus C$ must as well, and thus by the construction of F_τ, $e \notin F_\tau$, which is a contradiction. □

Proposition 14. *The cost of the final spanning forest F_τ is bounded above by $\left(2 - \dfrac{2}{|R|}\right) Z^*_{ART\text{-}IP}$.*

Proof sketch. The basic intuition of our result is that the average degree of a vertex in a forest of at most n vertices is at most $2 - \frac{2}{n}$. *Cf.* the proof of Proposition 3 on page 34. □

Assuming all messages can be both unicast and broadcast in a constant number of messaging rounds then, by the same argument as in the proof of 12, the main loop of the distributed algorithm on line 9 can run in a logarithmic number of iterations and thereby will have a logarithmic number of messaging rounds. If this assumption does not hold—for example, if *ad hoc* routing is required—then the algorithm can be trivially extended to support the Broadcast-Message function itself. To do this, the algorithm will use the partially constructed spanning trees within each connected component for multicast.

The most expensive operations in the distributed algorithm are (1) determining the fringe edge with minimal potential; and (2) merging two connected components once an edge between them becomes tight. Should efficient broadcast be unavailable, one way of implementing these operations is to have broadcast messages convergecasted up the partially constructed spanning tree in the component. This method was used to solve a similar problem in [68]. The root of the tree (*e.g.*, the vertex that was added the earliest and is of highest unique identifier) can then perform the operation and unicast the result back down to the relevant fringe member(s).

5.4   Conclusion

This chapter has presented a number of examples of how the generalized multidirectional graph search algorithm and framework can be applied to a number of real world problems. Empirical results for these problems agree with the theoretical bounds outlined in Chapters 3 and Chapter 4. Ultimately, we have furthered the results of Panconesi [67] by showing that the primal-dual optimization scheme proposed by Goemans and Williamson [73] can be successfully distributed into an algorithm with not only bounds on approximation, but also on runtime. This suggests that other connectivity problems might yield to the same approach.

Chapter 6

Dynamic Agent Organizations

The ultimate problem we are working toward solving is that of *dynamic* multiagent organization. Ideally, algorithms should not have to completely reoptimize subsequent to every perturbation of the problem. Vertices can enter and leave the graph at any time, and edges can be re-weighted or removed. *Superstabilizing* distributed algorithms can start at any state and are guaranteed to eventually converge to a solution. Can multidirectional graph search be extended such that it is superstabilizing? Another reasonable question to ask is: What if the dynamic problem contains constraints that are not trivially representable using proper functions? Both of these questions will be answered in this chapter. In §6.1 we define extensions to multidirectional graph search to allow for superstability subject to dynamic topological updates. In §6.2, a new, distributed superstabilizing algorithm is introduced for the problem of pseudotree construction.

6.1 Online Topology Updates

Given a graph $G = \langle V, E \rangle$ with a proper function f, let $H = \langle \tilde{V}, \tilde{E} \rangle$ be a 2-optimal constrained forest discovered by multidirectional graph search (Algorithm 2). Note that the nature of the algorithm ensures that if there exists a feasible solution then, at the algorithm's completion in round τ, H will contain no active components. Now consider that during round $\tau + 1$ a new vertex, v, is added to the input graph: $G' = \langle V \cup \{v\}, E \cup \delta(\{v\}) \rangle$. Clearly, re-running Algorithm 2 from scratch on the new graph G' will find the new 2-optimal solution H'. What if, however, the algorithm is simply allowed to continue off from where it left in round τ? Is it possible to exploit

some of the computation from the previous execution of the algorithm on
G to find the new solution H' in *fewer* rounds? In certain cases this *is*
possible; the remainder of this section describes when and how, and gives a
superstabilizing extension to multidirectional graph search.

We will first identify under which circumstances the addition of v will
maintain dual feasibility. Let $\tau' > \tau$ be the new round during which the con-
tinued execution of the algorithm (after the addition of v) reaches quiescence.
If we simply let the algorithm continue running as normal after the addition
of vertex v in a round subsequent to τ, we need to show that whatever was
a feasible solution before the addition of v remains a feasible solution after
the addition of v. In order for dual feasibility to persist, the dual constraints
must remain satisfied:

$$\forall e \in E \cup \delta(\{v\}) : \sum_{S:e\in\delta(S)} y_S \leq w(e).$$

Since we know that these constraints hold at round τ (before the addition of
v) it is therefore sufficient to only consider the sets S that cut edges incident
to v. By definition, the sum of the dual variables that cut an edge equals
the slack remaining on that edge. Therefore, the dual constraints will be
satisfied if the weight of each of v's incident edges is greater than or equal to
the amount of slack in all of their incident component's fringe edges. This
scenario is depicted in Figure 6.1: For each vertex $u \in \delta(\{v\})$ neighboring v,
it must be true that each fringe edge $\langle i, j \rangle \in \delta(\mu_\tau(u))$ that is in the cut of u's
component has slack less than or equal to the weight of $\langle u, v \rangle$:

$$\forall \langle i, j \rangle \in \delta(\mu_\tau(u)) : w(\langle u, v \rangle) \geq g(i) + g(j).$$

This concept—which in effect defines under which circumstances Proposi-
tion 2 will remain true after the addition of a new edge—is formalized in the
following proposition.

Proposition 15. *If the weights of all of v's incident edges are greater than
or equal to the slack of all of their neighboring vertices' fringe nodes, i.e.,*

$$\forall e = \langle v, u \rangle \in \delta(\{v\}) : \Big(\forall \langle i, j \rangle \in \delta(\mu_\tau(u)) : w(e) \geq g_\tau(i) + g_\tau(j) \Big), \quad (6.1)$$

*then, after the algorithm has re-quiesced in round τ', the sum of the weights
of the edges in $H_{\tau'}$ is bounded above by the sum of the dual variables that cut
them,*

$$\sum_{e\in H_{\tau'}} w(e) \leq \sum_{e\in H_{\tau'}} \sum_{S:e\in\delta(S)} y_S, \quad (6.2)$$

and $H_{\tau'}$ is a feasible solution to (D). *In other words, Proposition 2 will still
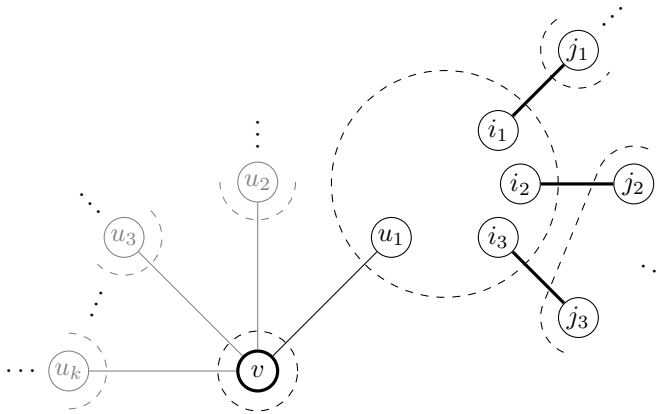hold.*

Figure 6.1: In order to maintain dual feasibility after the addition of v, for each u incident to v is must be true that the weight of $\langle u, v \rangle$ is greater than or equal to the slack of all of the other edges $\langle i, j \rangle$ in the cut of u's component (drawn in bold). The boundary of the cut with respect to each connected component is depicted with a dashed line. Only the edges that are across cuts are drawn.

Proof. We will first prove (6.2). If v is inactive, then none of v's incident edges will become tight and (6.2) will remain true. Therefore, let us consider the case when v is active. There must be at least one edge incident to v that will be added to H in order for v to become inactive. Assuming without loss of generality that a feasible solution to the new problem exists, v must have at least one new incident edge that is added to the problem: $\delta(\{v\}) \cap H' \neq \emptyset$. We now use induction. As a base case, we must show that Proposition 2 holds after the first addition of an edge in $\delta(\{v\})$ during round $\tau + 1$. To be precise, we need to show that

$$\sum_{e \in H_{\tau+1}} w(e) \leq \sum_{e \in H_{\tau+1}} \sum_{S : e \in \delta(S)} y_S. \tag{6.3}$$

From Proposition 2 we know that at the end of round τ the following must hold:

$$\sum_{e \in H_\tau} w(e) \leq \sum_{e \in H_\tau} \sum_{S : e \in \delta(S)} y_S.$$

To prove (6.3), it is therefore sufficient to show that the weight of a new edge $e \in \delta(\{v\})$ will not break this invariant:

$$w(e) \leq \sum_{S : e \in \delta(S)} y_S.$$

By line 11 of Algorithm 2,

$$
\begin{aligned}
\sum_{S:e\in\delta(S)} y_S &= y_{\mu_{\tau+1}(u)} + y_{\mu_{\tau+1}(v)} \\
&= \frac{\varepsilon}{1 + J_{\tau+1}(u,v)} + \frac{\varepsilon}{1 + J_{\tau+1}(v,u)} \\
&= \frac{\varepsilon}{1 + J_{\tau+1}(u,v)} + \frac{\varepsilon}{2} \\
&\geq \varepsilon \\
&= w(e) + g_{\tau+1}(u) + g_{\tau+1}(v) \\
&= w(e) + g_{\tau+1}(u),
\end{aligned}
$$

because $g_{\tau+1}(v)$ is initialized to zero and $J_{\tau+1}(u,v) \in \{0,1\}$. From the definition of (3.3),

$$
\begin{aligned}
w(e) + g_{\tau+1}(u) &= w(e) + \sum_{i=0}^{\tau} y_{\mu_i}(u) \\
&= w(e) + \sum_{S:e\in\delta(S\setminus\{v\})} y_S \\
&\geq w(e),
\end{aligned}
$$

which proves (6.3).

Now we shall show that $H_{\tau+1}$ remains a feasible solution to (D). Regardless of whether or not v is a terminal, the inductive hypothesis maintains that the dual constraints are satisfied by ensuring that the slack for all fringe edges does not exceed the edge's weight:

$$
\forall \langle v, u \rangle \in \delta(\{v\}):
$$

$$
\left(\forall e = \langle i, j \rangle \in \delta(\mu_{\tau+1}(u)) : w(e) \geq \sum_{S:e\in\delta(S)} y_S \geq g_{\tau+1}(i) + g_{\tau+1}(j) \right).
$$

This provides dual feasibility and, thereby, proves the inductive base case. The same logic as in the proof of Proposition 2 itself can then be used to prove the inductive case. □

Proposition 16. *If Proposition 15 holds for H' and all edge additions during the remaining $\tau' - \tau$ rounds are weighted in the range $\left[\tilde{\omega}, \frac{3}{2}\tilde{\omega}\right]$, then H' will also be 2-optimal.*

Proof. Proposition 15 ensures that the weight of $H_{\tau'}$ is

$$
\sum_{e\in H_{\tau'}} w(e) \leq \sum_{e\in H_{\tau'}} \sum_{S:e\in\delta(S)} y_S = \sum_{S\subset V\cup\{v\}} |H_{\tau'} \cap \delta(S)| y_S.
$$

The burden of this proof is therefore to show that, after the addition of v, the new constrained forest $H_{\tau'}$ is 2-optimal:

$$\sum_{S \subset V \cup \{v\}} |H_{\tau'} \cap \delta(S)| y_S \leq 2 \sum_{S \subset V \cup \{v\}} y_S. \tag{6.4}$$

We shall use induction to show that (6.4) is invariant over the construction of $H_\tau, H_{\tau+1} \ldots, H_{\tau'}$. Given that H_τ is 2-optimal, it must be true that

$$\sum_{S \subset V} y_S |H_\tau \cap \delta(S)| \leq 2 \sum_{S \subset V} y_S.$$

This proves the base case. The proof of the inductive case then follows the same logic as that of the proof of Proposition 3. $\qquad\square$

The following corollary is a direct consequence of Propositions 15 and 16:

Corollary 4. *A new vertex can be added to the problem at any time without breaking the guarantee of 2-optimality as long as its incident edges' weights are greater than or equal to the slack of all of their neighboring vertices' fringe nodes.*

Corollary 4 defines the circumstances in which new vertices may be dynamically added to the problem *without* having to restart the algorithm from scratch. Extending this capability to general multidirectional graph search (Algorithm 2) only requires a distributed means of determining if the conditions of Proposition 15 are met—namely, if (6.1) is satisfied—and a protocol for handling race conditions (*e.g.*, if two vertices are added at once). A modified version of Algorithm 2 that allows for dynamic addition of vertices is given in Algorithm 8.

Lines 6–22 of Algorithm 8 will only require worst-case $O(n)$ messages and linear local computation, because the number of fringe edges is always $O(n)$. Since the primary mechanism of Algorithm 8 is otherwise the same as in the original multidirectional graph search algorithm, the remainder of the computation will have similar efficiency bounds.

Note, however, line 12 of Algorithm 8; is it necessary to always have the algorithm restart from scratch if the conditions of Corollary 4 are not met? By recording the history of the search and performing backtracking, the answer turns out to be "no". In fact, we need only backtrack to the most recent round in which the conditions of Corollary 4 are met. In some cases this will require backtracking to round 0—in effect restarting the algorithm. More often than not, though, backtracking to round 0 will not be necessary. The added bookkeeping for this backtracking procedure is given in Algorithm 9.

Since the original algorithm is bounded by a linear number of communication rounds, $\tau = O(n)$, it stands that there can be at most $O(n)$ backtracks

Algorithm 8 The dynamic multidirectional graph search algorithm. The new code required for dynamic variable addition is emphasized; the original code from Algorithm 2 is grayed out.

1: **procedure** Dynamic-Multidirectional-Graph-Search(T, v, w, δ)
Require: T is the set of terminals. $v \in T$ is the terminal running this instance of the search algorithm. w is a function that maps edges to their associated weight in the metric space $\left[\tilde{\omega}, \frac{3}{2}\tilde{\omega}\right] \in \mathbb{Q}$. δ is a successor function such that $\delta(S)$ is the set of edges having exactly one endpoint in S.
Ensure: $H = \langle \tilde{V}, \tilde{E} \rangle$ is the resulting forest.
2:     $\tilde{V} \leftarrow \{v\}$ /* The initial solution has just our vertex. . . */
3:     $\tilde{E} \leftarrow \emptyset$ /* . . .and no edges */
4:     $F \leftarrow \delta(\{v\})$ /* The fringe of our search, initialized to v's incident edges */
5:     $\forall u \in V : g(v) \leftarrow 0$ /* Initialize the path-cost function, implicitly setting $y_S \leftarrow 0$ for all $S \subset V$ */
6:     Broadcast $\delta(\{v\})$ to our neighbors.
7:     Wait for acknowledgement from our neighbors' components that it is okay to proceed. /* *e.g.*, using Paxos [94] */
8:     $valid \leftarrow$ True
9:     **for all** $\langle u, v \rangle \in \delta(\{v\})$ **do**
10:         $F_u \leftarrow \delta(\mu(u))$ /* listen for and/or request the fringe edges of u's component */
11:         **if** $\exists \langle i, j \rangle \in F_u : g(i) + g(j) > w(\langle u, v \rangle)$ **then** /* Check if the conditions of Corollary 4 are met */
12:             $valid \leftarrow$ False /* v cannot be added unless the algorithm is restarted from scratch */
13:         **end if**
14:     **end for**
15:     **for all** $u \in V$ that are neighboring v **do**
16:         **if** $valid$ **then**
17:             Acknowledge receipt of F_u.
18:         **else**
19:             Tell u that the algorithm needs to be restarted.
20:             Dynamic-Multidirectional-Graph-Search(T, v, w, δ) /* restart our instance of the algorithm */
21:         **end if**
22:     **end for**
23:     **while** $(\tilde{V} \cap T \neq T) \wedge (F \neq \emptyset)$ **do** /* while H does not contain all terminals and the fringe is not empty */
24:         **while** there are pending requests for our fringe edges **do** /* from line 10 */
25:             Send $F_v = \{\langle i, j, g(i), g(j) \rangle : \langle i, j \rangle \in F\}$ to the requesting agent u.
26:             Wait for an acknowledgement from u.
27:         **end while**
28:         Find an edge in $e = \langle v, u \rangle \in F$ such that $\varepsilon = w(e) - g(u) - g(v)$ is minimized.
29:         **if** u either is being or already was expanded by another search **then**
30:             Union $\tilde{V}$, $\tilde{E}$, F, and g with the respective data structures of the search that already expanded u and then merge our execution with that search.
31:             **if** The other search also expanded the edge $\langle v, u \rangle$ this round **then**
32:                 $\varepsilon \leftarrow \frac{\varepsilon}{2}$
33:             **end if**
34:         **end if**
35:         **for all** $k \in \tilde{V} : k$ is incident to an edge in the fringe **do**
36:             $g(k) \leftarrow g(k) + \varepsilon$ /* Implicitly set $y_{\tilde{V}} \leftarrow y_{\tilde{V}} + \varepsilon$ */
37:         **end for**
38:         $F \leftarrow (F \setminus \{e\}) \cup \delta(\{u\})$ /* Remove e and add the edges incident to u */
39:         $\tilde{V} \leftarrow \tilde{V} \cup \{u\}$
40:         $\tilde{E} \leftarrow \tilde{E} \cup \{e\}$
41:     **end while**
42: **end procedure**

Algorithm 9 The dynamic multidirectional graph search algorithm with backtracking. The new code required for backtracking is emphasized; the original code from Algorithm 8 is grayed out.

1: **procedure** Dynamic-Multidirectional-Graph-Search-Backtracking(T, v, w, δ)

Require: T is the set of terminals. $v \in T$ is the terminal running this instance of the search algorithm. w is a function that maps edges to their associated weight in the metric space $[\tilde{\omega}, \frac{3}{2}\tilde{\omega}] \in \mathbb{Q}$. δ is a successor function such that $\delta(S)$ is the set of edges having exactly one endpoint in S.

2–14: Lines 2–14 are unchanged from Algorithm 8.

15: **for all** $u \in V$ that are neighboring v **do**

16:    **if** *valid* **then**

17:      Acknowledge receipt of F_u.

18:    **else**

19:      Tell u ~~that the algorithm needs to be restarted~~ to backtrack.

20:      Dynamic-Multidirectional-Graph-Search(T, v, w, δ) /* restart our instance of the algorithm */

21:    **end if**

22: **end for**

23: $\mathcal{A} = \{\langle \tilde{V}, \tilde{E}, F, g, \texttt{null}, \texttt{null}, \texttt{null}, \texttt{null}, \texttt{null}\rangle\}$ /* a stack for the history of the algorithm */

24: **while** $(\tilde{V} \cap T \neq T) \wedge (F \neq \emptyset)$ **do** /* while H does not contain all terminals and the fringe is not empty */

25:    **while** there are pending requests for our fringe edges **do** /* from line 10 */

26:      Send $F_v = \{\langle i, j, g(i), g(j)\rangle : \langle i, j\rangle \in F\}$ to the requesting agent u.

27:      Wait for an acknowledgement from u.

28:      **if** u tells us to backtrack **then**

29:        $\langle \tilde{V}, \tilde{E}, F, g, v_2, \tilde{V}_2, \tilde{E}_2, F_2, g_2\rangle \leftarrow \texttt{Stack-Pop}(\mathcal{A})$

30:        **if** $v_2 \neq \texttt{null}$ **then**

31:          Split our search with agent v_2, giving it the state $\tilde{V}_2, \tilde{E}_2, F_2, g_2$.

32:        **end if**

33:      **end if**

34:    **end while**

35:    Find an edge in $e = \langle v, u\rangle \in F$ such that $\varepsilon = w(e) - g(u) - g(v)$ is minimized.

36:    **if** u either is being or already was expanded by another search **then**

37:      $\texttt{Stack-Push}(\mathcal{A}, \langle \tilde{V}, \tilde{E}, F, g, u, \tilde{V}_u, \tilde{E}_u, F_u, g_u\rangle)$

38:      Union $\tilde{V}$, $\tilde{E}$, F, and g with the respective data structures of the search that already expanded u and then merge our execution with that search.

39:      **if** The other search also expanded the edge $\langle v, u\rangle$ this round **then**

40:        $\varepsilon \leftarrow \frac{\varepsilon}{2}$

41:      **end if**

42:    **end if**

43:    **for all** $k \in \tilde{V} : k$ is incident to an edge in the fringe **do**

44:      $g(k) \leftarrow g(k) + \varepsilon$ /* Implicitly set $y_{\tilde{V}} \leftarrow y_{\tilde{V}} + \varepsilon$ */

45:    **end for**

46:    $F \leftarrow (F \setminus \{e\}) \cup \delta(\{u\})$ /* Remove e and add the edges incident to u */

47:    $\tilde{V} \leftarrow \tilde{V} \cup \{u\}$

48:    $\tilde{E} \leftarrow \tilde{E} \cup \{e\}$

49: **end while**

50: **end procedure**

---

for the addition of a single new vertex. Since each backtrack only incurs a constant number of extra messaging rounds, Algorithm 9 will have the same asymptotic runtime bounds as the original algorithm. Furthermore, the only new data structure employed in the algorithm is the stack $\mathcal{A}$. The size of each element in the stack is $O(|V|^2)$ and there are at most τ elements in the stack at any time. Therefore, the additional memory overhead of the new algorithm is still polynomial: $|\mathcal{A}| = O(\tau |V|^2) = O(n^3)$.

With the backtracking technique of Algorithm 9, we also gain the ability to delete vertices and update edge weights. To see this, we will first prove a rather intuitive proposition.

Proposition 17. *As long as a feasible solution still exists after its deletion, any vertex that does not have any incident edges in the constrained forest can be deleted from the problem while maintaining 2-optimality.*

Proof. Let v be the node that is to be deleted. Since the conditions of the proposition require that a feasible solution still exists after the removal of v, we only need to consider how v's deletion affects 2-optimality (Proposition 16). Since none of v's incident edges are in the constrained forest, $\sum_{e \in H_{\tau+1}} w(e)$ remains unchanged after the removal of v. Therefore, the invariant used to prove 2-optimality in Proposition 16 remains unchanged, and the final solution must also be 2-optimal. $\square$

By this property, we need only backtrack to the most recent round during which v had no incident edges in the constrained forest, at which point it can be safely removed. Therefore, updating edges can simply be implemented by removing all incident vertices, adding/removing/updating the edges, and re-adding the affected vertices.

In this section we have identified the sufficient conditions under which new vertices may be added/deleted and edges modified in an existing constrained forest. Algorithm 8 implements these processes, with no affect to the asymptotic efficiency bounds of the original multidirectional graph search algorithm. Furthermore, if the conditions for these processes to take place are not met, we have created a backtracking mechanism by which any such dynamic modification to the constrained forest can occur. The backtracking mechanism is given in Algorithm 9, and only increases the memory overhead of the algorithm polynomially.

6.2   Pseudotree Construction

A very reasonable question to ask is: What if a constrained forest problem contains constraints that are not representable using proper functions or their extensions? A prominent example of this is the *pseudotree construction problem* [29, 30]. Recall from §1.1.5 that a valid pseudotree inherently has the

property that each pair of neighboring agents in the interaction graph are either ancestors or descendants of each other in the hierarchy. This ensures that no interaction will necessarily occur between agents in disjoint subtrees. Therefore, interactions in disjoint subtrees may occur in parallel. The general reason why this problem is not representable using a proper function is that (1) the pseudotree is rooted, and (2) a global invariant must be maintained over the ancestor/descendant relationships in the tree; this will be described in detail below.

Given a graph of expected interaction between the agents, this section introduces an algorithm, called Multiagent Organization with Bounded Edit Distance (Mobed), for constructing and maintaining an organizational hierarchy that is a valid pseudotree. It is shown that Mobed is correct and that it outperforms alternative approaches by as much as 300% in terms of edit distance between perturbations with little impact to computation and privacy.

6.2.1 The Mobed Algorithm

Let $G = \langle A, E \rangle$ be a graph consisting of an ordered set of agents, A, and a set of edges E. Each edge $\langle a_i, a_j \rangle \in E$ implies that agent a_i will need to interact with a_j. We shall hereafter call this the *interaction graph*. Let $N : A \to 2^A$ be a function mapping agents to their (open[1]) *neighborhood*; $N(a_i)$ returns the set of all agents that share an edge with a_i. In terms of the δ function that has been used up until this point, $N(a_i)$ is equivalent to $\{a_j \in A : \langle a_i, a_j \rangle \in \delta(\{a_i\})\}$. A *multiagent hierarchy* for a given graph $G = \langle A, E \rangle$ is an unordered, labeled, rooted tree denoted by the tuple $T = \langle A, \pi : A \to A \rangle$, where π is a function mapping agents to their parent in the tree. The inverse of the parent function, $\pi^{-1}(a_i)$, shall be used to denote the set of children of agent a_i. The notation "R^+" shall be used to represent the transitive closure of a binary relation R. Agent a_j is said to be the *ancestor* of an agent a_i in a hierarchy if a_j has the property $(a_j = \pi(a_i))^+$. Likewise, agent a_j is a *descendant* of a_i if $(a_j \in \pi^{-1}(a_i))^+$. For convenience—and at the expense of a slight abuse of notation—let C_i be the set of ancestors of agent a_i and let D_i be the set of descendants of agent a_i. The *depth* of an agent in the hierarchy is the number of edges in the shortest path from that agent to the root (which also happens to be equal to the number of the agent's ancestors: $|C_i|$). A hierarchy is said to be *valid* if all neighboring pairs of agents in the interaction graph are either ancestors or descendants of each other in the hierarchy. Let $\nu : 2^{A \times A} \to \mathbb{B}$ be a validity testing function defined as:

$$\nu(I) \mapsto \left(\forall \langle a_i, a_j \rangle \in I : a_i \in (C_j \cup D_j) \right). \tag{6.5}$$

---

[1] A neighborhood is "closed" if it also contains a_i.

Given an interaction graph $G = \langle A, E \rangle$, a multiagent hierarchy T is therefore valid for a given problem if $\nu(E) = \text{TRUE}$.

We assume that each agent a_i knows the existence of all of its neighbors: $a_j \in N(a_i)$, however, it may not know of *all* of the other agents in the network. Each agent that has already been placed in the hierarchy only knows its parents, children, and interaction graph neighbors. Agents also know the relative location of interaction graph neighbors (*i.e.*, ancestor or descendant).

As Mobed applies to DCR, it should be noted that our notion of an interaction graph can be equated to DCR's notion of a constraint graph. In this sense, though, our formalization is then in the context of constraint graphs of *agents* as opposed to constraint graphs of *variables* (the latter of which is the norm for DCR). This presentation was chosen for sake of both brevity and accessibility. Nothing precludes this algorithm from being applied to constraint graphs with multiple variables per agent; in such a case the work herein may be read such that "agents" are instead "variables."

With these assumptions in mind, there are a number of challenges in devising a DynDisMHG algorithm (complicated by the fact that there is no central server and agents act asynchronously in an asynchronous network):

1. To what extent can privacy be maintained?

2. What if a new agent has at least two neighbors in disjoint hierarchies?

3. How are multiple, concurrent requests for addition and removal handled?

4. How is the hierarchy initialized?

5. To what extent can the perturbation of an existing hierarchy be minimized subsequent to the addition or removal of an agent?

The remainder section introduces Mobed: an algorithm that addresses all of these challenges.

The Insertion Point

Given an existing hierarchy and a new agent, the first problem is to determine where in the hierarchy that agent should be added such that the hierarchy remains valid. We shall now propose and prove a series of lemmas that define such an insertion point. We first define (6.6) that tests whether or not a given agent already in the hierarchy is a valid insertion point. In Lemmas 11 and 12 we prove, respectively, that such an insertion point must exist and that it must be unique. In Lemmas 13 and 14 we prove that the new agent can be inserted either as the parent or child of the insertion point.

Let $\eta : A \times 2^A \to \mathbb{B}$ be a function that identifies the valid points in a hierarchy in which a new agent can be added. The η function is defined such that $\eta(a_i; I)$ is TRUE if and only if all of the following are true:

- a_i is an ancestor or descendant of all agents in $I - \{a_i\}$;

- a_i either has no descendants in I or a_i has more than one child whose subtree has agents in I; and

- none of a_i's ancestors are insertion points for I.

$$\eta(a_i; I) = v\Big(\{a_i\} \times (I \setminus \{a_i\})\Big)$$

$$\wedge \left(D_i \cap I = \emptyset \vee \left| \left\{ a_j \in \pi^{-1}(a_i) : (D_j \cup \{a_j\}) \cap I \neq \emptyset \right\} \right| > 1 \right)$$

$$\wedge \Big(\forall a_j \in C_i : \neg\eta(a_j; I) \Big). \quad (6.6)$$

We shall hereafter refer to an agent a_i that satisfies $\eta(a_i; I)$ as an *insertion point* for the set I.

If an existing hierarchy is valid, it occurs that there must exist an insertion point for the neighborhood of each agent that is to be added. In fact, the insertion point for each new agent must be unique. We will prove these two properties in the following two lemmas.

Lemma 11. *Given a valid hierarchy $T = \langle A, \pi \rangle$ and an agent $a_i \notin A$, if all of a_i's neighbors are already in the hierarchy then there must exist an agent in the hierarchy, a_ℓ that is a valid insertion point for a_i:*

$$N(a_i) \subseteq A \implies \Big(\exists a_\ell \in A : \eta\left(a_\ell; N(a_i)\right) = \text{TRUE} \Big). \quad (6.7)$$

Proof. Let us assume, on the contrary, that $N(a_i) \subseteq A$ but there does not exist an agent a_ℓ such that $\eta(a_\ell; N(a_i))$. This means that either $\forall a_j \in A :$ $N(a_i) \setminus \{a_j\} \nsubseteq D_j \cup C_j$; every agent has exactly one child whose subtree contains an agent in $N(a_i)$; or every agent has at least one ancestor that is an insertion point for $N(a_i)$. The first case contradicts $N(a_i) \subseteq A$. The second case implies that the hierarchy T is cyclic (and therefore invalid) which is a contradiction. The third case either means that a distinct a_ℓ must exist or it means that the hierarchy T is cyclic, both of which are contradictions. $\square$

Lemma 12. *The insertion point of a valid hierarchy must be unique.*

Proof. Let us assume, on the contrary, that there are at least two agents in A that satisfy the existential quantification of a_ℓ in (6.7); let us call two such agents a_1 and a_2. Both $\eta(a_1; N(a_i))$ and $\eta(a_2; N(a_i))$ must be TRUE, which

implies that neither a_1 nor a_2 is an ancestor of the other, further implying that a_1 and a_2 must be in disjoint subtrees. Since the hierarchy is valid it must not be cyclic, and there must be some agent a_3 that is the deepest common ancestor of a_1 and a_2. Since a_1 and a_2 are both insertion points for $N(a_i)$, all of the agents in $N(a_i) \setminus \{a_3\}$ must be in C_3, which by definition of (6.6) means that $\eta(a_3; N(a_i))$ must be TRUE, contradicting the fact that both a_1 and a_2 are insertion points for $N(a_i)$. □

Now that we have established that a unique insertion point must exist, a new question is raised: In what way can the insertion point be used to incorporate the new agent into the existing hierarchy? As we shall expose in the following lemma, the new agent may always be inserted as the new parent of the insertion point.

Lemma 13. *Given a valid hierarchy* $T = \langle A, \pi \rangle$, *the addition of a new agent* $a_i \notin A$ *inserted between* $a_\ell \in A$ *and* $\pi(a_\ell)$ *will produce a valid new hierarchy,*

$$T' = \langle A \cup \{a_i\}, (\pi \setminus \{\langle \pi(a_i), a_i \rangle\}) \cup \{\langle a_\ell, a_i \rangle, \langle \pi(a_i), a_\ell \rangle\} \rangle,$$

if a_ℓ *is a valid insertion point:* $\eta(a_\ell; N(a_i))$.

Proof. This will clearly be true when a_ℓ is the root of the hierarchy, since the addition of a new agent at the root of the hierarchy will always be valid because the root is the ancestor of all other agents. For all other cases, note that a_ℓ will be the only child of a_i, thus a_i will share all of a_ℓ's previous ancestors and descendants. Since a_ℓ was already a valid insertion point, a_i must also remain valid. □

Let $a_d \in A$ be the deepest agent in the hierarchy that is either an ancestor or descendant of all of a_i's interaction graph neighbors: $\nu(\{a_d\} \times N(a_i)) \mapsto$ TRUE. Therefore, a_i can always be validly added as the parent of a_d or any of a_d's ancestors. This leads to the relatively naïve algorithm of always adding a new agent as the parent of its associated a_d. While this technique will always provide valid hierarchies, it will also always produce chains which is the worst case in terms of parallelism. We therefore need some way imposing branches in the tree. Under certain circumstances the new agent may be added as a new leaf underneath the insertion point. The specifics of these circumstances are expounded in the following lemma.

Lemma 14. *Given a valid hierarchy* $T = \langle A, \pi \rangle$, *the addition of new agent* $a_i \notin A$ *as a child of* $a_\ell \in A$ *will produce a valid new hierarchy* $T' = \langle A \cup \{a_i\}, \pi \cup \{\langle a_i, a_\ell \rangle\} \rangle$ *if* $N(a_i) \cap D_\ell = \emptyset$ *and* a_ℓ *is the insertion point for* $N(a_i)$.

Proof. Assume, on the contrary, that the addition of a_i as a child of a_ℓ will produce an invalid hierarchy. By definition in (6.5), this means that there is at least one pair of neighboring agents that are neither ancestors nor descendants

of each other. Since the original hierarchy T was valid, we know that such a pair of agents must be in $\{a_i\} \times N(a_i)$. Since $\nu(\{a_\ell\} \times N(a_i))$ is true, we know that all of a_i's interaction graph neighbors are either ancestors or descendants of a_ℓ. Since a_i is added as a child of a_ℓ and therefore shares all of a_ℓ's ancestors, it must be true that $\exists a_j \in N(a_i) : a_j \in D_\ell$, which contradicts $N(a_i) \cap D_\ell = \emptyset$. $\qquad\square$

General Principles

Based upon the results of the previous section, Mobed adds an agent a_i to an existing hierarchy by the following procedure:

  1: find a valid insertion point a_ℓ /* one must exist according to Lemmas 11 and 12 */
  2: **if** $D_\ell \cap N(a_i) = \emptyset$ **then** /* Lemma 14 */
  3:    $\pi(a_i) \leftarrow a_\ell$ /* add a_i as a new leaf under a_ℓ */
  4: **else** /* Lemma 13 */
  5:    $\pi(a_i) \leftarrow \pi(a_\ell)$ /* insert a_i between a_ℓ's parent... */
  6:    $\pi(a_\ell) \leftarrow a_i$ /* ...and a_ℓ */
  7: **end if**

Determining a_ℓ (step one in the procedure) can be performed by recursively passing messages up the tree starting from all agents in $N(a_i)$. Without loss of generality, let us assume that all agents in $N(a_i)$ are already present in the same hierarchy; this assures that there must be some agent a_j that will eventually receive $|N(a_i)|$ messages from its children. Then,

  1: **while** a_j received exactly one message regarding the addition of a_i **do**
  2:    $a_j \leftarrow$ the child from which a_j received the message
  3: **end while**
  4: $a_\ell \leftarrow a_j$

Using this method it is trivial to check whether $D_\ell \cap N(a_i) = \emptyset$ (step two in the procedure): If a_ℓ did not receive any messages from its children then we know $D_\ell \cap N(a_i) = \emptyset$ is true. Figure 6.2 provides an example execution of this algorithm.

Merging Hierarchies

We shall now consider the case when a new agent's neighborhood contains agents in disjoint hierarchies. This may happen if a_i is an articulation point of the interaction graph. The approach for this case is simply to add a_i as the new parent of the roots of T_1 and T_2. The problem, however, is that no agent in $N(a_i)$ necessarily knows that they are in disjoint hierarchies and the addition process as described above will deadlock.

    The solution is as follows: Whenever a root of a hierarchy (*e.g.*, a_1 and a_3) receives an addition request regarding a new agent a_i—regardless of whether
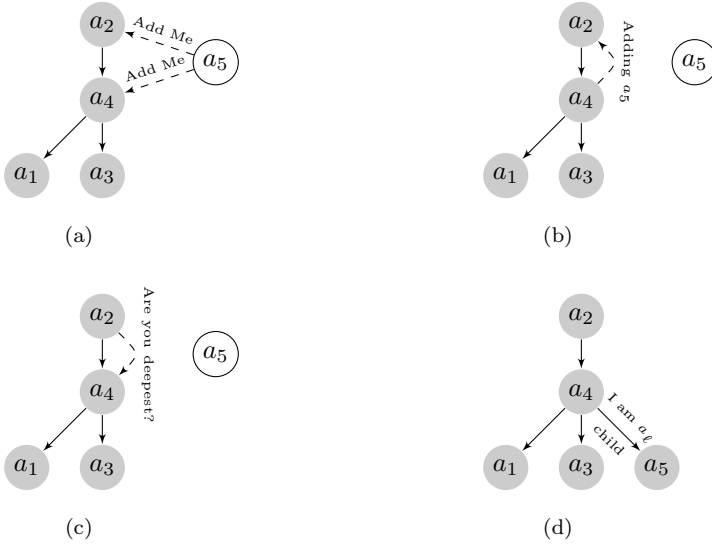
Figure 6.2: An execution of the algorithm for addition of a agent a_5 to an existing hierarchy. Solid edges represent edges in the existing hierarchy. Dashed edges represent message passing. In **(a)**, a_5 initiates its addition to the existing hierarchy by sending messages to all $a_j \in N(a_5)$. In **(b)**, a_4 forwards the message on to its parent, a_2. In **(d)**, a_2 receives two messages regarding the addition of a_5 and $|N(a_5)| = 2$, however, a_2 has a single child, so a_2 asks that child if it is deepest. Finally, in **(d)**, a_4 did not receive any messages from its children ($D_4 \cap N(a_5) = \emptyset$), so a_2 adds a_5 as a child.

that addition request was sent directly to the root or whether it was propagated up the hierarchy—and that root has received fewer than $|N(a_i)|$ such addition requests, then that root will additionally send a `Root` message to a_i stating that it is the root of a hierarchy. If a_i ever receives $|N(a_i)|$ such messages then a_i will become the new parent of all agents from whom it received the messages.

Preventing Race Conditions

The algorithm as it is presented above will work correctly if there is exactly one agent being added at a time. Due to the possibility of arbitrary message latency, there is a chance that concurrent additions could result in inconsistent modifications to the hierarchy among the agents. To address this we introduce a concept of *engaged blocks* of the hierarchy. When an agent a_j receives an add request regarding a new agent a_i, then a_j goes into *engaged* mode and proceeds as normal. If a_j is already engaged, however, it will immediately reply to a_i with an `AlreadyEngaged` message. Such an error condition implies that another agent in the subtree rooted at a_j (or an agent on the path from a_j to the root) is in the process of either being added or removed; we shall call this agent a_k. Agent a_j will also send an `AlreadyEngaged` message to a_k. This process is depicted in Figure 6.3. These messages contain a field stating whether $k > i$. If an agent ever receives such a message it will inform all agents in its neighborhood that it is canceling its addition. If the agent's identifier is lower priority than the other sent in the `AlreadyEngaged` message then that agent will perform an exponential backoff before restarting its addition. Otherwise, the agent will only sleep for a constant time period. Once the algorithm is complete, all agents that received an addition message regarding a_i become unengaged.

Initial Generation

The special cases addressed in the previous sections are sufficient to generate and maintain hierarchies *as long as* a hierarchy already exists. How should the initial hierarchy be generated? Which agent should become the first root?

The solution is to construct the initial hierarchy semi-synchronously: an agent will not attempt addition of itself to a hierarchy until all of its higher-priority neighbors are already members of a valid hierarchy. A new agent to be added to the hierarchy, a_i, will send an `AddMe` to all $a_j \in N(a_i)$, as described above, however, a_i will send them one-by-one in order of decreasing priority of j. After each `AddMe` is sent, a_i will block until receipt of a reply from a_j before proceeding to the next, lower-priority neighbor. The neighbor's reply can be one of three possibilities: an `AlreadyEngaged` message as described in §6.2.1, a `NoTree` message (meaning the neighbor has not yet been added to a tree), or a `AdditionStarted` message (meaning that the neighbor does have a tree
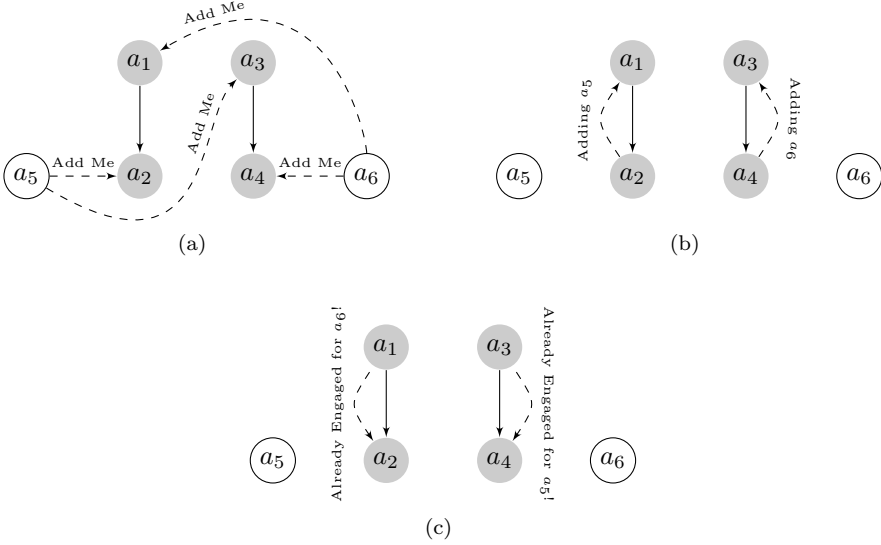
Figure 6.3: Handling of race conditions using engaged blocks. In **(a)**, agents a_5 and a_6 concurrently initiate their addition into existing hierarchies; a_1 becomes engaged for a_6 and a_3 becomes engaged for a_5. In **(b)**, agents a_2 and a_4 proceed with the algorithm as normal, propagating the add requests to their parents. Finally, in **(c)** agents a_1 and a_3 both reply with errors which are propagated back to a_5 and a_6 who perform a backoff before attempting to add themselves again. Without the use of engaged blocks, note that agents a_5 and a_6 will compete to be the root of the tree (*i.e.*, the parent of a_1 and a_3), possibly causing an inconsistency in the nodes' perception of the topology.

and has started the addition process for a_i as described above). If a_i receives a `NoTree` message from a_j and $j > i$ then a_i will send a `CancelSearch` message to all $a_k \in N(a_i)$ where $k > j$ and a_i will block until it receives an `AddRequest` message from another agent. If, on the other hand, a `NoTree` message is received from an a_j where $j < i$ then it is ignored. The addition will then proceed as in §6.2.1. Pseudocode for the entire algorithm—implementing the constructs in §6.2.1–6.2.1—is given in Algorithm 10.

Changes to Constraints and Agent Removal

Changes to constraints can be handled by removing and re-adding all affected agents. Removal of an agent a_i can be accomplished by making a_i, $\pi(a_i)$, and all $a_j \in \pi^{-1}(a_i)$ engaged. All of the children are then made the children of $\pi(a_i)$ and a_i is removed. The hierarchy will remain valid.

6.2.2   Analysis

This section analyzes—both theoretically and empirically—the performance of Mobed. For the empirical analysis, a series of random, connected graphs were randomly generated from a uniform distribution with varying numbers of vertices and edge densities[2]. A set of 100 random graphs were generated for each pair of number of vertices and edge density, for each of which both DFS and Mobed were run to generate a valid hierarchy. A new vertex was then randomly added in such a way as to maintain the given edge density, and both DFS and Mobed were then re-run to produce a new valid hierarchy. Various metrics including the average number of rounds of computation (*i.e.*, the length of the longest causal chain of synchronous messages required for the algorithm to reach quiescence) and the edit distance between hierarchy perturbations were recorded.

Computational Complexity

Let us consider the worst-case computational complexity of adding a new agent a_i to an existing valid hierarchy $T = \langle A_T, \pi \rangle$. We assume that $N(a_i) \subseteq A_T$ and there are no other agents attempting to be concurrently added. The agent will first send one message to each neighbor in $N(a_i)$. Each $a_j \in N(a_i)$ will then forward the message up the tree. In the worst case in terms of synchrony, each of the $|N(a_i)|$ messages will have to traverse up the tree to the root, followed by the root propagating a single message back down to the insertion point. The addition of a single agent therefore requires worst case $O(|N(a_i)|d_T)$ rounds, where d_T is the depth of T. In the worst case in

---

[2]Edge density, ranging in $[0, 1]$, is the percentage of possible edges that exist in the graph. A density of 0.0 means that the graph has no edges while a density of 1.0 means that the graph is complete.

Algorithm 10 Mobed's distributed addition of a new agent, regardless of whether or not any of the agent's neighbors are already in hierarchies or whether those hierarchies are disjoint. Message handlers are given in Algorithm 11.

Note that the t argument is used as a type of logical clock to avoid processing messages that have expired. If a message is ever received with a t value that is lower than any other message that has been received from the sender then the message is discarded.

1: **procedure** ADD-AGENT($a_i, t = 0$)
Require: a_i is the agent to be added. t is a counter for this addition attempt, initially set to zero.
2:     $H \leftarrow N(a_i)$
3:     **for all** $a_j \in N(a_i)$ in order of descending j **do**
4:         SEND-MESSAGE(HasTree?) to a_j
5:         wait for a reply from a_j
6:         **if** the reply is AlreadyEngaged **then**
7:             handle as described in §6.2.1.
8:         **else if** the reply is NoTree **then**
9:             **if** $j > i$ **then**
10:                send a CancelSearch(t) message to all $a_m \in N(a_i)$ where $a_m > a_j$.
11:                wait to receive an AddRequest message
12:                **return** ADD-AGENT($a_i, t + 1$)
13:            **else**
14:                $H \leftarrow H \setminus \{a_j\}$
15:            **end if**
16:        **else if** the reply is a success **then**
17:            do nothing
18:        **end if**
19:    **end for**
20:    **for all** $a_j \in H$ **do**
21:        SEND-MESSAGE(AddMe, $a_i, |H|, t$) to a_j
22:    **end for**
23:    $R \leftarrow \emptyset$ /* R is a set and therefore does not allow duplicates */
24:    **while** $|R| < |H|$ **do**
25:        **if** the next message is a Root message from a_j **then**
26:            $R \leftarrow R \cup \{a_j\}$
27:            **if** $|R| = |H|$ **then**   /* $|N(a_i)|$ contains agents in disjoint hierarchies */
28:                **for all** $a_r \in R$ **do**
29:                    $\pi(a_r) \leftarrow a_r$ /* Become the new parent of a_r */
30:                    Tell a_r that we are its new parent and that it can become unengaged.
31:                **end for**
32:            **end if**
33:        **else if** the next message is Added(p, c) from a_j **then**
34:            $\pi(a_i) \leftarrow p$ /* our new parent is p */
35:            $\pi(c) \leftarrow a_i$ /* our new child is c */
36:            $R \leftarrow H$
37:        **end if**
38:    **end while**
39:    send an AddRequest to all lower-priority agents to whom a_i has ever sent a NoTree message.
40: **end procedure**

Algorithm 11 Message handlers for agent addition.

1: **procedure** HANDLE-ADDME(a_n, q, t) sent from a_j
Require: a_n is the agent requesting to be added and a_i is the agent that received (and is
  processing) the message. $m : V \to \mathbb{N}$, $r : V \to 2^{\pi^{-1}(a_i)}$, and $s : V \to \mathbb{B}$ are all maps
  retained in memory. m maps variables to an integer, r maps variables to the power set of
  a_i's children, and s maps variables to a boolean. If a key does not exist in s then its value is
  taken to be FALSE. a_v is the agent for whom a_i is currently engaged, or $\emptyset$ if a_i is unengaged.
2:     **if** a_i is not yet in the hierarchy **then**
3:         SEND-MESSAGE(NoTree, t) to a_n
4:         **return**
5:     **else if** $a_v \neq \emptyset \neq a_n$ **then**
6:         SEND-MESSAGE(AlreadyEngaged, $v > n, t$) to a_n and a_v
7:         $a_v \leftarrow \emptyset$
8:         Clear all of the maps in memory and tell a_j to cancel its search, forwarding the
  message to all agents in the current engaged block.
9:         **return**
10:     **else**
11:         $a_v \leftarrow a_n$ /* make a_i engaged for a_n */
12:     **end if**
13:     **if** $a_j = a_n$ **then** /* a_j is the variable requesting to be added */
14:         $m(a_n) \leftarrow 1$, $r(a_n) \leftarrow \emptyset$, and $s(a_n) \leftarrow$TRUE
15:     **else if** $a_j \in \pi^{-1}(a_i)$ **then** /* a_j is one of our children */
16:         $m(a_n) \leftarrow m(a_n) + 1$
17:         $r(a_n) \leftarrow r(a_n) \cup \{a_j\}$
18:     **end if**
19:     **if** $m(a_n) = q$ **then** /* a_i satisfies (6.5) for $N(a_n)$ */
20:         **if** $|r(a_n)| = 1$ **then** /* a_i is not the deepest */
21:             $q' \leftarrow q$
22:             **if** $s(a_n) \mapsto$TRUE **then** /* a_i was originally sent a message from a_n (meaning
  $a_i \in N(a_n)$) */
23:                 $q' \leftarrow q' - 1$
24:             **end if**
25:             $a_k \leftarrow$ the single variable in $r(a_n)$
26:             SEND-MESSAGE(AddMe, a_n, q', t) to a_k
27:             $a_v \leftarrow \emptyset$
28:             remove $m(a_n)$, $r(a_n)$, and $s(a_n)$ from memory
29:         **else** /* a_i is the deepest vertex satisfying (6.5) */
30:             **if** $r(a_n) \neq \emptyset$ **then** /* we have at least one descendant that is in $N(a_n)$ (Lemma 13)
  */
31:                 $\pi(a_n) \leftarrow \pi(a_i)$
32:                 $\pi(a_i) \leftarrow a_n$
33:                 SEND-MESSAGE(Added, $\pi(a_n), a_i$) to a_n
34:             **else** /* Lemma 14 */
35:                 $\pi(a_n) \leftarrow a_i$
36:                 SEND-MESSAGE(Added, $a_i, \emptyset$) to a_n
37:             **end if**
38:             $a_v \leftarrow \emptyset$
39:             Tell all of the agents in $r(a_n)$ that the search is over and clear all of the maps in
  memory.
40:         **end if**
41:     **else if** our vertex a_i is not the root of the pseudotree **then**
42:         SEND-MESSAGE(AddMe, a_n, q, t) to $\pi(a_i)$ /* Forward the message to our parent */
43:     **else** /* This *may* occur if there are two or more variables in $N(a_n)$ from disjoint
  hierarchies */
44:         SEND-MESSAGE(Root, a_i, t) to a_n
45:     **end if**
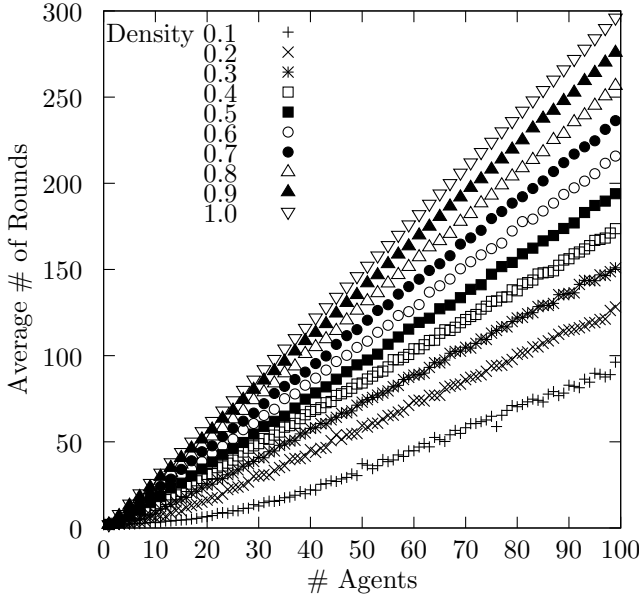46: **end procedure**

Figure 6.4: Average number of rounds for Mobed to reach quiescence for a single agent addition.

terms of existing hierarchy topology, T will be a chain and the worst case number of rounds for a single insertion is $O(|N(a_i)||A_T|)$. Construction of a hierarchy from scratch for a worst-case fully-connected interaction graph therefore requires

$$O\left(\sum_{i=1}^{|A|} i \cdot |A|\right) \quad = \quad O\left(\frac{|A|(|A|+1)^2}{2} - \frac{|A|(|A|+1)}{2}\right)$$

$$= \quad O\left(|A|^3\right)$$

rounds. The best-case runtime, however, is $O(|N(a_i)|)$, which in the real world will often be quite small. Therefore, the runtime in terms of number of rounds will always be polynomial and—if $N(a_i)$ is bounded—individual additions will run in amortized linear time. This bound is supported by the empirical results (Figure 6.4).

Edit Distance

The *edit distance* between two hierarchies, $T_1 = \langle A_1, \pi_1 \rangle$ and $T_2 = \langle A_2, \pi_2 \rangle$, is defined as the minimum number of child-parent relationships that must be
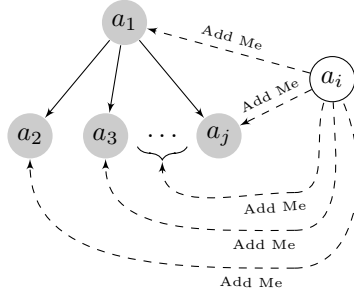
Figure 6.5: Worst case configuration for DFS edit distance: The existing hierarchy $T = \langle\{a_1, a_2, \ldots, a_j\}, \pi\rangle$ consists of a root, a_1, with $j-1$ children, and the agent being added, a_i, is connected to all $a_1, a_2, \ldots, a_j$ in T.

reassigned, added, or deleted in order for the two trees to become isomorphic:

$$\text{Edit-Distance}(A_1, A_2) =$$
$$\left| \left\{ \langle a_i, \pi_1(a_i)\rangle : a_i \in A_1 \right\} \ominus \left\{ \langle a_j, \pi_2(a_j)\rangle : a_j \in A_2 \right\} \right|.$$

Ideally, after a single addition of an agent the edit distance between the original hierarchy and the resulting hierarchy will be minimized. The worst case edit distance for DFS will occur whenever the existing hierarchy $T = \langle A_T, \pi\rangle$ consists of a root with $|A_T| - 1$ children and the agent being added, $a_i \notin A_T$, has the property $N(a_i) = A_T$ (*cf.* Figure 6.5). In this case $|A|$ edits may occur. In contrast, Mobed bounds the number of edits for each agent addition at two.

During our empirical analysis, edit distance according to the metric formalized above was noted for both DFS and Mobed between the initial and post-vertex-addition hierarchies. Figure 6.6 gives the percentage difference between the edit distance of DFS and Mobed; positive percentages mean that DFS had a worse edit distance. DFS performed worse for sparse graphs (density ≤ 0.5). Although DFS performed better on dense graphs, it was only ever one edit better. We believe that Mobed performs better on sparse graphs because in such instances Mobed is more likely to be able to add the new vertex as a leaf in the tree, which is the best case for both parallelism and edit distance.

Our definition of edit distance is quite favorable to DFS; in some domains a better metric may be the number of ancestor-descendant relationships that are modified as a result of each change to the interaction graph. Such perturbations in the context of DCR might cause large portions of the previously explored search space to be expanded again. With this stricter metric, Mobed still has a bounded edit distance of two, while DFS may perform much worse.
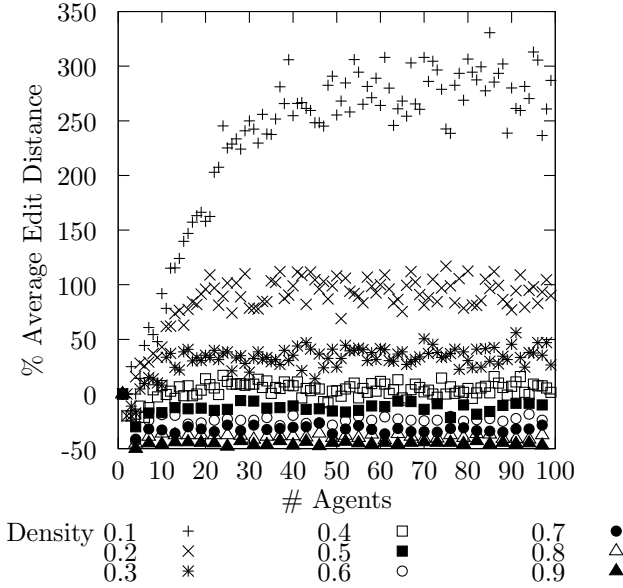
Figure 6.6: Comparison of the edit distance of DFS to Mobed. The y-axis is the percentage difference between DFS and Mobed; positive values imply DFS performed worse.

6.3 Conclusions

In this chapter we have tried to show how dynamic changes to the problem can be handled. First, we showed how multidirectional graph search can be extended to be a superstabilizing algorithm capable of handling dynamic changes consistent with proper functions. We were able to extend the algorithm such that it can handle dynamic modification with little or no additional overhead. This was accomplished by identifying a set of circumstances under which the algorithm execution could be continued after the addition or removal of a vertex. If such conditions are not met, a new backtracking mechanism is added that allows for the algorithm to revert to the most recent round in which the conditions are met. In doing so, the multidirectional graph search algorithm becomes superstabilizing; the algorithm is guaranteed to converge back to a valid state after any dynamic perturbation of the problem.

In the second half of the chapter, we investigated topologies which do not obey proper functions. We devised a new algorithm, Mobed, for solving a subset of such problems that take the form of the dynamic pseudotree construction problem. Mobed constructs and maintains multiagent hierarchies with bounded edit distance between hierarchy perturbations. Mobed was compared to the only other viable solution to the DynDisMHG problem: distributed DFS. It was shown that Mobed will always reach quiescence in a linear number of rounds for each agent addition with respect to the total number of agents, but under certain circumstances it can theoretically run in constant time. Re-running DFS after such a perturbation would also require a linear number of rounds, but may have arbitrarily bad edit distance. The edit distance of Mobed is always bounded at two edits, which is very low. For sparse graphs (density less than 0.5) Mobed has at least as good an edit distance as DFS, and exhibited as much as a 300% benefit. For those instances when Mobed exhibited a higher edit distance than DFS (*i.e.*, when the graph is dense) its edit distance was no more than one edit worse. Privacy is also maintained in Mobed insofar as agents only ever have knowledge of their interaction graph neighbors, hierarchy parents, and hierarchy children. Therefore Mobed is a viable replacement for DFS as a solution to the DynDisMHG problem, especially for sparse interaction graphs.

There are still some cases which have not been investigated in this chapter. For example, backtracking in multidirectional graph search may not always be necessary if additional memory is available to store more of the problem state. In the future we will also empirically analyze the average case efficiency of this approach. Furthermore, we will empirically analyze the use of Mobed in distributed problem solving algorithms. There is also much work to be done in studying constrained forest generation techniques that better balance the tradeoff between computational efficiency/messaging, edit distance, and

privacy, and also methods to maintain other invariants on the constrained forest's topology.

Chapter 7

Conclusions

This dissertation has shown that a large family of multiagent organization problems—collectively called *constrained forest problems*—can be solved efficiently in a distributed manner. Many of these problems are **NP**-Hard and are therefore intractable in centralized, sequential computation. We show that, in allowing distribution and exploiting locality in the primal-dual schema, speedups are achievable. In fact, we have shown that distributed algorithms can approximate a solution in linear time and, under certain well defined conditions, can even quiesce in polylogarithmic or even logarithmic time. This is a bit surprising because many of the constrained forest problems soluble to our approach are known to be **P**-Complete.

The primary novel contribution of this dissertation is a generalized distributed constrained multidirectional search algorithm based on the primal-dual schema that can solve constrained forest problems with a constant optimization bound in no worse than linear communication rounds. We have shown that the algorithm is correct and complete (*i.e.*, it is guaranteed to find a feasible solution if one exists). We have also provided a series of examples of how to instantiate this framework for specific problems, including Steiner network problems, art gallery/dominating set problems, and location design & vehicle routing problems.

Strong bounds on the convergence of the algorithm alone is not sufficient, however. We have therefore proven that if the edge weights of the input graph are mapped to a metric space that conforms to a specific set of constraints, then the solutions produced by our algorithm are guaranteed to be 2-optimal. It has been shown that this requirement is necessary to achieve the speedup from concurrency. If the input graph is not weighted in a sufficient metric space for the theoretical guarantees to hold, it was shown that there exists an ε such that the solution our algorithm discovers is with high probability ε-optimal. We have also motivated the fact that—even if the conditions of

the theoretical guarantees of 2-optimality (*qq.v.* Propositions 3 and 4) are not met—the solutions produced by the distributed multiagent graph search algorithm are with high probability 2-optimal.

The distributed approximation algorithm for multidirectional graph search is capable of finding a tree in the search space that connects all of a set of terminals by simultaneously performing a search emanating from each terminal. Each agent only requires local knowledge (*i.e.*, within the connected component of the forest), and there is no requirement of shared memory. That this type of search is efficiently distributable is an important result. We have shown that the algorithm and protocol will run in $O(n)$ communication rounds, however, empirical evidence suggests that the average case runtime is much lower. Furthermore, it was shown that the algorithm will produce a solution whose cost is within a factor of two of optimal. Once again, empirical evidence suggest that the average approximation bound is much closer to optimal (at about 1.3).

The constrained multidirectional search algorithm was also adapted for the location design and routing problem, retaining the same runtime bounds. Again, provided that the weight of the heaviest edge added in a round is no more than 150% of the lightest edge, the algorithm is guaranteed to produce a solution whose cost is no worse than two times optimal. This invariant can be maintained in general by embedding the true edge weights into $[1, \frac{3}{2}] \in \mathbb{Q}$. Empirical results suggest that the algorithm will in fact produce 2-optimal solutions for arbitrary edge weight distributions.

As another example of the framework's instantiation, the algorithm was extended for application to the art gallery and dominating set problems. It was shown that it is guaranteed to run in a number communication rounds on the order of the diameter of the visibility graph. The algorithm produces a solution whose cost is no worse than a constant factor of optimal with high probability. For art gallery variants in which the distances between guards and treasures/vertices must also be minimized, the algorithm is proven to be a 2-optimal approximation, provided that the edge weights are embedded in the proper metric space. It is known that the decision version of this and many other art gallery variants are **APX**-HARD [12], however, to the best of our knowledge the question of whether art gallery problems are in **APX** is an open problem. By identifying a class of art gallery problems that are amenable to 2-approximation using our algorithm, we have discovered that this class of problems is also in **APX** (and is thereby **APX**-COMPLETE).

Ultimately, we have furthered the results of Panconesi [67] by showing that the primal-dual optimization scheme proposed by Goemans and Williamson [73] can be successfully distributed into a multiagent algorithm with not only bounds on approximation, but also on runtime. This suggests that other multiagent coordination problems might yield to the same approach.

The ultimate problem we are working toward solving is that of *dynamic* multiagent organization. Ideally, the algorithm should not have to completely re-optimize subsequent to every perturbation of the problem. Therefore, we identified a family of events from which our algorithm can recover *faster* than having to re-optimize from scratch. There are also instances when the topological constraints on the desired forest are too expressive to be captured by our current primal-dual model. For such cases, we introduce an algorithmic extension called Multiagent Organization with Bounded Edit Distance (Mobed). Mobed was compared to the only other viable solution to the DynDisMHG problem: distributed DFS. It was shown that Mobed will always reach quiescence in a linear number of rounds for each agent addition with respect to the total number of agents, but under certain circumstances it can theoretically run in constant time. Re-running DFS after such a perturbation would also require a linear number of rounds, but may have arbitrarily bad edit distance. The edit distance of Mobed is always bounded at two edits, which is very low. For sparse graphs (density less than 0.5) Mobed has at least as good an edit distance as DFS, and exhibited as much as a 300% benefit. For those instances when Mobed exhibited a higher edit distance than DFS (*i.e.*, when the graph is dense) its edit distance was no more than one edit worse. Privacy is also maintained in Mobed insofar as agents only ever have knowledge of their interaction graph neighbors, hierarchy parents, and hierarchy children. Therefore Mobed is a viable replacement for DFS as a solution to the DynDisMHG problem, especially for sparse interaction graphs.

In the future we will empirically analyze the use of the generalized multidirectional graph search algorithm and Mobed in distributed systems and problem solving algorithms. There is also much work to be done in studying hierarchy generation techniques that better balance the tradeoff between computational efficiency/messaging, edit distance, and privacy, and also methods to maintain other invariants on the hierarchy's topology.

Despite the fact that proper functions capture a large family of constrained forest problems that are amenable to our approach, there is also work to be done in further generalizing the algorithms such that more expressive well behaved functions can be used (*e.g.*, well spaced, supermodular, submodular, and integer valued functions). A generalization of the quality of the solution for primal-dual methods as applied to constrained forest problems is also still an open question, as is the related problem of identifying necessary and sufficient conditions for the distribution of edge weights that will allow for a constant bound of approximation. Finally, we believe that algorithms of this ilk are a fruitful method for achieving speedups in constrained search and many other areas of multiagent systems through parallelism.

Many of the techniques espoused in this dissertation have potential application to the domain of social networking. Human systems and societies

have often beckoned researchers with questions like, "Does unchecked, nat-ural human competition lead to social inequity?" Such problems have been studied by sociologists [95], biologists [96, 97], and even physicists [98, 99]. Recent interest in social, "small-world" networks has also involved the com-puter science community [100], however, the majority of the existing work is focused on studying these systems with the intention of application to *human* societies, and therefore implicitly assume an adversarial environment. The scientific and engineering communities are mostly interested with data min-ing problems—*e.g.*, classifying cliques in social networks, or targeting ripe demographics for directed advertising. It is not difficult, however, to envi-sion certain agent-based societies in which, although each agent has its own goal, the agents are completely non-adversarial. Furthermore, existing stud-ies seem to focus on *modeling* and *simulating* social hierarchies; given their assumption that the mechanics by which societies are formed are immutable aspects of human nature, little or no emphasis is placed on societal *construc-tion* and *control*. These mechanics *are* mutable in computerized systems. Understanding of—and algorithms on—these types of social systems would appear to have wide application in many areas, especially once the dynamics of computerized agents and communications constraints are considered. If one were able to model such social systems as networks of interaction con-straints, the techniques explored in this dissertation might be extended to optimize such systems subject to their agents' requirements.

The following is a list of interesting and important open questions that arise from the results of this dissertation:

- Given the organizational structure employed by a set of agents, what is the underlying set of constraints that governs their interaction? (In a sense, this is the inverse of the problem explored in this dissertation.)

- To what extent can Elkin's time-approximation tradeoff bounds (*q.v.* §3.2.3) be unconditionally achieved for constrained forests in general?

- Given a *predictive* model of the communication constraints between agents, to what extent is it possible to construct an agent organization or social hierarchy that better enables coordination?

- To what extent do the existing techniques for social modeling apply in cooperative, non-adversarial systems?

- If we allow for adversarial agents, how can these techniques be extended to accommodate varying levels of trust?

- In some organizational topologies, certain agents will have more "power" or "social influence" than others. Can one discuss (or quantize) the so-cial influence between agents?

- Given a communications protocol, is it possible to place an *a priori* upper bound on the amount of social influence a single individual may have?

- To what extent can the maximum social influence of a single individual be minimized by solely modifying the communications network and/or manually defining the social hierarchy?

In the coming years, mobile computing technologies will undoubtedly continue to infiltrate every aspect of our daily lives, bringing us ever closer to this dissertation's vision of self-organizing agent organizations. The most significant impact will be gained through seamless, coalition-driven collaboration to aid humans in their daily functions. Successful collaboration will be achieved through the development of mechanisms for efficient construction and maintenance of self-organizing networks.

Bibliography

[1] Edsger W. Dijkstra. Self-stabilizing systems in spite of distributed control. *Communications of the ACM*, 17(11):643–644, 1974. ISSN 0001-0782. doi: http://doi.acm.org/10.1145/361179.361202. Cited on page 2.

[2] G. Xylomenos, G.C. Polyzos, P. Mahonen, and M. Saaranen. TCP performance issues over wireless links. *IEEE Communications Magazine*, 39(4):52–58, 2001. Cited on page 2.

[3] Gilbert Laporte. Location-routing problems. In Bruce L. Golden and Arjang A. Assad, editors, *Vehicle Routing: Methods and Studies*, pages 163–197. Elsevier, 1988. Cited on pages 3, 63, and 64.

[4] Maxim Peysakhov, Robert N. Lass, William C. Regli, and Moshe Kam. An ecological approach to agent population management. In *Proceedings of the Twentieth National Conference on Artificial Intelligence*, pages 146–151, 2005. Cited on page 4.

[5] Virginia Lo, Dayi Zhou, Yuhong Liu, Chris Gauthier-Dickey, and Jun Li. Scalable supernode selection in peer-to-peer overlay networks. In *Proceedings of the Second International Workshop on Hot Topics in Peer-to-Peer Systems*, pages 18–27, Washington, DC, USA, 2005. IEEE Computer Society. ISBN 0-7695-2417-6. doi: http://dx.doi.org/10.1109/HOT-P2P.2005.17. Cited on pages 4 and 64.

[6] Antony I. T. Rowstron and Peter Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *Middleware: Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms*, pages 329–350, London, UK, 2001. Springer-Verlag. ISBN 3-540-42800-3. Cited on page 4.

[7] Jan Melechovský, Christian Prins, and Roberto Wolfler Calvo. A meta-heuristic to solve a location-routing problem with non-linear costs. *Journal of Heuristics*, 11(5–6):375–391, 2005. ISSN 1381-1231. doi: http://dx.doi.org/10.1007/s10732-005-3601-1. Cited on pages 4 and 64.

[8] Joseph O'Rourke. *Art Gallery Theorems and Algorithms*. Oxford University Press, 1987. ISBN 0-19-503965-3. Cited on pages 5 and 79.

[9] Héctor González-Banos and Jean-Claude Latombe. A randomized art-gallery algorithm for sensor placement. In *Proceedings of the Seventeenth Annual Symposium on Computational Geometry*, pages 232–240, New York, NY, USA, 2001. ACM. ISBN 1-58113-357-X. doi: http://doi.acm.org/10.1145/378583.378674. Cited on page 5.

[10] Alok Aggarwal. *The Art Gallery Theorem: its Variations, Applications and Algorithmic Aspects*. PhD thesis, Johns Hopkins University, 1984. Cited on page 5.

[11] Der-Tsai Lee and Abel W. Lin. Computational complexity of art gallery problems. *IEEE Transactions on Information Theory*, 32(2): 276–282, 1986. ISSN 0018-9448. doi: http://dx.doi.org/10.1109/TIT. 1986.1057165. Cited on page 5.

[12] Stephan Eidenbenz, Peter Widmayer, and Christoph Stamm. Inapproximability results for guarding polygons and terrains. *Algorithmica*, 31 (1):79–113, 2001. Cited on pages 5 and 122.

[13] Richard M. Karp. Reducibility among combinatorial problems. In R. E. Miller and J. W. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103, New York, 1972. Plenum. Cited on page 5.

[14] David P. Williamson, Michel X. Goemans, Milena Mihail, and Vijay V. Vazirani. A primal-dual approximation algorithm for generalized steiner network problems. *Combinatorica*, 15(3):435–454, September 1995. Cited on pages 5, 20, 22, 33, 75, and 93.

[15] Joseph Macker, Ian Downard, Justin Dean, and Brian Adamson. Evaluation of distributed cover set algorithms in mobile ad hoc network for simplified multicast forwarding. *SIGMOBILE Mobile Computing Communications Review*, 11(3):1–11, 2007. doi: 10.1145/1317425.1317426. URL http://portal.acm.org/citation.cfm?id=1317425.1317426& coll=Portal&dl=GUIDE&CFID=64765812&CFTOKEN=41652687. Cited on pages 5, 11, and 80.

[16] Onn Shehory and Sarit Kraus. Methods for task allocation via agent coalition formation. *Artificial Intelligence*, 101(1-2):165–200, 1998. ISSN 0004-3702. doi: http://dx.doi.org/10.1016/S0004-3702(98) 00045-9. Cited on pages 7 and 10.

[17] Katia Sycara, Keith Decker, Anandeep Pannu, Mike Williamson, and Dajun Zeng. Distributed intelligent agents. *IEEE Expert: Intelligent*

Systems and Their Applications, 11(6):36–46, 1996. Cited on pages 7, 10, 15, 29, and 76.

[18] Marius C. Silaghi and Makoto Yokoo. Distributed constraint reasoning. In Juan Ramón Rabuñal Dopico, Julián Dorado de la Calle, and Alejandro Pazos Sierra, editors, *Encyclopedia of Artificial Intelligence*, pages 507–513. Information Science Reference, 2008. Cited on pages 7, 9, and 12.

[19] Pragnesh Jay Modi, Wei-Min Shen, Milind Tambe, and Makoto Yokoo. ADOPT: Asynchronous distributed constraint optimization with quality guarantees. *Artificial Intelligence Journal*, 161(1–2):149–180, 2005. Cited on pages 7, 11, 13, 14, and 26.

[20] Adrian Petcu and Boi V. Faltings. A scalable method for multiagent constraint optimization. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 266–271, 2005. Cited on pages 7, 13, and 14.

[21] Anton Chechetka and Katia Sycara. No-commitment branch and bound search for distributed constraint optimization. In *Proceedings of the fifth international joint conference on autonomous agents and multiagent systems*, pages 1427–1429, Hakodate, Japan, 2006. ACM Press. ISBN 1-59593-303-4. doi: http://doi.acm.org/10.1145/1160633.1160900. Cited on page 7.

[22] Lap Kong Law, Srikanth V. Krishnamurthy, and Michalis Faloutsos. Understanding and exploiting the trade-offs between broadcasting and multicasting in mobile ad hoc networks. *IEEE Transactions on Mobile Computing*, 6(3):264–279, 2007. Cited on page 7.

[23] Adrian Petcu and Boi Faltings. S-DPOP: Superstabilizing, fault-containing multiagent combinatorial optimization. In *Proceedings of the Twentieth National Conference on Artificial Intelligence*, pages 449–454, July 2005. Cited on page 8.

[24] Adrian Petcu and Boi Faltings. R-DPOP: Optimal solution stability in continuous-time optimization. In *Proceedings of the International Conference on Intelligent Agent Technology*, November 2007. Cited on page 8.

[25] Robert N. Lass, Evan A. Sultanik, and William C. Regli. Dynamic distributed constraint reasoning. In *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence*, pages 1886–1887, 2008. Cited on page 8.

[26] Klaus Fischer, Michael Schillo, and Jörg Siekmann. Holonic and multi-agent systems for manufacturing. In *Holonic Multiagent Systems: A Foundation for the Organisation of Multiagent Systems*, volume 2744 of *Lecture Notes in Computer Science*, pages 1083–1084. Springer, 2004. Cited on pages 8 and 11.

[27] Chris J. van Aart, Bob Wielinga, and Guus Schreiber. Organizational building blocks for design of distributed intelligent system. *International Journal of Human-Computer Studies*, 61(5):567–599, 2004. ISSN 1071-5819. doi: http://dx.doi.org/10.1016/j.ijhcs.2004.03.001. Cited on page 8.

[28] Makoto Yokoo. Asynchronous weak-commitment search for solving distributed constraint satisfaction problems. In *Proceedings of the First International Conference on Principles and Practice of Constraint Programming*, pages 407–422, 1995. Cited on pages 8 and 9.

[29] Anton Chechetka and Katia Sycara. A decentralized variable ordering method for distributed constraint optimization. In *Proceedings of the Fourth International Joint Conference on Autonomous Agents and Multiagent Systems*, July 2005. Cited on pages 8, 9, and 104.

[30] Evan A. Sultanik, Robert N. Lass, and William C. Regli. Dynamic configuration of agent organizations. In *Proceedings of the International Joint Conference on Artificial Intelligence*, July 2009. Cited on pages 8, 10, and 104.

[31] Stefan Arnborg. Efficient algorithms for combinatorial problems on graphs with bounded decomposability—a survey. *BIT Numerical Mathematics*, 25(1):1–23, March 1985. Cited on page 8.

[32] Youssef Hamadi, Christian Bessière, and Joël Quinqueton. Backtracking in distributed constraint networks. In *Proceedings of the European Conference on Artificial Intelligence*, pages 219–223, 1998. Cited on page 9.

[33] Zeev Collin and Shlomi Dolev. Self-stabilizing depth-first search. *Information Processing Letters*, 49(6):297–301, 1994. Cited on page 9.

[34] John Davin and Pragnesh Jay Modi. Hierarchical variable ordering for distributed constraint optimization. In *Proceedings of the Fifth International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 1433–1435, New York, NY, USA, 2006. ACM. Cited on page 9.

[35] Marius Silaghi and Makoto Yokoo. Dynamic DFS tree in ADOPT-ing. In *Proceedings of the Twenty-Second AAAI Conference on Artificial Intelligence*, Vancouver, BC, Canada, September 2007. Cited on page 9.

[36] James Atlas and Keith Decker. A complete distributed constraint optimization method for non-traditional pseudotree arrangements. In *Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems*, pages 1–8, New York, NY, USA, 2007. ACM. ISBN 978-81-904262-7-5. doi: http://doi.acm.org/10.1145/1329125. 1329262. Cited on page 9.

[37] Roie Zivan and Amnon Meisels. Dynamic ordering for asynchronous backtracking on DisCSPs. *Constraints*, 11:179–197, 2006. Cited on page 9.

[38] Thomas R. Ioerger and Linli He. Modeling command and control in multi-agent systems. In *Proceedings of the 8th International Command and Control Research and Technology Symposium*, June 2003. Cited on page 11.

[39] Dimitrios J. Vergados, Nikolaos A. Pantazis, and Dimitrios D. Vergados. Energy-efficient route selection strategies for wireless sensor networks. *Mobile Networks and Applications*, 13(3–4):285–296, 2008. URL http://portal.acm.org/citation.cfm?id=1414641.1414646& coll=Portal&dl=ACM&CFID=66742779&CFTOKEN=96468443. Cited on page 11.

[40] Qunfeng Dong. Maximizing system lifetime in wireless sensor networks. In *Proceedings of the 4th International Symposium on Information Processing in Sensor Networks*, Los Angeles, California, 2005. IEEE Press. ISBN 0-7803-9202-7. URL http://portal.acm.org/citation.cfm?id=1147685.1147690& coll=Portal&dl=ACM&CFID=66742779&CFTOKEN=96468443. Cited on page 11.

[41] William Yeoh, Ariel Felner, and Sven Koenig. BnB-ADOPT: an asynchronous branch-and-bound DCOP algorithm. In *Proceedings of the 7th International Joint Conference on Autonomous Agents and Multiagent Systems*, volume 2, pages 591–598, Estoril, Portugal, 2008. International Foundation for Autonomous Agents and Multiagent Systems. ISBN 978-0-9817381-1-6. URL http://portal.acm.org/citation.cfm?id=1402298.1402307& coll=Portal&dl=GUIDE&CFID=63340268&CFTOKEN=98798546. Cited on pages 11 and 14.

[42] Jonathan P. Pearce and Milind Tambe. Quality guarantees on k-optimal solutions for distributed constraint optimization problems. In *Proceedings of the 20th International Joint Conference on Artifical Intelligence*,

pages 1446–1451, Hyderabad, India, 2007. Morgan Kaufmann Publishers Inc. URL `http://portal.acm.org/citation.cfm?id=1625509`. Cited on page 11.

[43] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, January 1979. Cited on pages 11, 12, and 80.

[44] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. McGraw-Hill, second edition, 2001. Cited on pages 12 and 141.

[45] David Musser. Introspective sorting and selection algorithms. *Software: Practice and Experience*, 27(8):983–993, 1997. Cited on page 12.

[46] Nancy A. Lynch. *Distributed Algorithms*. Morgan Kaufmann, 1997. ISBN 1558603484. Cited on pages 12, 29, and 76.

[47] Ying Zhang and Alan K. Mackworth. Parallel and distributed algorithms for finite constraint satisfaction problems. In *Proceedings of the Third IEEE Symposium on Parallel and Distributed Processing*, pages 394–397, 1991. Cited on page 13.

[48] Harsh Bhatia, Rathinasamy Lenin, Aarti Munjal, Srini Ramaswamy, and Sanjay Srivastava. A queuing-theoretic framework for modeling and analysis of mobility in WSNs. In *Proceedings of the Eighth Performance Metrics for Intelligent Systems Workshop*. The National Institute of Standards and Technology, September 2008. Cited on page 13.

[49] Weixiong Zhang, Zhao Xing, Guandong Wang, and Lars Wittenburg. An analysis and application of distributed constraint satisfaction and optimization algorithms in sensor networks. In *Proceedings of the Second International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 185–192, New York, NY, USA, 2003. ACM. ISBN 1-58113-683-8. doi: http://doi.acm.org/10.1145/860575.860605. Cited on page 14.

[50] Adrian Petcu and Boi Faltings. MB-DPOP: a new memory-bounded algorithm for distributed optimization. In *Proceedings of the 20th International Joint Conference on Artifical Intelligence*, pages 1452–1457, Hyderabad, India, 2007. Morgan Kaufmann Publishers Inc. URL `http://portal.acm.org/citation.cfm?id=1625510`. Cited on page 14.

[51] Amnon Meisels, Eliezer Kaplansky, Igor Razgon, and Roie Zivan. Comparing performance of distributed constraints processing algorithms. In *Proceedings of the Third International Workshop on Distributed Constraint Reasoning*, Bologna, Italy, July 2002. URL `citeseer.ist.psu.edu/meisels02comparing.html`. Cited on page 13.

[52] John Davin and Pragnesh Jay Modi. Impact of problem centralization in distributed constraint optimization algorithms. In *Proceedings of the Fourth International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 1057–1063, Utrecht, The Netherlands, 2005. ACM Press. ISBN 1-59593-093-0. doi: http://doi.acm.org/10.1145/1082473.1082633. Cited on page 13.

[53] Marius Silaghi, Robert N. Lass, Evan A. Sultanik, William C. Regli, Toshihiro Matsui, and Makoto Yokoo. Constant cost of the computation-unit in efficiency graphs for dcops. In *Proceedings of the International Conference on Intelligent Agent Technology*, December 2008. Cited on page 13.

[54] Leslie Lamport. Time, clocks and the ordering of events in a distributed system[1]. *Communications of the ACM*, 21(7):558–565, July 1978. Cited on page 13.

[55] Yaacov Fernandess, Antonio Fernández, and Maxime Monod. A generic theoretical framework for modeling gossip-based algorithms. *SIGOPS Operating Systems Review*, 41(5):19–27, 2007. doi: 10.1145/1317379.1317384. URL `http://portal.acm.org/citation.cfm?id=1317379.1317384&coll=GUIDE&dl=ACM&CFID=64961512&CFTOKEN=44495261`. Cited on page 15.

[56] Alexandros G. Dimakis, Anand D. Sarwate, and Martin J. Wainwright. Geographic gossip: efficient aggregation for sensor networks. In *Proceedings of the 5th international conference on Information processing in sensor networks*, pages 69–76, Nashville, Tennessee, USA, 2006. ACM. ISBN 1-59593-334-4. doi: 10.1145/1127777.1127791. URL `http://portal.acm.org/citation.cfm?id=1127777.1127791&coll=GUIDE&dl=ACM&CFID=64961512&CFTOKEN=44495261`. Cited on page 15.

[57] Stephen Boyd, Arpita Ghosh, Balaji Prabhakar, and Devavrat Shah. Randomized gossip algorithms. *IEEE Transactions on Information Theory*, 52(6):2508–2530, 2006. URL `http://portal.acm.org/citation.cfm?id=1148663.1148679&coll=GUIDE&dl=ACM&CFID=64961512&CFTOKEN=44495261`. Special issue of IEEE Transactions on Information Theory and IEEE/ACM Transactions on Networking. Cited on page 15.

---

[1]This is one of the most cited computer science papers of all time [101] and contains perhaps the most amusing footnote of any academic paper ever[2] (which was the inspiration for the footnotes on page 13 of this dissertation).

[2]. . . even more amusing than this footnote.

[58] Mauro Sozio. *Efficient Distributed Algorithms via the Primal-Dual Schema*. PhD thesis, "La Sapienza" University, Rome, September 2006. Cited on pages 15 and 20.

[59] Amir Sadeh. Distributed primal-dual approximation algorithms for network design problems. Master's thesis, The Open University of Israel, December 2008. Cited on pages 15 and 20.

[60] Manica Aggarwal and Naveen Garg. A Scaling Technique for Better Network Design. In *Proceedings of the Fifth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 233–240, Philadelphia, PA, USA, 1994. Society for Industrial and Applied Mathematics. ISBN 0-89871-329-3. URL `http://portal.acm.org/citation.cfm?id=314464.314498&coll=Portal&dl=ACM&CFID=31452541&CFTOKEN=77022537#`. Cited on pages 17, 22, 25, 28, and 61.

[61] Vijay V. Vazirani. Primal-Dual schema based approximation algorithms (Abstract). In *Computing and Combinatorics*, pages 650–652, 1995. URL `http://citeseer.ist.psu.edu/vazirani95primaldual.html`. Cited on pages 18, 22, and 28.

[62] Vašek Chvátal. *Linear Programming*. W. H. Freeman, New York, 1983. Cited on page 19.

[63] Carlo Lombardi. Primal-dual algorithms. Lecture Notes, June 2008. Cited on page 19.

[64] Harold W. Kuhn. The hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2:83–97, 1955. Cited on page 19.

[65] George Bernard Dantzig, Lester Randolph Ford, Jr., and Delbert Ray Fulkerson. A primal-dual algorithm for linear programs. In Harold W. Kuhn and Albert W. Tucker, editors, *Linear Inequalities and Related Systems*, pages 171–181, Princeton, NJ, 1956. Princeton University Press. Cited on page 19.

[66] Reuven Bar-Yehuda and Shimon Even. A linear-time approximation algorithm for the weighted vertex cover problem. *Journal of Algorithms*, 2:198–203, 1981. Cited on page 19.

[67] Alessandro Panconesi. *Fast Distributed Algorithms Via Primal-Dual (Extended Abstract)*, volume 4474, pages 1–6. Springer, Heidelberg, 2007. Cited on pages 20, 95, and 122.

[68] Marcelo Santos, Lúcia M. A. Drummond, and Eduardo Uchoa. A distributed primal-dual heuristic for steiner problems in networks. In *Experimental Algorithms*, volume 4525, pages 175–188. Springer, 2007. Cited on pages 20, 77, and 94.

[69] Ricardo C. Corrêa, Fernando C. Gomes, Carlos A. S. Oliveira, and Panos M. Pardalos. A parallel implementation of an asynchronous team to the point-to-point connection problem. *Parallel Computing*, 29(4): 447–466, May 2002. Cited on page 20.

[70] Alessandro Panconesi and Mauro Sozio. Fast distributed scheduling via primal-dual. In *Proceedings of the twentieth annual symposium on Parallelism in algorithms and architectures*, pages 229–235, Munich, Germany, 2008. ACM. ISBN 978-1-59593-973-9. Cited on page 20.

[71] Samir Khuller, Uzi Vishkin, and Neal Young. A primal-dual parallel approximation technique applied to weighted set and vertex cover. *Journal of Algorithms*, 17(2):280–289, October 1994. Cited on page 20.

[72] F. Grandoni, J. Könemann, Alessandro Panconesi, and Mauro Sozio. Primal-dual based distributed algorithms for vertex cover with semi-hard capacities. In *Proceedings of the twenty-fourth annual ACM symposium on Principles of distributed computing*, pages 118–125, Las Vegas, NV, USA, 2005. ISBN 1-59593-994-2. Cited on page 20.

[73] Michel X. Goemans and David P. Williamson. A general approximation technique for constrained forest problems. *SIAM Journal on Computing*, 24:296–317, 1995. Cited on pages 20, 21, 61, 64, 68, 85, 93, 95, and 122.

[74] Kamal Jain. A factor 2 approximation algorithm for the generalized steiner network problem. In *Proceedings of the 39th Annual Symposium on Foundations of Computer Science*, page 448. IEEE Computer Society, 1998. ISBN 0-8186-9172-7. URL `http://portal.acm.org/citation.cfm?id=796444`. Cited on page 22.

[75] Michael X. Goemans, Andrew V. Goldberg, Serge Plotkin, David B. Shmoys, Éva Tardos, and David P. Williamson. Improved approximation algorithms for network design problems. In *Proceedings of the 5th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 223–232, January 1994. Cited on page 22.

[76] J. E. Dennis, Jr. and Virginia Torczon. Direct search methods on parallel machines. *SIAM Journal on Optimization*, 1(4):448–474, November 1991. Cited on page 25.

[77] David Ilcinkas, Nicolas Nisse, and David Soguet. The cost of monotonicity in distributed graph searching. *Distributed Computing*, 22(2): 117–127, September 2009. Cited on page 25.

[78] David Šišlák, Pavel Jisl, Přemysl Volf, Michal Pěchouček, David Nicholson, David Woodhouse, and Niranjan Suri. Integration of probability

collectives for collision avoidance in agentfly. In *Proceedings of 8th International Conference on Autonomous Agents and Multiagent Systems*, pages 69–76, May 2009. Cited on page 25.

[79] Richard E. Korf. Linear-time disk-based implicit graph search. *Journal of the ACM*, 55(6):1–40, 2008. ISSN 0004-5411. doi: http://doi.acm.org/10.1145/1455248.1455250. Cited on page 26.

[80] Michael Elkin. Unconditional lower bounds on the time-approximation tradeoffs for the distributed minimum spanning tree problem. In *Proceedings of the Thirty-Sixth Annual ACM Symposium on Theory of Computing*, pages 331–340, New York, NY, USA, 2004. ACM. ISBN 1-58113-852-0. doi: http://doi.acm.org/10.1145/1007352.1007407. Cited on pages 41, 42, and 43.

[81] Sándor Csörgő and Gordon Simons. Precision calculation of distributions for trimmed sums. *The Annals of Applied Probability*, 5(3): 854–873, 1995. Cited on page 48.

[82] H. N. Nagaraja. Order statistics from independent exponential random variables and the sum of the top order statistics. In N. Balakrishnan, Enrique Castillo, and José María Sarabia, editors, *Advances in Distribution Theory, Order Statistics, and Inference,* Part III, Statistics for Industry and Technology, pages 173–185. Birkhäuser, Boston, 2006. Cited on page 52.

[83] Shay Halperin and Uri Zwick. Optimal randomized EREW PRAM algorithms for finding spanning forests and for other basic graph connectivity problems. In *Proceedings of the Seventh Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 438–447, Philadelphia, PA, USA, 1996. Society for Industrial and Applied Mathematics. ISBN 0-89871-366-8. Cited on page 64.

[84] Richard Cole, Philip N. Klein, and Robert E. Tarjan. Finding minimum spanning forests in logarithmic time and linear work using random sampling. In *Proceedings of the Eighth Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 243–250, New York, NY, USA, 1996. ACM. ISBN 0-89791-809-6. doi: http://doi.acm.org/10.1145/237502.237563. Cited on page 64.

[85] Leslie Lamport, Robert Shostak, and Marshall Pease. The byzantine generals problem. *ACM Transactions on Programming Languages and Systems*, 4(3):382–401, 1982. ISSN 0164-0925. doi: http://doi.acm.org/10.1145/357172.357176. Cited on page 76.

[86] Robert N. Lass, Michael J. Grauer, Evan A. Sultanik, and William C. Regli. A decentralized approach to the art gallery problem. In *Proceedings of the 17th Fall Conference on Computational Geometry*, November 2007. Cited on page 80.

[87] Amnon Meisels. *Distributed Search by Constrained Agents: Algorithms, Performance, Communication.* Springer-Verlag, London, 2008. ISBN 1848000391, 9781848000391. Cited on page 80.

[88] Anurag Ganguli, Jorge Cortés, and Francesco Bullo. Visibility-based multi-agent deployment in orthogonal environments. In *Proceedings of the American Control Conference*, pages 3426–3431, New York, July 2007. Cited on page 80.

[89] Jie Wu and Hailan Li. On calculating connected dominating set for efficient routing in ad hoc wireless networks. In *Proceedings of the 3rd International Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications*, pages 7–14, New York, NY, USA, 1999. ACM. ISBN 1-58113-174-7. doi: http://doi.acm.org/10.1145/313239.313261. Cited on page 80.

[90] Lu Ruan, Hongwei Du, Xiaohua Jia, Weili Wu, Yingshu Li, and Ker-I Ko. A greedy approximation for minimum connected dominating sets. *Theoretical Computer Science*, 329(1–3):325–330, 2004. ISSN 0304-3975. doi: http://dx.doi.org/10.1016/j.tcs.2004.08.013. Cited on page 80.

[91] Chuanhe Huang, Chuan Qin, and Yi Xian. A distributed algorithm for computing connected dominating set in ad hoc networks. *International Journal of Wireless and Mobile Computing*, 1(2):148–155, 2006. ISSN 1741-1084. doi: http://dx.doi.org/10.1504/IJWMC.2006.012474. Cited on page 81.

[92] Linda L. Deneen and Shashikant Joshi. Treasures in an art gallery. In *Proceedings of the 4th Canadian Conference on Computer Geometry*, pages 17–22, 1992. Cited on page 92.

[93] Svante Carlsson and Håkan Jonsson. Guarding a treasury. In *Proceedings of the 5th Canadian Conference on Computer Geometry*, pages 85–90, 1993. Cited on page 93.

[94] Leslie Lamport. Paxos made simple. *ACM SIGACT News (Distributed Computing Column)*, 32(4):51–58, December 2001. Cited on page 102.

[95] James Samuel Coleman. *Foundations of Social Theory.* Belknap, New York, 1994. Cited on page 124.

[96] Lee Alan Dugatkin. Winner and loser effects and the structure of dominance hierarchies. *Behavioral Ecology*, 8:583–587, 1997. Cited on page 124.

[97] Charlotte K. Hemelrijk. Towards the integration of social dominance and spatial structure. *Animal Behavior*, 59:1035–1048, 2000. Cited on page 124.

[98] Eric Bonabeau, Guy Theraulaz, and Jean-Louis Deneubourg. Phase diagram of a model of self-organizing hierarchies. *Physics A*, 217:373–392, 1995. Cited on page 124.

[99] Lazaros K. Gallos. Self-organizing social hierarchies on scale-free networks. *International Journal of Modern Physics C*, 16(8):1329–1336, 2005. Cited on page 124.

[100] Michael Kirley. Dominance hierarchies and social diversity in multi-agent systems. In *Proceedings of the 8th annual conference on Genetic and Evolutionary Computation*, pages 159–166, New York, NY, USA, 2006. ACM. ISBN 1-59593-186-4. doi: http://doi.acm.org/10.1145/1143997.1144026. Cited on page 124.

[101] CiteSeer. Most cited articles in computer science, September 2006. `http://citeseer.ist.psu.edu/articles.html`. Cited on page 133.

[102] Leslie Lamport. *LATEX: A Document Preparation System*. Addison-Wesley, Menlo Park, 1986. Cited on page 151.

[103] Donald E. Knuth. *The TEXbook*. Addison-Wesley, Menlo Park, 1984. Cited on page 151.

[104] Donald E. Knuth. *Computer Modern Typefaces*, volume E of *Computers and Typesetting*. Addison-Wesley, Reading, MA, USA, 1986. ISBN 0-201-13446-2. Cited on page 151.

Appendix A

Notation, Nomenclature, and Glossary

Notation

$\equiv$	The definition of an identity.
R^+	The transitive closure of a binary relation R.
$X \times Y$	If X and Y are sets, the Cartesian product (or direct product) of two sets: $\{(x,y)|x \in X \wedge y \in Y\}$, otherwise multiplication.
$\prod$	The Cartesian product over the elements of a set. $\prod_{i \in \{1,2,3,4\}} X_i \equiv X_1 \times X_2 \times X_3 \times X_4$.
2^X	The power set of X.
$|X|$	Cardinality, when X is a set; the absolute value of X otherwise.
$\langle a,b,c \rangle$	A tuple containing elements a, b, and c.
$\setminus$	The relative complement set operator: $A \setminus B \equiv \{x \in A : x \notin B\}$.
$\ominus$	The symmetric difference set operator: $A \ominus B \equiv (A \cup B) - (A \cap B)$.
X^T	The transpose of a matrix X.
$\text{erf}(x)$	The error function: $\text{erf}(x) = \dfrac{2}{\sqrt{\pi}} \displaystyle\int_0^x e^{-t^2} dt$.
$\max_{x \in X} f(x)$	The maximum value of $f(x)$ over all elements in the set X.

$f : A \rightarrow B$	A function, f, mapping the elements of set A to the elements of the set B.
$f(a) \mapsto b$	The statement that function f maps $a \in A$ to $b \in B$.
$f^{-1}(b)$	The inverse of a function; given a function $f : A \rightarrow B$, f^{-1} maps B to a subset of A such that $f^{-1}(b) \mapsto \{a \in A : f(a) \mapsto b\}$.
$\implies$	The material conditional (*i.e.*, implies operator); $p \implies q \equiv \neg p \lor q$.
$u \rightsquigarrow v$	The path between vertices u and v in a graph.
$\iff$	If and only if: $p \iff q \equiv (p \implies q) \land (q \implies p)$.
$\stackrel{d}{=}$	The operands exhibit the same distribution.
$O(g(x))$	Given two functions $f(x)$ and $g(x)$ defined on some subset of $\mathbb{R}$, we say $f(x)$ is $O(g(x))$ as $x \rightarrow \infty$ if and only if $\exists x_0 \exists y > 0$ such that $|f(x)| \leq y|g(x)|$ for $x > x_0$.
$L_1 \leq_p L_2$	The language L_1 is *polynomial-time reducible* to a language L_2; there exists a polynomial-time function $f : \{0,1\}^* \rightarrow \{0,1\}^*$ such that for all $x \in \{0,1\}^*$, $x \in L_1 \iff f(x) \in L_2$.

Nomenclature

$\emptyset$	The empty set.
$\mathbb{B}$	A boolean domain: $\{\text{TRUE}, \text{FALSE}\}$.
δ	A function $\delta : 2^V \rightarrow 2^{V \times V}$ such that given a set of vertices of a graph $S \subseteq V$, $\delta(S) \mapsto C$ implies that each edge in C contains exactly one endpoint in S.
N	Unless otherwise stated, $N : V \rightarrow 2^V$ is a function mapping vertices of a graph to their neighbors. $N(v) \mapsto \{x, y\}$ means that vertices x and y are directly connected to v. Note that N is an "open" neighborhood function, meaning that v itself is never returned in the set $N(v)$.
$\mathbb{N}_0$	The natural numbers, including zero (*i.e.*, non-negative integers).
NP	The class of languages (*i.e.*, problems) can be verified by a polynomial-time algorithm. A language L belongs to **NP** if and only if there exist a two-input polynomial-time algorithm A and a constant c such that $L = \{x \in \{0,1\}^* :$ there exists a certificate y with $|y| = O\left(|x|^c\right)$ such that $A(x, y) = 1\}$.

NP-HARD	Non-deterministic Polynomial-time Hard. A class of languages (*i.e.*, problems) that contains all languages L such that for all $L' \in$ **NP**, $L' \leq_p L$.
$\mathbb{Q}$	The rational numbers.
$\mathbb{R}$	The real numbers.
§	A section of this document, *e.g.*, "§2.1" refers to the first section of the second chapter.
"we"/"us"/"our"	*Pluralis modestiæ.* The author has attempted to forgo first person singular pronouns in an effort to engage the reader, remain somewhat modest, and tempt assumptions of schizophrenia.
$\mathbb{Z}$	The integers.

The definitions for $O(g(x))$, $L_1 \leq_p L_2$, **NP**, and **NP**-HARD were adapted from [44].

Glossary of Abbreviations, Acronyms, and Initialisms

a.k.a.	also known as	
CDF	Cumulative Distribution Function	
cf.	*confer*	consult
CREW	Concurrent Read Exclusive Write	
DisCOP	Distributed Constraint Optimization Problem (sometimes also abbreviated "DCOP")	
DPOP	The Distributed Pseudotree Optimization Procedure	
DSA	The Distributed Stochastic Algorithm	
DynDisMHG	Dynamic Distributed Multiagent Hierarchy Generation	
&	*et*	and
& [pl.] al.	*et [pluribus] alii/aliæ/alia*	and [many] other men/women/things
&c.	*et cetera*	and the rest
e.g.	*exempli gratia*	for example
i.e.	*id est*	that is [to say]
IP	Integer Program	
LAN	Local Area Network	
LP	Linear Program	
MANET	Mobile *Ad Hoc* Network	
MAS	Multiagent System	
MST	Minimum Spanning Tree	

Mobed	Multiagent Organization with Bounded Edit Distance
ε-OPT	An algorithm is ε-Optimal (ε-OPT) if its solutions are guaranteed to be no worse than ε times the cost of the optimal solution.
PRAM	Parallel Random Access Machine
PDF	Probability Density Function
q.v.	*quod vide* for which, see _____ (herein)
qq.v.	*quæ vide* for which (plural), see _____ (herein)
Ti*k*Z	Ti*k*Z ist fein Zeichenprogramm Ti*k*Z is *not* a drawing program
vice versa	Latin for "the other way around".
vi&.	*videlicet/videre licet* that is to say (precisely)

Index

Important references are given in italics.

“Any inaccuracies in this index may be explained
by the fact that it has been sorted with the help
of a computer. ”

—Donald Knuth
Volume 3 of The Art of Computer Programming

Vitæ Curriculum Brevis

Evan A. Sultanik concurrently earned the degrees of Bachelor of Science in Mathematics from the Drexel University College of Arts and Sciences and both Bachelor & Master of Science in Computer Science from the Drexel University College of Engineering in 2006, having graduated with honors distinction from Pennoni Honors College. His Masters thesis was on *Enabling Multi-Agent Coordination in Stochastic Peer-to-Peer Environments* and was co-advised by Drs. William C. Regli, Pragnesh Jay Modi, and Moshe Kam. During that time Evan occupied a joint position in the Drexel University Geometric and Intelligent Computing Laboratory (GICL), the Applied Communications and Information Networking (ACIN) Institute, and was a *de jure* member of the Data Fusion Laboratory (DFL). Upon deciding to continue his study toward a Ph.D., Evan took on Dr. Ali Shokoufandeh as a co-advisor to Dr. Regli. In a previous life during the "dot-com bubble", Evan worked as a software engineer at the document management company Feith Systems.

Evan is a member of the Association for the Advancement of Artificial Intelligence (AAAI), the Association for Computing Machinery (ACM), the Institute for Electrical and Electronics Engineers (IEEE), the IEEE Communications Society, the American Association for the Advancement of Science (AAAS), the National Eagle Scout Association (NESA), and the eGullet Society for Culinary Arts and Letters.

Aside from when he is required to write in a biographical format, Evan does not often refer to himself in the third grammatical person.

Colophon

The majority of the text of this dissertation was devised at the Applied Communications and Information Networking (ACIN) Institute in Camden, New Jersey. The text was written using Esterbrook fountain pens[1] that were also devised in Camden, just a couple blocks away from ACIN. The text was typeset using the LaTeX document markup language [102] for the TeX document preparation system [103]. The bibliography was automatically generated using BibTeX. Typing and editing were executed using a combination of Emacs[2] and Vim[3] on computers running GNU/Linux. All figures were produced in TikZ[4] and all graphs were rendered using Gnuplot[6]. The æsthetics of this document are attributable to these excellent tools.

All of the typefaces used in this dissertation are from the Computer Modern family[7], created by Donald Knuth in METAFONT [104].

This dissertation contains approximately fifty-one thousand fifty-two words and two hundred eighty-nine thousand five hundred fifty characters, including those of this sentence.

For additional credits see page vi.

For legal notices see the copyright page.

---

[1] Nibs: 9550 (firm extra-fine; bookkeeping), 2284 (broad; signature stub), and 9128 (flexible extra-fine; Pitman shorthand).

[2] http://www.gnu.org/software/emacs/

[3] http://www.vim.org/

[4] TikZ is a recursive acronym for "TikZ ist *kein* Zeichenprogramm"[5]. http://sourceforge.net/projects/pgf/

[5] ...which is German for "TikZ is *not* a drawing program".

[6] http://www.gnuplot.info/

[7] Except for the use of Yannis Haralambous' 𝔉𝔯𝔞𝔠𝔱𝔲𝔯 font on page 142 and Knuth's METAFONT font above. (Both fonts are also used in this footnote.)