This thesis presents techniques that enable multi-agent coordination in peer-to-peer environments such as mobile ad-hoc networks (MANETs). Although MANETs are increasingly used to support mobile, networked computing in many civil and military applications, very few methods exist for distributed multi-agent coordination in such a paradigm. The coordination problem can be decomposed into two sub-problems: the problem of developing a local belief of the global state, thereby allowing for formation of local preferences, and the problem of using the local preferences to distributedly find local policies that satisfy a global optimum. We address the former problem in the domain of service discovery on peer-to-peer networks, and the latter in the domain of distributed task scheduling. Due to the hardware requirements and unpredictability of wireless networks, however, a simulated environment in which to compare agent-based algorithms and perform controlled experiments is first required. To address this need I present the Macro Agent Transport Event-based Simulator (MATES), a novel application-layer simulator created to investigate the behavior of distributed agent-based systems running atop dynamic peer-to-peer networks and MANETs. MATES was implemented *not* to simulate a specific agent framework, but to provide a generic, easily extendible environment for mobile agent-based system testing. MATES is used for empirical validation of the algorithms presented in this thesis. Next, I propose a technique for mobile service discovery on MANETs using mobile agents, along with a model of the system that provides a performance profile for time-critical reasoning on the discovery process. I show that the model of the system is highly correlated to the ground truth. The service discovery procedure can be used to either automate or augment the process of modeling the world in a language such as C_TÆMS. A mapping of a subset of C_TÆMS to an equivalent distributed constraint optimization problem (DCOP) is presented. We provide a proof that our subset of C_TÆMS has not been trivialized (*i.e.*, finding the optimal schedule is still $\mathcal{NP}$-HARD), validate the approach using MATES and the DCOP algorithm Adopt, and propose domain bounding techniques for improving the efficiency of this process by up to 60%.

Committee:
William C. Regli
Pragnesh Jay Modi
Moshe Kam

Evan A. Sultanik

Master's Thesis

Master's Thesis

Enabling Multi-Agent Coordination in Stochastic Peer-to-Peer Environments

Evan A. Sultanik

A Thesis
Submitted to the Faculty of Drexel University
in partial fulfillment of the requirements for the
degree of
Master of Science in Computer Science

April, 2006

Enabling Multi-Agent Coordination in Stochastic Peer-to-Peer Environments

Evan A. Sultanik

A Thesis
Submitted to the Faculty of Drexel University
in partial fulfillment of the requirements for the degree of
Master of Science in Computer Science

Dedications

This thesis is dedicated to my grandparents Ann & Sol Sultanik and Vivian & Byron Clyman for their sacrifices and support, without which this thesis would not exist.

Acknowledgements

First and foremost, I must thank William C. Regli for having the patience and foresight in guiding me into the world of academic research five years ago. In many respects Moshe Kam and Pragnesh Jay Modi were also my advisors, for whom I must equally thank for donating their time and expertise.

Several people have critiqued portions and/or drafts of this thesis, including Bill Regli, P. Jay Modi, Nadya Belov, Joe Kopena, and Max Peysakhov, to whom I owe thanks for their time and input. I would also like to thank David Dorsey, Chris Dugan, Joe Kopena, Rob Lass, Andrew Mroczkowski, and Max Peysakhov for their use of and comments on my implementation of the Macro Agent Transport Event-based Simulator.

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

This text was typeset using LaTeX [31], which was built atop TeX [27]. The bibliography was automatically generated using BibTeX. Typing and editing were done using a combination of Emacs and Vim on Linux-based computers. All figures were produced in either Xfig or Dia, and all graphs were rendered using Gnuplot. This thesis would have been less polished and æsthetically pleasing had it not been for these tools.

Contents

Abstract **xi**

1 Introduction **1**
1.1 Motivation . . . 2
1.2 Approach . . . 3
1.2.1 Evaluating Multi-Agent Systems on Dynamic Peer-to-Peer Networks . . . 4
1.2.2 Developing a Belief of the Global State . . . 4
1.2.3 Converging Upon the Global Schedule . . . 4
1.3 Organization of the Thesis . . . 6

2 Background **7**
2.1 Agents . . . 7
2.2 Peer-to-Peer Networks . . . 7
2.3 Service-Based Computing . . . 8
2.4 Coordination . . . 9
2.5 Related Work . . . 10

3 Tools for Investigating Multi-Agent Coordination **13**
3.1 Motivation and Design Goals . . . 13
3.2 Architecture . . . 14
3.2.1 Hosts and Agents . . . 14
3.2.2 Simulated Time . . . 15
3.2.3 Host Mobility Model . . . 18
3.2.4 Link Connectivity Model . . . 18
3.2.5 Data Transport Model . . . 18
3.2.6 Routing . . . 18
3.3 Implementation . . . 18
3.4 Scalability . . . 20
3.5 Examples & Validation . . . 24
3.6 Limitations . . . 25

4 Coping with Local Knowledge **27**

4.1 Introduction 27
4.2 Technical Approach 30
4.2.1 Random Walking Mobile Agents 31
4.2.2 Predicting Agent Arrival at a Host 32
4.2.3 Mathematics of Random Walks 33
4.2.4 Approximating ν 34
4.2.5 Accounting for τ 34
4.2.6 Constructing $\psi(N, s, h)$ 35
4.2.7 Using $\psi(N, s, h)$ 36
4.3 Empirical Validation 37
4.3.1 Accuracy of PageRank. 37
4.3.2 Verification of Services' Distribution. 38
4.3.3 Accuracy of $\psi(N, s, h)$. 38
4.4 Discussion 38

5 Multi-Agent Coordination **43**
5.1 Formalization 44
5.1.1 Modeling Planning Problems 44
5.1.2 Distributed Constraint Optimization 45
5.1.3 Analysis 46
5.2 Technical Approach 47
5.2.1 Mapping C_TÆMS to a DCOP 47
5.2.2 Naïve Domain Bounding 49
5.2.3 Bound Propagation 50
5.2.4 Constraint Propagation 50
5.3 Results 52

6 Conclusions **59**
6.1 Evaluating Multi-Agent Systems on Dynamic Peer-to-Peer Networks 59
6.1.1 Estimating Global State 59
6.1.2 Multi-Agent Coordination 60
6.1.3 Coordinating Agents in Stochastic Peer-to-Peer Environments 61

A Notation and Nomenclature **63**

List of Tables

5.1 Efficiency of Algorithm 8 at reducing average domain size and state space size, in terms of solubility. 56
5.2 Solubility statistics for different complexities of C_TÆMS instances. All simulations were conducted with four agents. None of the problems bounded using the naïve method were soluble. * This is the default configuration for the C_TÆMS scenario generator. . 57

List of Figures

1.1 Given a model of the world represented in C_TÆMS, map this to a DCOP whose solution is guaranteed to lead to a feasible schedule (with bounds on optimality). 5

3.1 The simulator cycle. 15

3.2 Flow of control of the behavioral model in a simulation of two hosts and three agents. MATES first allocates execution time to h_1, which sub-allocates time to its agents a_1 and a_2. When each agent reaches a blocking function, control returns to the simulation kernel. When h_1 is finished, h_2 will be given execution time. Finally, h_2 allocates execution time to its agent a_3. 16

3.3 Screen shot of MATES' GUI, simulating a mobile ad hoc network of 25 hosts. 19

3.4 Computation time for simulating 500 quanta as a function of the number of agents and the number of hosts. 20

3.5 Memory required for simulating 500 quanta as a function of the number of agents and the number of hosts. 21

3.6 Topological statistics recorded over the experiments. When the hosts are bounded in a physical space, the average topology diameter remains constant while the average host degree increases linearly. 22

3.7 Computation time and memory usage as a function of average host degree. Note that both values are highly correlated to host degree. 23

4.1 A scenario showing web services on a MANET. Agents on Host A require services offered by each provider S_i. 28

4.2 An agent randomly walking a peer-to-peer network. 31

4.3 Profile of $\psi(N, s, h)$. 36

4.4 (a) depicts the error in the predicted agent frequency, ν, in static and dynamic networks over a 100 quanta (second) simulation using the PageRank algorithm. The error converges to 0 over time. (b) compares the estimator given in Equation (4.2) to the actual value over the course of a simulation. The means are highly correlated. 39
4.5 Prediction, $\psi(N, s, h)$, and actual agent visitation probabilities for a 300000 quanta (simulated seconds) experiment. 40

5.1 An example task hierarchy, (a), with associated representation in C_TÆMS, (b), along with our mapping to a DCOP, (c). 48
5.2 Feasible start times for methods M_1 and M_2 both before, (a), and after, (b), constraint propagation. If the C_TÆMS model declares M_1 as enabling M_2, (a), then constraint propagation will increase the lower bound on the start time of M_2 to the earliest start time of M_1 plus the expected duration of M_1, (b). 52

Abstract

Enabling Multi-Agent Coordination
in Stochastic Peer-to-Peer Environments

Evan A. Sultanik
Advisor: William C. Regli, Ph. D.

This thesis presents techniques that enable multi-agent coordination in peer-to-peer environments such as mobile ad-hoc networks (MANETs). Although MANETs are increasingly used to support mobile, networked computing in many civil and military applications, very few methods exist for distributed multi-agent coordination in such a paradigm. The coordination problem can be decomposed into two sub-problems: the problem of developing a local belief of the global state, thereby allowing for formation of local preferences, and the problem of using the local preferences to distributedly find local policies that satisfy a global optimum. We address the former problem in the domain of service discovery on peer-to-peer networks, and the latter in the domain of distributed task scheduling. Due to the hardware requirements and unpredictability of wireless networks, however, a simulated environment in which to compare agent-based algorithms and perform controlled experiments is first required. To address this need I present the Macro Agent Transport Event-based Simulator (MATES), a novel application-layer simulator created to investigate the behavior of distributed agent-based systems running atop dynamic peer-to-peer networks and MANETs. MATES was implemented *not* to simulate a specific agent framework, but to provide a generic, easily extendible environment for mobile agent-based system testing. MATES is used for empirical validation of the algorithms presented in this thesis. Next, I propose a technique for mobile service discovery on MANETs using mobile agents, along with a model of the system that provides a performance profile for time-critical reasoning on the discovery process. I show that the model of the system is highly correlated to the ground truth. The service discovery procedure can be used to either automate or augment the process of modeling the world in a language such as C_TÆMS. A mapping of a subset of C_TÆMS to an equivalent distributed constraint optimization problem (DCOP) is presented. We provide a proof that our subset of C_TÆMS has not been trivialized (*i.e.*, finding the optimal schedule is still $\mathcal{NP}$-HARD), validate the approach using MATES and the DCOP algorithm Adopt, and propose domain bounding techniques for improving the efficiency of this process by up to 60%.

Chapter 1

> “Scientific discovery and scientific knowledge have been achieved only by those who have gone in pursuit of it without any practical purpose whatsoever in view.”
> —Max Karl Ernst Ludwig Planck

Introduction

Consider, for a moment, the following thought experiment. You are a delegate on the floor of the United Nations on November 24th, 1945; there are over fifty delegates, each originating from a different social context, trying to ratify their charter. The majority of the delegates' native languages are not mutually intelligible, necessitating translation. Although no dedicated translators are available, the delegates can proceed by translating for each other. The Belgian representative can translate between French, German and Dutch, the Ukrainian between Polish, Ukrainian, and Russian, and so on. It is noon; the delegates wish to adjourn for a meal and therefore must agree on a time to reconvene. Each delegate has his own personal commitments and constraints, so a unanimous decision must be made. In order to voice your preferences, you want to know the current time. For example, you know that you have a meeting at 12:30 and therefore will not be able to reconvene until 13:00, at the earliest. The only language you speak is Spanish, you neither know whom else speaks Spanish nor do you know that the Dutch delegate is the only one with a watch. Therefore, you ask, «*¿Que hora es?*» This is understood by the multilingual Mexican representative and repeated as “What is the time?” The multilingual South African representative understands this and repeats it in Zulu and—the key—Netherlands: „*Hoe laat is het?*” The Dutch delegate understands the question and the process is reversed to relay the time.

The problem posed by this scenario is not one of natural language processing, nor is it one of translation; it is that the answer is outdated—and therefore invalid—by the time you receive it. Given your outdated information, how can you develop a belief of the world-state (*i.e.*, the time)? Furthermore, what happens if you do not receive an answer? The lack of a reply could have four implications:

1. You have not waited long enough for an answer;

2. No one else knows the answer;
3. Someone else knows the answer, but there does not exist a chain of delegates to relay the question to those that know the answer; or
4. There does not exist a chain of delegates to relay the answer back to the questioner.

The problems that arise in this scenario are not distinct to the dubious domain of distributedly-deciphered diplomacy; they have striking parallels in other disciplines, such as Computer Science. In the setting of a system of multiple, autonomous, intelligent agents, these are the questions for which my thesis provides answers:

- How can the agents discern which of the implications from a lack of a reply are true?
- To what extent is it possible to develop an *a priori* estimate of the expected time until a reply?
- More importantly, once everyone has enough information about the global state to develop a *local* preference, how can the agents coordinate to converge upon the *globally-optimal* decision (in which violations of the agents' local preferences is minimized)?
- How can multi-agent algorithms designed to answer the above problems be empirically validated and compared?

1.1 Motivation

The proliferation of mobile and handheld computing (*i.e.*, laptops, PDAs, and smart phones) has propelled distributed computing into mainstream society. Consider coordinating a night at the theater with one's friends, each using his or her PDA for collaboration. Like the delegates in the thought experiment, complete communication between peers is not ensured. For example, one's PDA might not be in radio range of—and will therefore be incommunicado with—another's. Just as the network of communication among the delegates was defined by language barriers, the peer-to-peer network of PDAs might be defined by radio connectivity.

As a more pertinent example, first responders (*e.g.*, triage, police, fire fighters) at the scene of a natural disaster cannot expect to have any sort of existing communications infrastructure; their devices must be able to create a mobile *ad hoc* network (MANET) [69]. In this domain, a police officer, while preventing looting, might encounter a building that is emitting smoke from its top floor. The officer neither knows if there are victims trapped inside, nor does he or she know the severity of the fire. The officer immediately alerts the fire

department and emergency medical units, using the MANET. Both the fire department and the medical units need to prioritize the situation (*i.e.*, develop a local preference) before any coordination can occur, especially if the demand for firefighters and medics is greater than the supply. Developing such a local preference might require assessing the current state of existing emergencies. For example, the availability of fire fighters might depend upon the location and degree of control of existing fires, and on a related note, the location and state of the fire fighting infrastructure (*e.g.*, trucks, oxygen tanks, firefighters, and water). The globally-optimal decision, however, cannot solely be based upon the local assessments made by each department. For example, even if both the fire department and triage units have enough resources to immediately address the situation, it might be more beneficial for the triage units to be temporarily diverted elsewhere since the building must be deemed safe by the firefighters before the medics can enter.

The coordinated management of inter-dependent plans or schedules belonging to different agents is an important, complex, real-world problem. Diverse application domains such as disaster rescue, small-team reconnaissance, security patrolling and others require human teams to schedule their tasks and activities in coordination with one another. This is currently difficult because it requires search over an overwhelming number of decisions and actions under stressful conditions. It is envisioned that automated planning and scheduling processes in the form of intelligent software agents operating on portable computing hardware can assist human teams in such domains. The problem is inherently a distributed one; an agent that administers the schedule of a human user must effectively manage the interdependencies between its schedule and the schedules of other users by exchanging information and coordinating with other agents. No single agent has a global view of the problem; instead, each must make local planning and scheduling decisions through collaboration with other agents to ensure a high quality global schedule.

1.2 Approach

Due to the hardware requirements and unpredictability of wireless networks, a simulated environment in which to compare agent-based algorithms and perform controlled experiments is required before any empirical investigation into multi-agent coordination can occur. Therefore, we present the Macro Agent Transport Event-based Simulator (MATES) [63] to address this need. The problem of multi-agent coordination itself can be decomposed into two distinct processes: developing a local belief of the global state and using one's local knowledge to converge on an optimal schedule. The following sections highlight our approach to both developing and validating solutions to these problems.

1.2.1 Evaluating Multi-Agent Systems on Dynamic Peer-to-Peer Networks

In developing multi-agent systems on peer-to-peer networks [62], I have experienced firsthand numerous problems that arise from working with a live testbed. Hardware failures and system misconfiguration can cost valuable research time. Although producing agent-based technologies for a functional wireless network has always been my goal, I have recognized the need for a simulated environment in which to compare agent-based algorithms and perform controlled experiments. To address this need I have developed the MATES: an application-layer simulator created to investigate the behavior of distributed agent-based systems running atop dynamic peer-to-peer networks and MANETs. Given such a goal, MATES was implemented *not* to simulate any specific agent framework, but to provide a generic, easily extendible environment for mobile agent-based system testing. MATES is used for empirical validation of the algorithms presented herein.

1.2.2 Developing a Belief of the Global State

In order to develop one's local preferences, one might need a belief of the global state of the world. Recall from the United Nations scenario that the delegates might need to know the time—an element of non-local state—in order to decide upon their *local* preference for a time to reconvene. In order to distribute state information throughout the network, I present a bottom-up method that exploits the emergent behavior of mobile service discovery agents that stochastically sample the hosts on a wireless network. These agents act like bees, working to "pollenate" the network with state information; using a model of behavior derived from the theory of random walks, the agents develop a belief about the state of network services and propagate these beliefs to the hosts as they travel the network. Furthermore, this model can be used as a performance profile for time-critical reasoning. A fixed-memory randomized method for approximating the location of a service in a stochastic environment with a probabilistic certainty in a fixed amount of time is proposed by augmenting the discovery agents' knowledge with domain and network semantics. Empirical analyses are provided to show the accuracy of the technique on simulated wireless networks.

1.2.3 Converging Upon the Global Schedule

Once one has enough global state information to develop a model of the world and thereby a local preference, there is still the problem of *coordinating* to converge upon the *global* optimum. First, we present a subset of the Coordinators Task Analysis Environmental Modeling and Simulation (C_TÆMS) modeling language [10] that is expressive enough to represent these types of coordination and distributed scheduling problems. C_TÆMS is a general language (based on

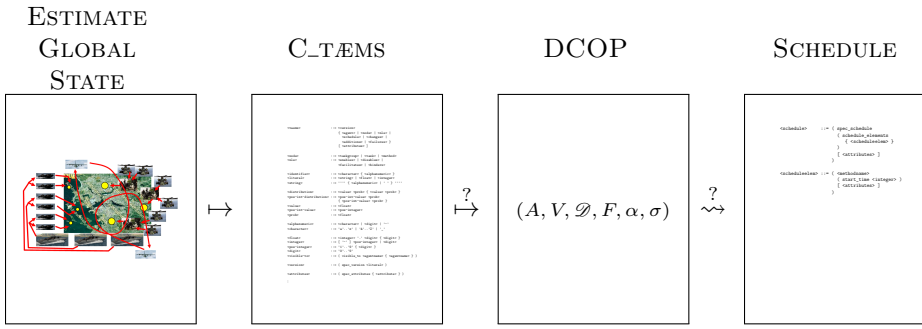


Figure 1.1: Given a model of the world represented in C_TÆMS, map this to a DCOP whose solution is guaranteed to lead to a feasible schedule (with bounds on optimality).

the original TÆMS language [20]) that was jointly designed by several multi-agent systems researchers explicitly for multiagent task scheduling problems. We prove that our subset of C_TÆMS has not been reduced to a point that it has become trivial; solving the scheduling problems represented by our subset of C_TÆMS is $\mathcal{NP}$-HARD. Next, we propose a *distributed constraint reasoning* approach for solving problems expressed in our subset of the C_TÆMS modeling language [10]. Our methodology is one of problem transformation in which we create a mapping that converts a C_TÆMS instance into an instance of the Distributed Constraint Optimization Problem (DCOP) [41], as depicted in Figure 1.1. DCOP was devised to model constraint reasoning problems where several agents must communicate to ensure that solutions are globally optimal.

We demonstrate three key advantages of our constraint based approach. First, we provide a mapping that automatically transforms a C_TÆMS problem into an *equivalent* DCOP. That is, we show that the optimal solution to a given C_TÆMS problem is also the optimal solution to the DCOP obtained by applying our problem transformation mapping, *i.e.*, our mapping preserves the problem exactly. Second, arguably one of the primary advantages of adopting a constraint based approach is the ability to apply existing constraint propagation techniques from the constraint programming literature. Thus, we leverage our problem transformation by applying constraint propagation pre-processing techniques to perform domain pruning to significantly reduce the search space. We demonstrate a 96% reduction in state space size for a set of benchmark problems. Finally, another advantage of our approach is that by mapping the C_TÆMS problem into a DCOP, we are able to immediately apply existing solution techniques for DCOP to the C_TÆMS problem domain. We report on experiments using the Adopt algorithm [41] which is one of the state-of-the-art DCOP algorithms that finds guaranteed optimal solutions through asyn-

chronous peer-to-peer message passing between agents with worst-case polynomial space requirements.

1.3 Organization of the Thesis

The remainder of this thesis is organized as follows:

§2 Provides background on the various concepts upon which this thesis is built and overviews related research;

§A Defines the notation and nomenclature used for formalization;

§3 Outlines the simulator, MATES, created for investigating multi-agent collaboration on MANETS and subsequently used for empirical validation of our algorithms;

§4 Develops our bottom-up approach for distributing state information throughout a dynamic peer-to-peer network;

§5 Formalizes and validates our top-down approach to mapping coordination problems to DCOPs; and

§6 Concludes on my research and presents areas for future investigation.

Chapter 2

> “Today's scientists have substituted mathematics for experiments, and they wander off through equation after equation, and eventually build a structure which has no relation to reality.”
>
> —Nikola Tesla
> “Radio Power Will Revolutionize the World.”
> Modern Mechanics and Inventions, July, 1934.

Background

2.1 Agents

According to the Agent Systems Reference Model [9], “An agent is a situated computational process with one or more of the following properties: autonomy, proactivity and interactivity.” In the context of this thesis, an agent may also be assumed to have the ability to *migrate*. Migration occurs when an agent halts its execution, saves its state, and is transferred to another *host* at which it resumes execution. A “host,” therefore, is any node on the network capable of receiving and executing agents.

2.2 Peer-to-Peer Networks

A network is considered “peer-to-peer” when its constituent nodes have no concept of a “server” or “client;” there exists a *peering protocol* that allows for communication between any pair of nodes. Additional constraints may be placed upon who can talk to whom, as in MANETs. A MANET's network topology is defined by the radio connectivity of the nodes, which is usually dictated by the hosts' spatial interrelationships. The structure of such networks inherently imposes the restriction of communication to one's “neighbors.” Communication over multiple hops in the network is difficult or impossible without facilities such as ad hoc routing [49, 23, 48, 24]. Recent research has invested the use of mobile agents for routing in ad hoc networks [38, 40, 57], establishing the

utility of further investigation in this area [39]. Most methods rely on using an agent's frequency of host visitation to predict the network topology. Significantly less work exists on discovering the current locations of *services* throughout MANETs—in other words, the "service topology."

MANETs are vastly different from enterprise networks. Nodes on a MANET communicate over wireless links, with each node acting as a router, forwarding messages. Network management tasks, such as keeping the route tables up to date, is a significantly harder problem on MANETs, necessitating ad-hoc routing protocols that add complexity and network overhead to the system. Bandwidth is typically severely limited. Throughput in IEEE 802.11b MANETs, the most common civil standard, is often less than 50% of the theoretical maximum of 11 Mbps [70]. In contrast, traditional switched wired networks can reach rates upward of 100 Mbps. Further, limited link capacities are accompanied by low connectivity between nodes. As network hosts move through their environment, the likelihood of a wireless signal being obscured by buildings, cars, and other obstacles increases. Network routes are constantly changing and end-to-end connectivity is rare [2]. Host devices are also diverse, ranging from powerful laptops to resource-limited handheld devices.

2.3 Service-Based Computing

As networks and software become increasingly heterogeneous, the problem of classifying the assets provided by and required for each entity is becoming nontrivial. The service-oriented computing paradigm addresses this issue by abstracting the assets in a network. This abstraction allows for matching between agents and services based on semantic value, as opposed to simple, name-based addressing.

A service-oriented computing framework is generally comprised of five basic elements relating to services:

1. **Service Description: Ontology.** The service description element is perhaps the most important, as it dictates the expressiveness of the entire framework. Known in the philosophical and artificial intelligence communities as *ontology*, the service description should be formal enough for automated reasoning to be carried out, while abstract enough to be human-readable. Theoretically, multiple agents with the same ontological description of a service should equally be able to inter-communicate about/with that service.

2. **Advertisement** Most service frameworks have a mechanism by which services can advertise themselves when available. Advertisement usually takes the form of a network broadcast/multicast; therefore this is not present in all frameworks, as bottlenecks such as bandwidth are prohibitive.

3. **Discovery** On small, flat, fully-connected networks, service discovery is a somewhat trivial problem. However, when networks get excessively large and complex, service discovery is an exceedingly difficult problem. There are generally three steps in the service discovery process:

   (a) An agent formulates a description of the desired service;

   (b) Either the agent or the framework creates a matching function for the desired description; and

   (c) Once a match is found, the framework provides a mechanism through which the agent can talk to the service provider.

4. **Invocation** Although not always implemented in existing frameworks, the invocation element provides abstracted, transparent communication between the agent and service provider.

5. **Composition** An automated mechanism able to combine multiple services into one.

2.4 Coordination

DCOPs [41] are a subclass of distributed decision processes in which a set of agents are responsible for assigning the values of their respective variables subject to a set of constraints. The objective of a DCOP is for the agents to assign values to their variables such that the cost incurred by the constraints is either minimized or maximized. Several distributed algorithms for DCOP currently exist in the literature including distributed dynamic programming (DPOP [50]), cooperative mediation (OptAPO [37]) and asynchronous backtracking (Adopt [41, 1, 14, 66, 36, 17]).

Constraint propagation is a general family of techniques that exploits a constraint-based representation for consistency checking to reduce problem search space, often in a pre-processing phase. Bartak provides a survey of the general topic in [5]. Arguably one of the primary advantages of adopting a constraint based approach is the ability to apply constraint propagation techniques such as node, arc or path consistency. As such, constraint propagation is well-studied with several algorithms available that vary in complexity and the amount of pre-processing computation and memory required. The first arc consistency algorithms were formalized by Mackworth [35] including the widely used AC-3 algorithm. Bessiere, *et al.*, improve on average-case efficiency with the AC-6 and AC-7 algorithms [7, 8]. More recently, researchers have introduced distributed methods for arc consistency processing including the DisAC-9 algorithm [22] and the DMAC protocol [59].

2.5 Related Work

Issues concerning pervasive computing environments appear close the problems of distributing state information and service discovery in dynamic environments. New service discovery architectures and protocols are being developed to accommodate the limitations of pervasive computing, such as [13] which has opted for intelligent forwarding of service requests over the typical advertisement registry.

Due to the proliferation of mobile applications and peer-to-peer file sharing technology, significant research has been made in the area of dynamic networks. Service discovery architectures exist for such networks [32, 25], some specifically designed for MANETs [30], however, these often require services to register themselves when available and are therefore not suitable for distributing global state information. Search techniques have also been applied to the problem of localization, such as Content-Addressable Networks (CANs) [56, 61], sometimes using agent-based techniques [18, 43]. However, search-based approaches often assume that agents can arbitrarily migrate between hosts in the network [4]. Furthermore, CANs and Distributed Hash Tables assume that the index and data elements are fixed. None of these approaches directly address the problem of locating *mobile* services.

Research in service matching and discovery includes a popular matching algorithm that defines a match in varying degrees of success while also using a Semantic Web language to extend the capabilities of a commonly used web service repository, UDDI[1] [45]. The number of inputs and outputs shared between the service request and the service advertisement creates a convenient preference hierarchy for service matches. [71] adds flexibility to the service matching process by allowing the description of a service's behavior, rather than just inputs and outputs. Additionally, it utilizes a logic language to exploit the inheritance relationships within an ontology to increase the possibility of a useful match. Sometimes, to achieve a desired result, many services and service processes may be involved. In this situation, [68] describes an HTN planning process to perform composite process models and [44] uses petri nets to analyze specific maintenance and safety conditions before automatically composing a service.

Other work incorporates both service discovery and process execution to foster the complete automation of tasks. The work in [47] suggests a service architecture composed of a DAML-S/UDDI Matchmaker, with the planning and service execution performed by an autonomous agent. The agent uses the Matchmaker to find desired services, checks provided schedules for date and time conflicts, and then uses the DAML-S virtual machine to help interpret information retrieved during Web service invocation. This system represents a typical service oriented architecture: a central registry for maintaining service advertisements, paired with appropriate parsing and inferring mechanisms, as

[1] http://www.uddi.org/

well as a querying agent acting on behalf of a user or company. Similar full Semantic Web systems have been implemented in various ways, including using a broker agent architecture as in [46].

Numerous techniques have been employed to address the problem of multi-agent coordination and scheduling (and various subproblems thereof). Musliner, *et al.*, map C_TÆMS to a Markov decision process which is then used to generate a policy dictating when and how agents execute methods [42]. Musliner, *et al.*, also propose distributed constraint satisfaction as a method for negotiating a feasible schedule by maximizing previously-calculated local expected qualities [42]. Smith, *et al.*, address the problem of dynamic schedule revision (due to changes in the model) using Simple Temporal Networks [60]. Phelps and Rye address this same problem through a domain-independent implementation of Generalized Partial Global Planning, in which *proxy methods* allow for distributed approximation of the performance characteristics of potentially complex tasks [55].

Chapter 3

> “In agent-based simulation, formulation does not require quantification.”
>
> —Scott Moss
> At the Sixth International Workshop on Multi-Agent-Based Simulation

Tools for Investigating Multi-Agent Coordination

This chapter is organized as follows: we present our motivation for developing MATES in §3.1, followed by a description of MATES' architecture in §3.2. §3.3 presents implementation details for MATES. Empirical validation of the scalability of MATES is presented in §3.4. Examples of how MATES has been used and validation of its models are presented in §3.5. We outline limitations of the current implementation, as well as mention possible improvements, in §3.6.

3.1 Motivation and Design Goals

The domain of mobile agency on ad hoc networks is unique, in that it falls between the fields of artificial intelligence and networking. There already exist many simulators from each field, such as MASON [34], for agent simulation, and the almost ubiquitous NS-2 [67] and OPNET[1] for network simulation.

The authors considered extending an existing simulator for use with dynamic peer-to-peer networks, however none allowed for their specific design goals. The first design goal was that the simulator should only model the network at a macroscopic level, allowing for computation time to be diverted to the agent model. Secondly, developing agents for the simulator should be analogous to development for a real-world agent architecture, such as EMAA [33] or COUGAAR [6].

The first design goal eliminated virtually all network simulators, as their purpose is to model low level interaction (even down to the physical layer). While there exist extensions to NS-2 for allowing simulation of mobile agents,

[1] `http://www.opnet.com/`

such as AgentJ[2] and a module by Shah [58], efficiency of the simulation itself is not a specific design goal. For example, Shah's extension to NS-2 was designed with the intent of investigating the efficacy of mobility as a design paradigm, not as a general-purpose mobile agent simulation environment. In general, network simulators are primarily used to compare low-layer processes, such as routing algorithms. As such, most network simulators do not provide sufficient hooks for implementing application-layer protocols. Modeling an agent system at such a resolution has proven to create a fair amount of overhead. Therefore, MATES was created to approximate the low-layer processes to divert more resources to the agent model. This concession of accuracy mitigates the problem of efficiency; further analysis is provided in §3.5.

A huge disparity between agent simulators and actual multi-agent systems is their scheduling model: the means by which agents are allotted processing time. Real-world multi-agent systems (especially mobile agent systems) almost always provide each agent with its own thread of execution, allowing for agents to run concurrently. In these systems, agents need not compartmentalize their execution into blocks over which the system will iterate. Likewise, agents need not schedule future execution times with the agent architecture. Instead, since each agent is encapsulated by a thread, the operating system or virtual machine can allot processing time. Most all agent simulators take an opposite approach. Simulators tend to provide each agent with a hook that gets called once every simulator iteration. This makes modeling processing constraints unintuitive. For example, it is not immediately clear how one would model a network of heterogeneous hosts, each having drastically different computational power. An agent currently hosted on a PDA might only have a tenth the computational capability of an agent hosted on a laptop. MATES' solution to this problem is presented in §3.2.2.

3.2 Architecture

MATES is based upon four core models used for simulation: *Host Mobility*, *Link Connectivity*, *Transport*, and *Agent Behavior*. Each model can be thought of as a conglomeration or approximation of an associated "block" of the OSI model [72]. During every cycle of the simulator, each of the models is successively applied to the domain, as in Figure 3.1.

3.2.1 Hosts and Agents

MATES models two primary entities: *hosts* and *agents*. Unlike some agent architectures, such as EMAA, MATES does not provide a thread of execution for servers. Instead, static agents (*i.e.*, agents that never migrate) can be used.

Each agent can query its current host for the following percepts:

[2] `http://pf.itd.nrl.navy.mil/srss/`

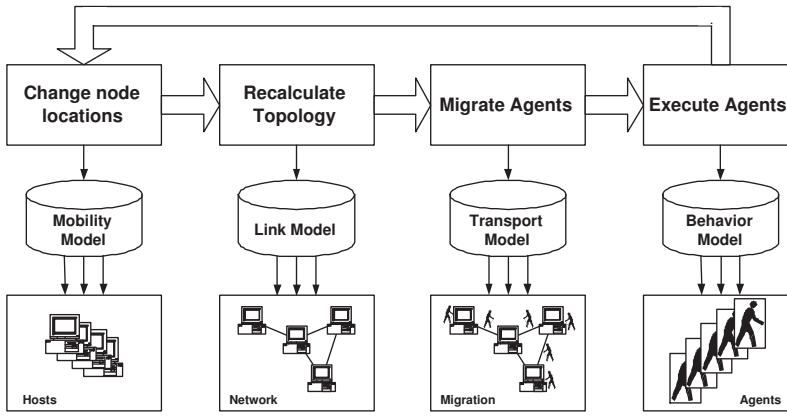


Figure 3.1: The simulator cycle.

- handles to any *other* agents at the current host;
- geographic location of the current host; and
- network addresses of the neighboring hosts.

It is also important to note that agents do not exist on any host while in transit between hosts. Furthermore, agent migration can fail, for example, if the source and destination hosts move out of range mid-transmit, or if a timeout occurs. In the event of a migration failure, simulation parameters can dictate whether an agent returns to the host from which it was sent or dies. The latter case can be useful for modeling network packets as agents [65].

3.2.2 Simulated Time

Time in MATES is modeled using *iterations* that represent simulated time quanta. The simulator iterates over the quanta, allotting processing time to each of the hosts. Hosts, in turn, may divide their allotted time to any agents they are hosting. Similar approaches are taken by most other agent simulators.

MATES differentiates its approach by additionally providing each agent with a separate thread of execution. This does not mean that agents execute concurrently; the threads are used to allow for agents to *suspend* their execution while waiting on blocking operations, such as sleeping and migrating between hosts. When the blocking function is complete, the agent's thread may then be started again, continuing execution at the same point at which it suspended. The flow of control while simulating the behavioral model is pictured in Figure 3.2. The approach of preserving agent execution state throughout an entire simulation is similar to that of Java in Simulated Time [3].

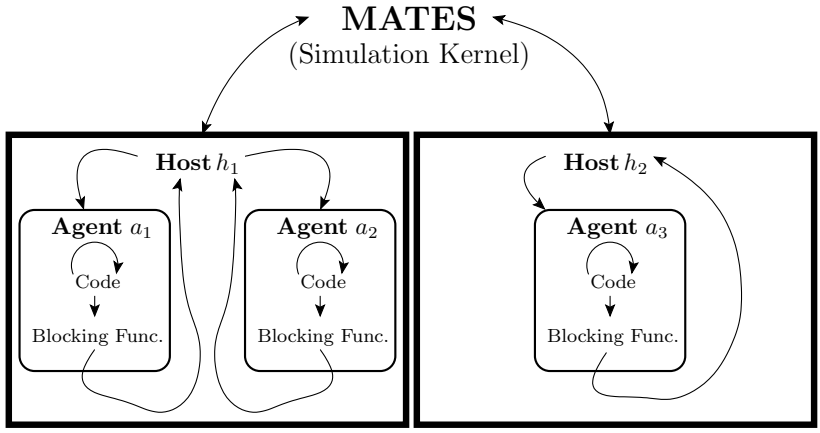


Figure 3.2: Flow of control of the behavioral model in a simulation of two hosts and three agents. MATES first allocates execution time to h_1, which sub-allocates time to its agents a_1 and a_2. When each agent reaches a blocking function, control returns to the simulation kernel. When h_1 is finished, h_2 will be given execution time. Finally, h_2 allocates execution time to its agent a_3.

Suspension of agent execution while preserving execution state allows for a familiar agent development environment and greatly increases the transparency of the simulator. This imparts a sense of "continued execution" from the point of view of the programmer. For example, consider Algorithm 1. This algorithm describes an agent that randomly walks the network, printing "Hello World" at each host visited. Note that lines 3 and 5 of the code are blocking. Ideally, one would like other agents on a host to acquire processing time once the currently-executing agent goes to sleep or migrates.

If implemented in a traditional agent simulator, the HELLO-WORLD-AGENT would have a function that would be called once every simulator iteration. Therefore, to have the same functionality as in Algorithm 1, the iteration function would require a mechanism such as a finite state machine to keep track of the agent's state, as in Algorithm 2. In MATES, however, each agent's main execution function is called only once: when the agent is instantiated. In MATES, one could implement the HELLO-WORLD-AGENT's functionality in the exact same procedural form as in Algorithm 1. MATES accommodates for this by halting an agent's thread when it reaches a blocking function (i.e. a sleep, migration, or yield).

Algorithm 1 HELLO-WORLD-AGENT

```
1: while TRUE do
2:     PRINT("Hello World")
3:     SLEEP(1000)
4:     N ← GET-NEIGHBORING-HOSTS()
5:     MIGRATE-TO-HOST(RANDOM(N))
6: end while
```

Algorithm 2 HELLO-WORLD-AGENT-FINITE-STATE-MACHINE

```
1: if STATE = SLEEPING ∧ TIME() ≥ SLEEPUNTIL then
2:     STATE ← STARTMIGRATION
3: end if
4: if STATE = MIGRATING ∧ MIGRATION-COMPLETE() = TRUE then
5:     STATE ← ∅
6: end if
7: if STATE = ∅ then
8:     PRINT("Hello World")
9:     STATE ← SLEEPING
10:    SLEEPUNTIL ← TIME() + 1000
11: end if
12: if STATE = STARTMIGRATION then
13:    N ← GET-NEIGHBORING-HOSTS()
14:    START-MIGRATION-TO-HOST(RANDOM(N))
15:    STATE ← MIGRATING
16: end if
```

3.2.3 Host Mobility Model

Every host has a *mobility model* that dictates the way in which it moves. During every successive quanta of simulated time, each host's mobility model can dictate its location anywhere within the bounds of the simulated environment (which are a simulation parameter). In most cases, mobility models will move the host to a position adjacent to its current, however this is not a requirement.

3.2.4 Link Connectivity Model

The simulation itself has a *link connectivity model* that defines the conditions under which two hosts have a connection. This allows for simulation of both static and ad hoc networks. In the former case, the link connectivity model is basically a lookup table for connections. In the latter, link connectivity is a function of hosts' locations. The link connectivity model also determines the links' quality: a metric roughly corresponding to signal strength in a wireless network.

3.2.5 Data Transport Model

The simulation also has a *data transport model* that defines the amount of time required for an entity to be sent over a specific link. Here, "time" is actually referring to simulator quanta. This will usually be a function of the link quality and the size of the entity.

3.2.6 Routing

Agents are assumed to route themselves, in the *Active Networking* paradigm [65]. However, it is possible to implement a routing protocol in the simulator. This can be accomplished by having a static agent on each host responsible for carrying out the routing tasks. When an agent needs to know the next hop in a route, it will query its host for a handle to the "routing agent," and query that agent for the route table.

3.3 Implementation

MATES is free, open-source, and is hosted at `http://mates.sourceforge.net/`. The current version of MATES has been implemented in Java, chosen primarily for its ease of its class polymorphism, reflection, and extension. Also, many of the current leading agent architectures are implemented in Java [33, 6]. A screen shot of our implementation of MATES is provided in Figure 3.3. The implementation provides built-in host mobility, link connectivity, and data transport models, however each of these can be overridden.

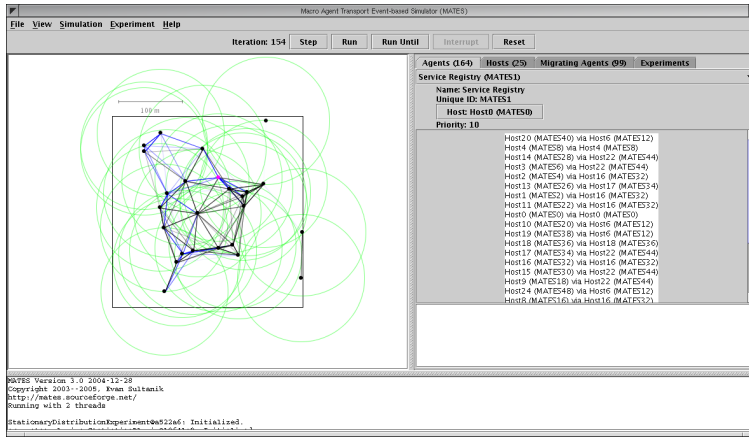


Figure 3.3: Screen shot of MATES' GUI, simulating a mobile ad hoc network of 25 hosts.

Variations of the *Random Walk Mobility Model* and *City Section Mobility Model* [12] have been implemented. The host mobility model is passed a handle to the link connectivity model during host placement. This enables the host mobility model to move hosts in such a way to preserve a certain topology. For example, one might never want the topology to be disconnected. Therefore, the host mobility model can predetermine if a specific host-repositioning will disconnect the network.

The default link connectivity model emulates the *exact connectivity* model for ad hoc network graph generation [3]. Connections are determined by the Euclidean distance between hosts. Each host has a default radio range of 300 meters, which can vary on a host-to-host basis. Also note that, since hosts can have varying radio ranges, the resulting network topology is a directed graph. This means the fact that host X can "hear" host Y *does not* imply that host Y can hear host X.

The default data transport model dictates that transit times are calculated with an inverse linear relationship to link quality. Therefore, agent transit times have an exponential relationship to the Euclidean distance between hosts. By default, the maximum transit time is a constant number of quanta; all traffic takes the same amount of time to transmit. However, each agent may implement an interface such that it can dynamically define its own transit time, allowing for simulation of agents of different size.

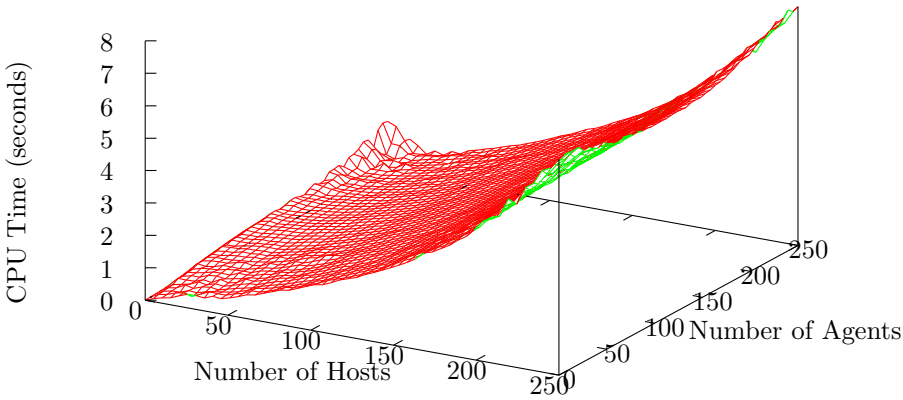


Figure 3.4: Computation time for simulating 500 quanta as a function of the number of agents and the number of hosts.

3.4 Scalability

The two primary parameters that dictate the MATES' computation and memory complexities are the number of hosts and the number of agents in simulation. We profile these complexities by varying the parameters over a simulation of 500 quanta. The hosts moved according to the City Section Mobility Model [12] and each agent performed a random walk on the network. The domain was restricted to a 300 meter square, and hosts' radio ranges were set to 100 meters. The data were collected on a computer with dual AMD Opteron 240 processors, 1GB of RAM, and running Linux kernel `2.6.7-gentoo-r9`. `Blackdown-1.4.2-rc1` was used as the Java runtime environment.

The results for the computational complexity of running the experiment are given in Figure 3.4. The times recorded in the figure are the amount of CPU time that elapsed between the start and end of the experiment; the time required for MATES to initialize was not included. The data indicate that computation time scales linearly with respect to the number of agents in the simulation, and scales polynomially with respect to the number of hosts in the simulation.

The results for the memory-space complexity of running the experiment are given in Figure 3.5. These data are inherently error-prone, as the Java virtual machine will often allocate more memory than it actually requires. The data indicate that memory usage scales linearly with respect to the number of agents

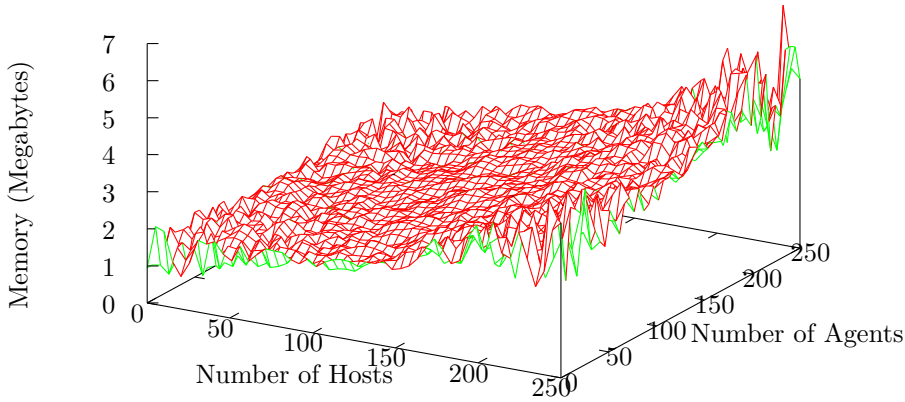


Figure 3.5: Memory required for simulating 500 quanta as a function of the number of agents and the number of hosts.

in the simulation, and scales polynomially with respect to the number of hosts in the simulation.

Intuitively, the complexity of the simulation is not only dependent on the number of hosts, but also the topology of the network. For example, a completely disconnected network (with no edges) will simulate much faster than a completely connected network. Therefore, we also investigate the correlation between the complexity of the simulation and the edge density. Figure 3.6 shows the average topology diameter and average host degree over the experiments as a function of the number of hosts. Since the mobility of the hosts are constrained to a 300 meter square, adding more hosts linearly increases the average host degree. Likewise, once the square becomes saturated with hosts (which occurs at about 45 hosts) the average topology diameter becomes constant.

Figure 3.7 shows the computation time and memory usage for the experiments with respect to the average host degree. Both computation time and memory usage are highly correlated, however the computation time data have very low variance. This indicates that the computational complexity is much higher with respect to the number of edges in the network than with respect to the number of agents.

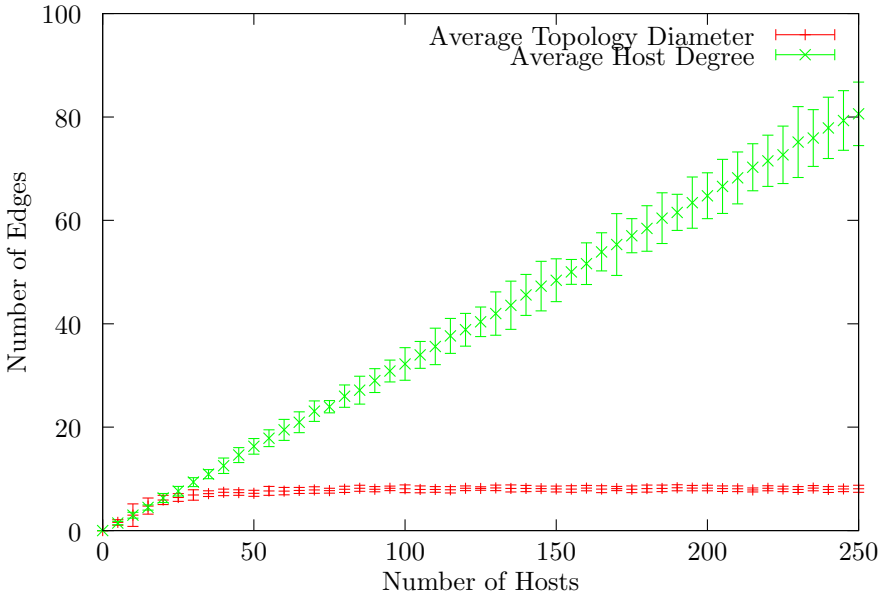


Figure 3.6: Topological statistics recorded over the experiments. When the hosts are bounded in a physical space, the average topology diameter remains constant while the average host degree increases linearly.

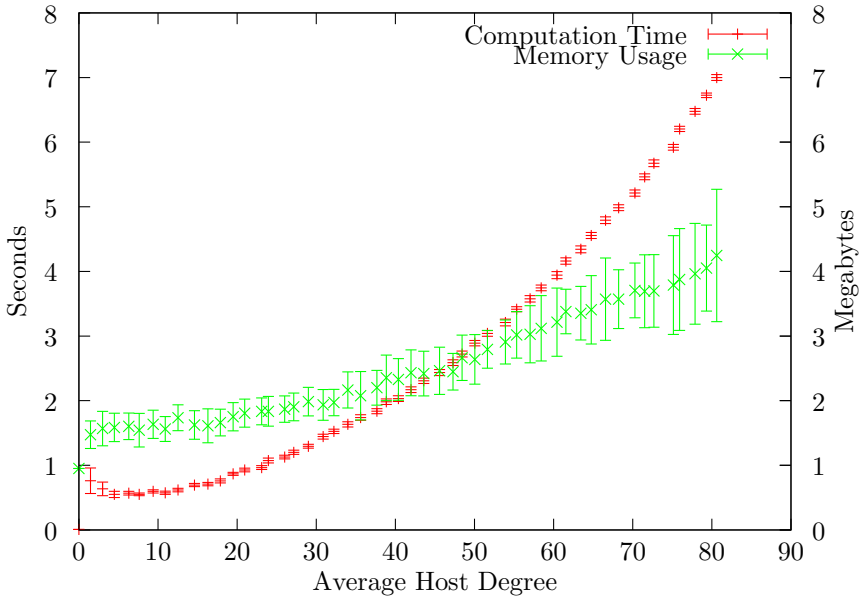


Figure 3.7: Computation time and memory usage as a function of average host degree. Note that both values are highly correlated to host degree.

3.5 Examples & Validation

MATES has already been used for experimentation in various areas of both artificial intelligence and networks research. In these experiments, simulated data from MATES has shown to correlate strongly to equivalent experiments conducted on live MANET [53]. Networking topics studied with MATES include MANET power management and localization (based on partial GPS information and signal strength). The remainder of this section outlines artificial intelligence research conducted using MATES.

Agent Population Management. One of the first experiments conducted using MATES entailed analysis of an ant algorithm for estimating and managing the optimal number and location of services in a multi-agent system deployed over a MANET [53]. Specifically, service availability[3] was measured under variation of:

- number of hosts on the network;
- link models; and
- network mobility models.

The experiments were conducted both with and without the ant-inspired managing services. MATES' random number generation architecture allowed for the exact same sequence of host movement during both control and variable experiments.

The same experiments were then conducted on the Secure Wireless Agent Testbed [62], running on iPAQ PDAs. The agents were implemented using Lockheed Martin's Extendible Mobile Agent Architecture (EMAA) [33]. MATES' "continued execution" architecture allowed for agent code to be ported almost directly from MATES to EMAA. As detailed in [53], the experimental results from MATES mirrored the real-world empirical results from the SWAT almost exactly.

Agent Ecosystems. MATES has also been used to simulate agent ecosystems, in which the supply and demand for virtual food is used to optimize the stability of an agent system [52, 54]. In other words,

- Agents collect food as a reward for completing a task.
- Agents consume food regularly to stay alive.
- Agents that exhaust their food supply die.

[3] "Service availability" is a metric for a host's ability to efficiently utilize a service on the network.

- An abundance of food causes new agents to spawn.

A formal model of this system was created, and was validated under MATES.

Service-Based Computing. MATES is one of the few environments available in which one is able to extensively test service-based computing and the agent paradigm in the domain of dynamic networks. Because of this, MATES was chosen to simulate and study the feasibility of service-based computing on disruption and delay-prone networks [29]. Multiple service discovery methods were compared on simulated satellite relay, interplanetary, and mobile ad-hoc networks. It was shown that, even in optimal network conditions, effective agents must perceive, reason, and act on network state.

3.6 Limitations

A major limitation of MATES, in its current implementation, is its atomic representation of time. This means that all user-created simulation components (i.e. agents) must agree on a mapping from quanta to "simulated time."

MATES was not designed for large networks, and as such does not make an effort to aggregate or parallelize homogeneous entities and tasks. With that said, MATES has exponential computational complexity with respect to the number of agents and hosts, with memory usage scaling linearly. This undesirable performance, however, is neither a function of MATES' design nor implementation; it is due to the complexity of the underlying problem.

Chapter 4

> “It is startling to realize how much unbelief is necessary to make belief possible. What we know as blind faith is sustained by innumerable unbeliefs.”
>
> —Eric Hoffer
> Section 56, *The True Believer*

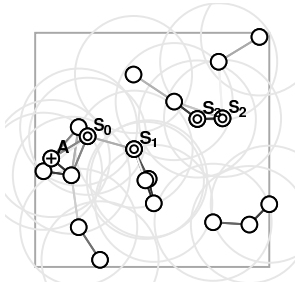
Coping with Local Knowledge

4.1 Introduction

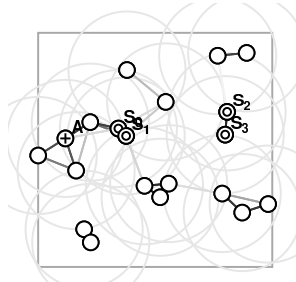
The focus of this chapter is the support for distribution of global state information—enabling service discovery within a Service Oriented Architecture (SOA)—on emerging wireless networks.

Example of Service Oriented Architectures on MANETs. Figure 4.1 illustrates the stark differences between a MANET and the enterprise computing scenarios that are typically the doman of SOAs. An application on host A requires services available from service providers S_0, S_1, S_2, and S_3. Initially, the host labeled “A” is indirectly connected to S_1 through S_0, as in Figure 4.1(a). As the scenario progresses in Figure 4.1(b) and the hosts on the MANET continue their random walk, providers S_0 and S_1 move in opposite directions. By Figure 4.1(c), the situation has reversed and S_1 is now the closest service provider. In Figure 4.1(d) host A loses connectivity to S_0 completely. By discovering available services and intelligently choosing service providers, an agent may more efficiently utilize resources and recover from errors. In this example, agents from A would switch from S_0 to S_1 when it becomes the closest provider.

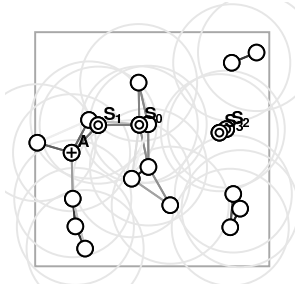
As an example of an important application domain, the synthetic aircraft world of Tambe *et al* [64] affirms the problem of service discovery for multi-agent planning systems. A group of agent-driven helicopters are deployed on a battlefield; one helicopter “disappears.” In this case “disappearance” might mean “over a ridge,” “out of communication range,” or “destroyed.” How do the



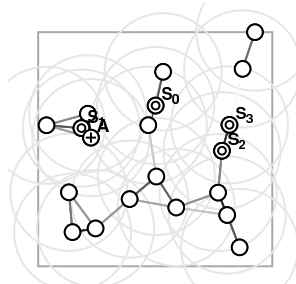
(a) Host A is connected to S_1 through S_0.



(b) The service providers move in opposite directions.



(c) S_0 is no longer directly connected to Host A.



(d) Host A is no longer connected to S_0.

Figure 4.1: A scenario showing web services on a MANET. Agents on Host A require services offered by each provider S_i.

remaining agents decide if a node, or service on that node, has become unavailable and re-planning is required? Existing work assumes that such information is instantaneously announced to all nodes and agents, a process which fails to take into account the realities of information propagation on peer-to-peer and wireless networks. In the synthetic aircraft example, knowledge and intervention of a human agent is required to alert the agent system that the helicopter has disappeared. Furthermore, recent research has shown that no fixed memory deterministic algorithm can locate a service in a network in a fixed amount of time [26]. We shall use the synthetic aircraft domain as a running example in this chapter.

In some domains services will be provided by mobile agents whose location in the network changes over time. For example, a certificate authority might be required for secure communication between helicopters in the synthetic aircraft domain. This server could be encapsulated by a mobile agent capable of reasoning about the network [51]. The agent might then continuously migrate to portions of the network with low volatility. For instance, the agent might migrate to helicopters less likely to be removed from the network. To improve performance and minimize latency, heuristics for such migration might include proximity to the geographic center of the group or association with the center of the network topology graph. Therefore, a method for pro-actively tracking the location of services in dynamic networks is required.

Research Approach and Contributions. This chapter presents an approach to service discovery in MANET environments. The current literature on SOAs does not yet address these issues directly, nor has there been any substantial theoretical or empirical studies of approaches to solve the discovery problem for SOAs on MANETs.

On non-fault tolerant, peer-to-peer and ad hoc wireless networks, web services may become unavailable due to network partitioning, traffic congestion, or attack—thus inhibiting service discovery, binding and composition. The approach in this chapter exploits the emergent behavior of mobile service discovery agents that stochastically sample the hosts on a wireless network. Using a model of behavior derived from the theory of random walks, these agents develop a belief about the state of network services and propagate these beliefs to the hosts as they travel the network. Furthermore, this model can be used as a performance profile for time-critical reasoning. By augmenting the discovery agents' knowledge with domain and network semantics, a fixed-memory randomized method for approximating the location of a service in a stochastic environment with a probabilistic certainty in a fixed amount of time is proposed. Empirical analyses are provided to show the accuracy of the technique on simulated wireless networks.

To address the unpredictability and constrained resources of a MANET, communicating applications must be able to determine and adapt to available

resources. Network-aware mobile agents are uniquely well suited to this task [28, 15]. Such agents collect and transmit data and messages, making informed decisions in transit based on properties such as network topology. services may be used by such an agent to define, discover, and utilize other agents providing services necessary in accomplishing some task. This enables an agent to conserve resources by finding the most accessible service providers and recover from failures such as network partitions that disconnect services.

Chapter Organization: This chapter is organized as follows. §4.2 presents the technical formulation of the approach, where §4.2.1–§4.2.6 provide the theoretical model. §4.2.7 proposes examples of using the model. §4.3 empirically validates the approach through simulation in MATES. §4.4 discusses the approach, including possible applications and domains, limitations, and future work.

4.2 Technical Approach

This section presents an approach to accurate, on-line service discovery using a set of service monitoring agents, A, randomly walking a set of hosts, H, on a peer-to-peer network. As these agents traverse the network, they construct and update a model of service locations. This knowledge is deposited on each host for use by other agents.

As seen in Figure 4.2, each agent's walk is dictated by the underlying network topology. Algorithmically, this method is very simple, exploiting the emergent behavior of the agent collective. As the service discovery agents randomly walk the network, they remember the time and host at which each service was seen. Other agents in the system can then query the service discovery agents for their beliefs of service locations. However, there is no guarantee that all service discovery agents visiting a host have encountered a service recently (or even at all) during their walks. Furthermore, since the migration patterns of the service discovery agents are stochastic, there is no deterministic guarantee as to when a service discovery agent will visit a specific host. These problems are addressed by the model proposed in the remainder of this chapter.

This approach has four important advantages over alternatives, such as naïve message passing or broadcasting:

1. Very little network bandwidth is used, and bandwidth usage scales linearly with respect to the number of discovery agents. This is an important issue for resource-constrained mobile devices and large-scale peer-to-peer networks.

2. Mobile code technology allows for straightforward deployment on heterogenous networks.

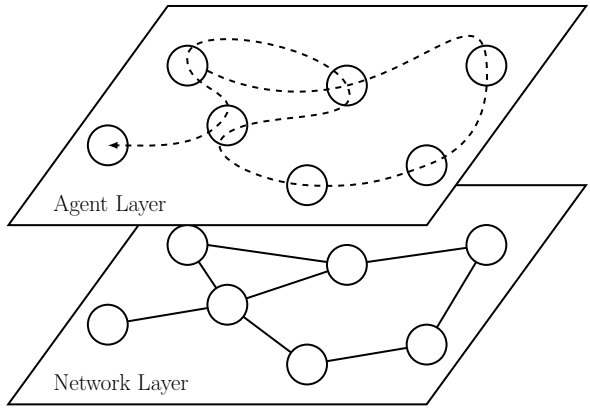


Figure 4.2: An agent randomly walking a peer-to-peer network.

3. Services only register with a local listing, not with centralized registrars.

4. Properties of random walks are relatively easy to model and perform inferences upon.

This work develops a mathematical model for the behavior of these agents, allowing for time-critical reasoning and probabilistic inferences to be made about the system. We use properties of Markov chains to model the probability that an agent will visit a specific host, a binomial distribution to model the probability that an agent has seen a service, and another binomial distribution to develop an expected value for the number of agents that will visit a specific host in some amount of time.

4.2.1 Random Walking Mobile Agents

The agents' task environment, a dynamic peer-to-peer network, is stochastic, dynamic, and continuous. There exists a delay between actual topology changes and the propagation of knowledge of these changes throughout the network. The discovery agents do not have a goal, *per se*, their sole purpose is to randomly walk the network and gather information. Their percepts are comprised solely of the set of services available at the current host, S_h, and the set of hosts neighboring the current host, $\{x \in H | E_{h,x} > 0\}$ (where $E \subseteq H \times H$ is the set of edges in the topology and the notation $E_{x,y}$ denotes the weight of the edge from node x to node y). Edge weights represent transition probabilities between hosts in the network; for most networks these will be uniform. However, ad hoc wireless networks might correlate edge weights to link quality between hosts to avoid agent migration over unreliable links.

Agents' actions are comprised solely of migrating to a neighbor host from their current host. At each host agents query for services and store their collected data in memory (along with a timestamp). The agents' itineraries are dictated by the network. Successor hosts for migration are randomly selected from available neighboring hosts.

4.2.2 Predicting Agent Arrival at a Host

The frequency of agent visits can be predicted by developing a function, $\psi : \mathscr{N} \times S \times H \to [0, 1]$, for the probability that at least one agent $a \in A$ with knowledge of a service $s \in S$ will visit a specific host $h \in H$ in a duration of time δ. $\mathscr{N}$ is the set of all possible local state descriptions, where each $N \in \mathscr{N}$ is a local state description represented by a tuple containing the following elements:

δ - length of the time interval;
$\nu : \mathbb{R} \to [0, 1]$ - function returning the probability that at least one discovery agent will visit h in a duration of time;
η - number of instances of the service s;
$|H|$ - cardinality of the set of hosts;
$|A|$ - cardinality of the set of agents;
ℓ - average time needed for an agent to hop between neighbors; and
τ - maximum desired amount of time since an agent last saw the service.

$\psi(N, s, h)$ is therefore a probability distribution over the space N for a given service and host. We decompose $\psi(N, s, h)$ into three component distributions:

π_h probability that a discovery agent will be at host h at any given time;
$\hat{\nu}$ an approximation of ν; and
$P(n \geq 1)$ probability that a discovery agent will see at least 1 instance of the service s in time τ.

These distributions need not be calculated *a priori*. A method for approximating π_h is given in §4.2.3 which is then used to develop $\hat{\nu}$ in §4.2.4. $P(n \geq 1)$ is then defined in §4.2.5. Finally, the approximation of $\psi(N, s, h)$ is constructed from these component distributions in §4.2.6.

Intuitively, the larger $|A|$ and the smaller $|H|$ and ℓ the more often agents will visit hosts. τ is meant to be a measure of the "age" of each agent's knowledge of the services. Since global time synchronization is a difficult problem in some domains, τ can also be replaced by a heuristic that approximates the age of the agent's data. For example, the number of hosts the agent has visited since it last saw an instance of the service could be used.

4.2.3 Mathematics of Random Walks

Random walks along graphs can be modeled as finite Markov chains, and therefore both share many of the same properties. Gkantsidis, et al., experimentally showed that, when searching for items occurring frequently in a network, random walks perform better than flooding (for the same number of network messages) in certain cases [21]. In order to predict the frequency of randomly-walking agents visiting a specific host, though, we must first develop a probability that an agent will be on a specific host at any time.

The PageRank algorithm [11] determines the probability that a random web surfer will be on a given web page at any time. PageRank employs Markov chains to model random walks along the graph of the Internet. One can therefore use PageRank to determine the probability that an agent randomly walking a network will be visiting a specific host at any given time. The first eigenvector, $\vec{\pi}$, of a graph's adjacency matrix, J, is fundamentally intertwined with the *stationarity* of the graph. The eigenvector $\vec{\pi}$ corresponds to the eigenvalue λ_1 such that $\vec{\pi}J = \lambda_1\vec{\pi}$. PageRank exploits this fact and provides a means for approximating the primary right eigenvector of an adjacency matrix. We present Algorithm 3 as an adaptation of PageRank for our domain.

Algorithm 3 AGENT-VISITATION-PROBABILITIES($J, d, iterations$)

Require: J is the adjacency matrix representation of the network, d is a real number damping factor in the range $[0, 1]$ (usually set to 0.85), *iterations* is the number of iterations to run, and all elements of $\vec{\pi}$ are initialized to $\frac{1}{|H|}$.

Ensure: $\vec{\pi}$, the primary right eigenvector of J, contains the probabilities that a random agent will be on any node

```
for i = 1 to iterations do
  for j = 1 to |H| do
    sum ← 0
    for k = 1 to |H| do
      if J_{k,j} > 0 then
        links ← |{x | (1 ≤ x ≤ |H|) ∧ (J_{k,x} > 0)}|
        if links > 0 then
          sum ← sum + π_k ÷ links
        end if
      end if
    end for
    π_j ← (1 − d) + d· sum
  end for
end for
```

$\vec{\pi}_h$ (*i.e.*, the element of the vector $\vec{\pi}$ corresponding to host h) is then the probability that an agent $a \in A$ is on a specific host $h \in H$ at any given time.

4.2.4 Approximating ν

ν is defined as a function returning the probability that at least one service discovery agent will visit the host in time δ. By definition, one can use $\vec{\pi}$ to develop an estimator of ν:

$$\hat{\nu}(0) \mapsto \vec{\pi}_h \text{ when } |A| = 1\,, \tag{4.1}$$

however, a binomial distribution must be used to define $\hat{\nu}$ for all values of δ and $|A|$:

$$\begin{aligned} \hat{\nu}(\delta) &\mapsto 1 - \left(1 - \left(1 - (1 - \vec{\pi}_h)^{|A|}\right)\right)^{\delta} \\ &= 1 - (1 - \vec{\pi}_h)^{\delta\,|A|}\,. \end{aligned} \tag{4.2}$$

In other words, $\hat{\nu}(\delta)$ is 1.0 minus the probability that *none* of the $|A|$ agents will visit in time δ.

Unfortunately, it is not sufficient to define the mapping of $\psi(N, s, h)$ solely based upon $\hat{\nu}$ because not all agents that visit a host have recent enough data about the service being located. Therefore, a function defining the probability that the random agent visiting host h will have a recent-enough[1] memory of service s is needed.

4.2.5 Accounting for τ

ν provides a prediction mechanism for the number of random-walking agents that will visit a host. However, ν does not take into account the fact that these agents may not have recently seen an instance of the service s. τ is an element of the state description that dictates the maximum allowable amount of time since a service discovery agent has seen the service. Therefore, τ must be incorporated into the probability.

Let $H' \subseteq H$ be the set of hosts that an agent visits in time τ and $S' \subseteq S$ be the set of services an agent sees in time τ. The expected value for the number of hosts the agent will visit is given by:

$$\langle |H'| \rangle = \left\lfloor \frac{\tau}{\ell} \right\rfloor\,. \tag{4.3}$$

It is assumed that, due to the mobility of the network and its associated random topology, the probability of a randomly-walking agent visiting a host with a service is normally distributed. This claim is empirically investigated in §4.3.2. We can then say $\frac{\eta}{|H|}$ is the probability that an instance of the service exists at a randomly-selected host. The probability that an agent walking the network

[1] By "recent-enough" we mean "of age less than or equal to τ."

will encounter an instance of s in time τ can then be modeled as a binomial distribution of $|H'|$ trials:

$$P\left(n \,\middle|\, |H'|\right) = \binom{|H'|}{n} \left(\frac{\eta}{|H|}\right)^n \left(1 - \frac{\eta}{|H|}\right)^{|H'|-n} \tag{4.4}$$

where n is the number of instances of s discovered:

$$n = |\{x | x \in S' \wedge x = s\}| \,. \tag{4.5}$$

Summing (4.4) over all n where $n \geq 1$:

$$\begin{aligned} P\left(n \geq 1\right) &= \sum_{i=1}^{|H'|} \left(\binom{|H'|}{i} \left(\frac{\eta}{|H|}\right)^i \left(1 - \frac{\eta}{|H|}\right)^{|H'|-i} \right) \\ &= 1 - \left(1 - \frac{\eta}{|H|}\right)^{\left\lfloor \frac{\tau}{\ell} \right\rfloor} . \end{aligned} \tag{4.6}$$

$P(n \geq 1) = P(\exists x, x \in S' \wedge x = s)$ is the probability that an agent has seen at least one instance of the service s while walking the network in time τ.

4.2.6 Constructing $\psi(N, s, h)$

Given the probability that a service discovery agent will visit a host, ν, and the probability that a service discovery agent will have seen an instance of the service being sought, $P(n \geq 1)$, we can define the mapping of $\psi(N, s, h)$.

We assume the event that an agent has seen an instance of service s is independent of the agent visiting host h. Let $A' \subseteq A$ be the set of service discovery agents with knowledge of s that visit h in time δ. In actuality, ψ will be binary: either we received an useful agent during the interval δ or we did not:

$$\psi(N, s, h) \quad \mapsto \quad \begin{cases} 1.0, & A' \neq \emptyset \\ 0, & A' = \emptyset \end{cases} .$$

Since the emptiness of A' is unknown *a priori*, and we really want the *probability* that the event will occur, we need to derive the probability that $A' \neq \emptyset$. This is trivial when $\delta = 0$:

$$\begin{aligned} \psi(N, s, h) &\mapsto P(A' \neq \emptyset) \\ &\mapsto P(n \geq 1)\, \nu. \end{aligned}$$

To extend this mapping for all values of δ, we need another binomial distribution over δ trials:

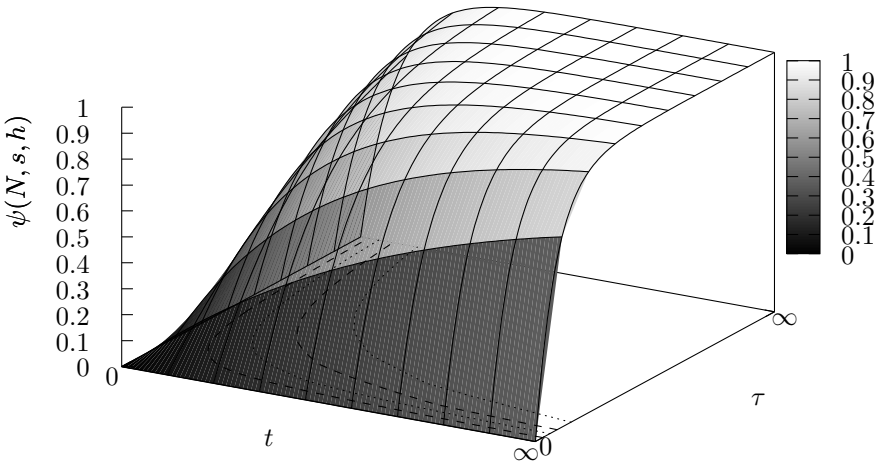


Figure 4.3: Profile of $\psi(N, s, h)$.

$$\begin{aligned}
\psi(N, s, h) &= P(A' \neq \emptyset) \\
&= \sum_{i=1}^{\delta} \left(\binom{\delta}{i} \left(P(n \geq 1)\,\nu\right)^{i} \left(1 - P(n \geq 1)\,\nu\right)^{\delta - i} \right) \qquad (4.7) \\
&= 1 - \left(1 - \nu + \nu \left(1 - \frac{\eta}{|H|} \right)^{\left\lfloor \frac{\tau}{\ell} \right\rfloor} \right)^{\delta}. \qquad (4.8)
\end{aligned}$$

Note that although Equation 4.7 uses summation and the binomial coefficient on the variable δ, this is strictly for elucidation of our derivation; the resulting expression in Equation 4.8 does not preclude any of the parameters—including δ—from being continuous. The profile of $\psi(N, s, h)$, as a function of δ and τ, is given in Figure 4.3.

The function $\psi(N, s, h)$ is useful for predicting the number of randomly walking agents that have seen service s in time τ and will also visit host h in time δ.

4.2.7 Using $\psi(N, s, h)$

Take the synthetic aircraft domain as an example; suppose an agent needs to locate a service in a fixed amount of time. If s does not exist, the agent will need to re-plan. If the service is only available from the helicopter that has disappeared, the agent will waste its time trying to look for the service. Using $\psi(N, s, h)$, the agent can predict if it will hear from any of the service discovery

agents in time δ. If $\psi(N, s, h)$ returns a low probability, the agent will know to immediately re-plan without waiting for any of the service-discovery agents to arrive.

Now let us consider the same scenario, but in terms of service composition. In this case, the service being sought is really composed of a set of services, $S = \{s_1, s_2, \ldots, s_n\}$. Since the events of receiving location information about each of the constituent services in S can be assumed independent, the availability of S would simply be the product of the individual $\psi(N, s, h)$ probabilities for each $s \in S$.

4.3 Empirical Validation

Network simulation is accomplished using MATES. Variation of the Network links are determined by the Euclidean distance between hosts. The hosts' movements are bounded by a 1200x1200 meter box, and each host has a radio range of 300 meters. At the beginning of each experiment, hosts are placed randomly in the box and given a random direction. Both agents and instances of the service s are randomly distributed among the hosts. Every iteration of the time quanta of the simulation:

1. Each host's direction is randomly chosen by either maintaining in its current heading (with probability 0.6), rotating 45° clockwise (with probability 0.2), or rotating 45° counter-clockwise (with probability 0.2);

2. hosts move forward one meter in their respective directions;

3. agents not currently in transit migrate from their current host to a randomly-selected neighbor host (as described in §4.2.1). Agent transit times are calculated with an inverse exponential relationship to the Euclidean distance between hosts. The average transit time for agents, ℓ, is 1 iteration;

4. every instance of s is treated as a mobile agent; each service migrates to a random neighbor host as described above; and

5. every δ iterations, each host uses $\psi(N, s, h)$ to develop a probability of a knowledgeable agent visiting it in the subsequent δ iterations. The actual frequency of knowledgeable agent visits, $\frac{|A'|}{\delta}$, is also recorded.

The results we present are from 30 runs of the simulation, 300000 time quanta each, with 30 hosts, 15 agents, and 3 instances of the service s.

4.3.1 Accuracy of PageRank.

Figure 4.4a illustrates Algorithm 3's error as it converges to the true stationary distribution value. In the case of a static network, the algorithm will converge

monotonically to an error of zero. In the case of a dynamic network, the algorithm tends toward zero, but is not guaranteed to converge due to the changing topology.

Figure 4.4b illustrates Equation (4.2)'s accuracy in predicting the frequency of agent visits. One can see that the prediction approximates the actual value very closely and is also strongly correlated. The average coefficient of correlation between these variables over the set of 30 runs is 0.68. The average bias for the predicted probability is 0.01.

4.3.2 Verification of Services' Distribution.

The frequency distribution for the number of instances of service s agents saw was recorded. The Shapiro-Wilk normality test returns a value of 0.5532 for the experimental distribution (with an infinitesimally small P-value), meaning the experimental distribution does partially deviate from normality. Nonetheless, this result implies that one *cannot* say that the data are *not* normally distributed. The deviation from normality can be explained by the low probability of an agent seeing an instance of the service; the data are skewed more toward an F-distribution. However, as demonstrated by the accuracy of $\psi(N, s, h)$ in Figure 4.5, it is reasonable to assume this distribution is normal.

4.3.3 Accuracy of $\psi(N, s, h)$.

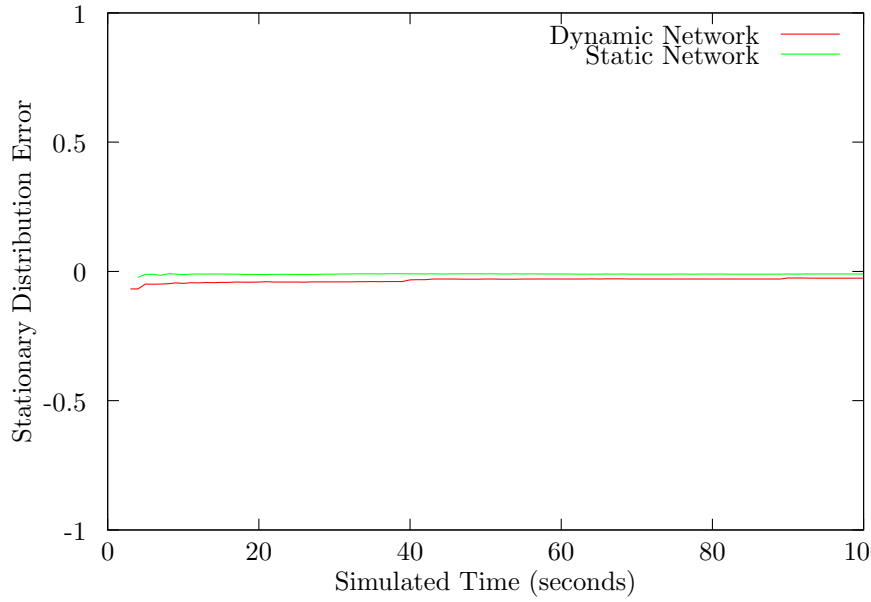
Figure 4.5 illustrates Equation (4.8)'s accuracy in predicting the frequency of knowledgeable agent visits. One can see that the prediction approximates the actual value very closely and is also correlated. The average coefficient of correlation between these variables over the set of 30 runs is 0.60. The average bias for the predicted probability is -0.03.

4.4 Discussion

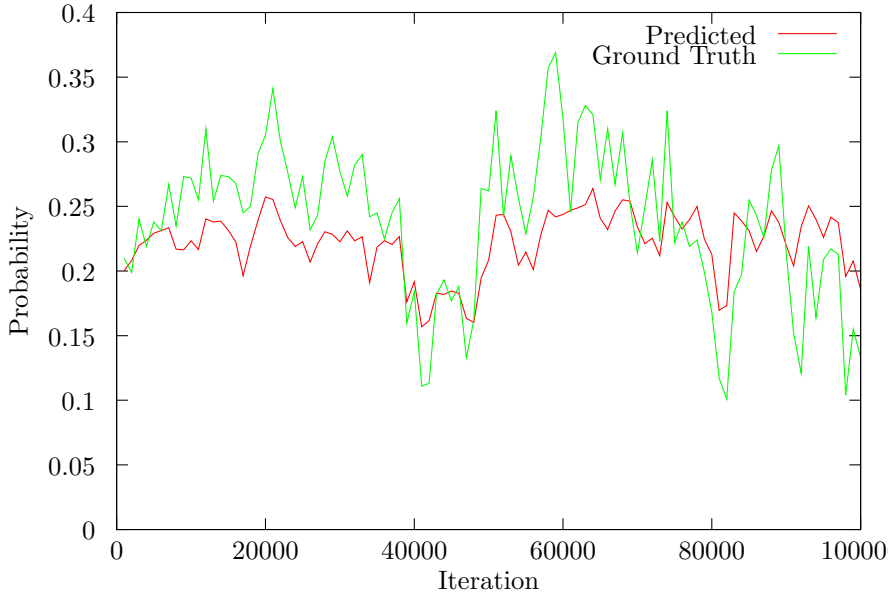
Examples of Applying the Technique. The information updates provided by the service discovery agents can be used to create new capabilities for multi-agent systems operating on peer-to-peer networks. Examples include:

- Learning availability thresholds, where if a service or host has not been "seen" by a discovery agent the remaining hosts and agents can (with high probability) assume that it has become unavailable and re-plan accordingly.

- Network triage, where the disappearance of discovery agents or their lack of contact with vital network nodes can be used to infer that the network has been damaged, compromised or segmented.



(a)



(b)

Figure 4.4: (a) depicts the error in the predicted agent frequency, ν, in static and dynamic networks over a 100 quanta (second) simulation using the PageRank algorithm. The error converges to 0 over time. (b) compares the estimator given in Equation (4.2) to the actual value over the course of a simulation. The means are highly correlated.

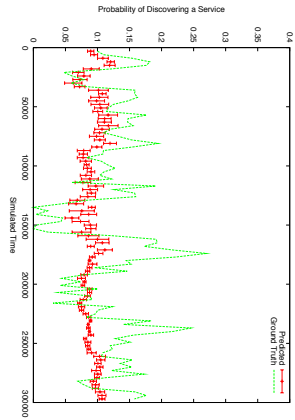


Figure 4.5: Prediction, $\psi(N, s, h)$, and actual agent visitation probabilities for a 300000 quanta (simulated seconds) experiment.

- Time-critical reasoning, where hosts use information provided by discovery agents as a performance profile for how long time-critical functions might execute if they depend on remote services.

- Optimizing the number of agents, where, given an expected value for ν (which can be calculated using an expected network topology), one can use Equation (4.8) to compute the optimal number of agents needed to achieve a required frequency of service discovery agent updates.

Limitations. Elements of the state description, N, can be both variable and unknown in some domains. Therefore agents might develop beliefs about the values of these parameters, such as ℓ and η. Furthermore, there are more efficient ways for the agents to traverse the network (i.e. self-avoiding walks). Research into these alternate techniques is required, however mathematically modeling them is more complex than with random walks.

Although ν can be defined from $\vec{\pi}_h$, developing a belief of the global network topology, J, is a difficult problem on dynamic peer-to-peer networks. Pro-active routing algorithms for ad hoc networks often define a protocol to propagate this information, but this is expensive. The amount of memory/bandwidth required for each network message can in the worst case be $O(n^2)$ (to transmit the entire adjacency matrix). Since computation of $\psi(N, s, h)$ really only requires $\vec{\pi}_h$ (not the entire adjacency matrix J), $\vec{\pi}_h$ could be inferred by the observed frequency of agent visits at host h over a period of time. Research is required to evaluate this.

Future Work. Our proposed method for propagating service location information throughout the network can be used as a heuristic for mobile agent-based search. For example, hosts could cache data brought to them by the random walking service discovery agents. In doing so, each host would develop an index (or "belief") of the locations of services. These beliefs will become more accurate in conjunction with a host's proximity to the service. Therefore these beliefs, along with the timestamp of when the agent last saw the service, could be used as an A* search heuristic when locating the service. Work is needed to prove the feasibility, admissibility, and accuracy of this heuristic; we are in the process of using the SWAT to do so.

Further experimentation is required to ascertain the effect of varying parameters, such as ℓ. In addition, the effect of CPU and network bandwidth limitations is not clear. Finally, the accuracy of our approach on networks of heterogeneous hosts is unknown.

Chapter 5

> “Below every tangled hierarchy lies an inviolate level.”
>
> —Douglas Hofstadter
> *Gödel, Escher, Bach: an Eternal Golden Braid*

Multi-Agent Coordination

A key step towards addressing the distributed scheduling problem is to devise appropriate representations. In this chapter, we have three goals in mind. We desire a representation that

1. captures the problem's distributed multiagent aspects,
2. gives formal guarantees on solution quality, and
3. is amenable to distributed solving by existing methods.

In regard to the first goal, the Coordinators Task Analysis Environmental Modeling and Simulation (C_TÆMS) representation [10] is a general language (based on the original TÆMS language [20]) that was jointly designed by several multiagent systems researchers explicitly for multiagent task scheduling problems. C_TÆMS is a Hierarchical Task Network (HTN) style representation where the task nodes in the network have probabilistic utility and duration. Utility of tasks are aggregated up the hierarchy using Quality Accumulation Functions (QAFs). The most primitive task nodes are called methods and the multiagent distributed aspect of the problem is modeled by associating methods with different agents. C_TÆMS is an extremely challenging class of scheduling problem and has been adopted as a common challenge problem domain representation for distributed multiagent task scheduling research.

While the level of expressivity of C_TÆMS is required for some domains, it may not be required for all domains of interest. Also, the expressivity comes at a cost. It is arguable that the primary concern in the design of the C_TÆMS language was to comprehensively express the complexities of the distributed scheduling problem, while less concern was placed on goals two and three enumerated above. Indeed, current distributed solution techniques can not be applied straightforwardly or are unable to provide solution quality guarantees. Thus, to make progress on this difficult problem, we begin by contributing a

set of representational compromises for C_TÆMS. In this chapter, we restrict ourselves to a subset of the C_TÆMS language that retains its distributed multi-agent nature but abstracts away the probabilistic quality and duration of tasks by replacing the probability distributions with their expected values. We also restrict ourselves to summation as the most natural type of QAF. We show that even this restricted subclass of C_TÆMS problems is $\mathcal{NP}$-HARD.

5.1 Formalization

This section formalizes the notion of a DCOP and specifies the subset of C_TÆMS on which we focus. We then analyze this subset, proving that it is sufficiently complex to remain $\mathcal{NP}$-HARD.

5.1.1 Modeling Planning Problems

C_TÆMS is a derivative of the TÆMS modeling language [20] and can be used for representing instances of task domains of the distributed scheduling problem. Unlike other HTN representations, C_TÆMS emphasizes modeling of the inter-relationships between tasks more so than those between agents and their environment [42]. For the sake of exposition and formalism, we represent C_TÆMS instances using set theory, whereas the actual specification is a grammar. Our notation covers a simplified subset of C_TÆMS; specifically, we use expected values for parameters whereas C_TÆMS, a discrete stochastic modeling language, uses probability distributions. We represent a C_TÆMS instance as a tuple $(N, \mathscr{M}, \mathscr{T}, E, G, \mu, \phi, \omega)$, where

- A is a set of agents;
- $\mathscr{M}$ is a set of methods;
- $\mathscr{T}$ is a set of tasks;
- E is a set of "Non-Local Effects" (*i.e.* precedence constraints on execution time);
- G is a special task, $G \in \mathscr{T}$, known as the "Task Group;"
- μ is a function $\mu : \mathscr{M} \to A$ mapping methods to agents;
- ϕ is a function $\phi : \mathscr{M} \cup \mathscr{T} \to \mathscr{T} \cup \{\emptyset\}$ that maps methods and tasks to their parent task (a method may not be a parent); and
- ω is a quality accumulation function $\omega : \mathscr{M} \cup \mathscr{T} \to \mathbb{R}^+ \cup \{0\}$ that returns the quality of a method or task. For $T \in \mathscr{T}$, $\omega(T)$ is usually defined as a function of the associated qualities of all $X \in \phi^{-1}(T)$.

Each $M \in \mathscr{M}$ is itself a tuple, (l, u, d), where

- l is the earliest start time of the method;
- u is a deadline for the method; and
- d is the expected duration of the method.

It is assumed that ϕ creates parent-child relationships such that the resulting hierarchy is a tree rooted at G. Note that the range of ϕ ensures that all methods

are leaves.

Each $e \in E$ is a function $e : (\mathscr{M} \cup \mathscr{T}) \times (\mathscr{M} \cup \mathscr{T}) \to \mathbb{B}$ mapping pairs of methods and tasks to a boolean. Let $\varphi : \mathscr{T} \times (\mathscr{M} \cup \mathscr{T}) \to \mathbb{B}$ be a function such that

$$\varphi(X, Y) \mapsto \textsc{True} \iff \phi(X) \mapsto Y,$$

and let φ^+ be the transitive closure of φ. $\varphi^+(X, Y)$ implies that X is in the subtree rooted at Y. Therefore,

$$\begin{aligned} e(X,Y) \implies (\forall X', Y' \mid \varphi^+(X', X) \wedge X' \text{ is a method} \\ \wedge \varphi^+(Y', Y) \wedge Y' \text{ is a method} : \\ X' \text{ must have finished executing before } Y' \text{ can start}). \end{aligned}$$

In other words, $e(X, Y)$ means that all methods in the subtree rooted at X must have finished executing *before* any of the methods in the subtree rooted at Y may be executed. Other types of Non-Local Effects (NLEs) exist within C_TÆMS, but we will focus on this type of "enables" effect in this chapter. For each $e \in E$, if the transitive closure $e^+(X, Y)$ maps to True, X is said to precede Y in an "NLE chain."

Note that $l \leq u$, but $l + d$ might not necessarily be less than or equal to u. Each $T \in \mathscr{T}$ is defined as a similar tuple, except tasks (as a type of virtual method aggregator) do not have explicit durations. Also, these bounds on execution time are inherited from parents. In other words, $l_{\phi(X)} \leq l_X$ and $u_{\phi(X)} \geq u_X$.

A "schedule" is a grammar within the C_TÆMS specification for defining the chosen start times of methods; it can be formalized as a function $s : \mathscr{M} \to \mathbb{N}_0 \cup \{\emptyset\}$ mapping methods to start times. A start time of $\emptyset$ means that the method will not be executed. A *feasible* schedule obeys both mutex and precedence constraints (*i.e.* an agent may not execute more than one method at a time and all NLEs are obeyed). The objective is to create a feasible schedule that maximizes $\omega(G)$.

5.1.2 Distributed Constraint Optimization

A DCOP can be formalized in set-theoretic notation as a tuple $(A, V, \mathscr{D}, F, \alpha, \sigma)$, where

- A is a set of agents;
- V is a set of variables, $\{v_1, v_2, \ldots, v_{|V|}\}$;
- $\mathscr{D}$ is a set of domains, $\{D_1, D_2, \ldots, D_{|V|}\}$, where each $D \in \mathscr{D}$ is a finite set containing the feasible values for its associated variable;
- F is a set of $|V|^2$ cost functions, one for each pair of variables, such that $f_{ij} : D_i \times D_j \to \mathbb{N}_0 \cup \{\infty\}$. Each cost function maps every possible variable assignment to its associated cost, for all pairs of variables and for all possible assignments. These functions can also be thought of as constraints;
- α is function $\alpha : V \to A$ mapping variables to their associated agent. $\alpha(v_i) \mapsto a_j$ implies that it is agent a_j's responsibility to assign the value of variable v_i. Note that it is not necessarily true that α is either an injection or surjection; and
- σ is a function $\sigma : F \to \mathbb{N}_0$ that aggregates the individual costs. This is usually accomplished through summation.

The objective of a DCOP is to have each agent assign values to its associated variables in order to minimize $\sigma(F)$ for a given assignment of the variables[1].

5.1.3 Analysis

From Theorem 1 we see that, not only is determining an optimal feasible schedule $\mathcal{NP}$-Hard, but the satisfaction problem of deciding which methods to execute and which not to execute is also $\mathcal{NP}$-Hard.

Theorem 1. *The optimization problem of finding a feasible schedule with maximum $\omega(G)$ is $\mathcal{NP}$-Hard.*

Proof. We will prove this with a reduction from k-coloring of graphs (where $k \geq 3$). Given a graph $G = (V, E)$, create one agent in A for each $v \in V$. Create k methods associated with each agent, one for each of the k colors. Therefore, $(\forall a \in A : |\mu^{-1}(a)| = k)$, and $|\mathscr{M}| = k|V|$. Define each method as $\langle 0, 0, 0 \rangle$, meaning that each method can either be executed at time 0 or it will not be executed at all. The mutex constraints will ensure that each agent can only have *at most* one method executed. Create an enables NLEs (*i.e.* precedence constraint) between all pairs of methods associated with like colors on adjacent vertices. Since each method has only one feasible start time, these NLEs ensure that two adjacent vertices cannot have two methods of the same color scheduled to execute. Creating this mapping will require $\binom{n}{2}2k = O(n^2)$ operations, which is in $\mathcal{P}$. □

[1] This definition was adapted from [19] and has been modified for the sake of brevity, clarity, and applicability.

5.2 Technical Approach

The basis of our approach is to map a given C_TÆMS representation of the coordinators problem to an equivalent DCOP whose solution leads to an optimal schedule, as in Figure 1.1. The technical challenge lies in ensuring that the resulting DCOP's solution leads to an optimal schedule. The following section formalizes the approach.

5.2.1 Mapping C_TÆMS to a DCOP

For each agent in the C_TÆMS instance, $a \in A$, create an associated DCOP agent. For each method, $M \in \mathscr{M}$, create an associated variable $v_M \in V$. Therefore, the α function of the DCOP can be defined as an analogue of the μ function of C_TÆMS. The domains of the variables will contain all possible start times of the method (including an option for the method to forgo execution). In order to encode the mutex constraints,

$$(\forall a \in A : (\forall (M_i, M_j) \in \mu^{-1}(n) \times \mu^{-1}(n) : f_{ij}(x \in D_i, y \in D_j) \mapsto \infty \\ \text{when } (x < y < x + d_{M_i}) \vee \big(y < x < y + d_{M_j}\big))).$$

In other words, for all agents $a \in A$ find all pairs of methods M_i and M_j that share agent a and create a hard DCOP constraint (*i.e.* of infinite cost) for all pairs of equal domain values for the associated variables. This will ensure that an agent may not execute multiple methods at once. NLEs (*i.e.* precedence constraints) are encoded similarly:

$$\begin{array}{rcl} (\forall e \in E : (\forall X, Y & | & e(X,Y) : \\ (\forall M_i, M_j & | & \varphi^+(M_i, X) \wedge M_i \in \mathscr{M} \wedge \varphi^+(M_j, Y) \wedge M_j \in \mathscr{M} \\ & : & f_{ij}(x \in D_i, y \in D_j) \mapsto \infty \\ & & \text{when } y < x + d_{M_i}))). \end{array}$$

For all *enables* NLEs, find all pairs of methods that are in the subtree rooted by the endpoints of the NLE and add a hard DCOP constraint for their associated variables when the NLE is violated.

Finally, add one unary soft constraint for all methods' variables as follows:

$$(\forall M_i \in \mathscr{M} : f_i(\emptyset) \mapsto \omega\,(M_i))\,.$$

If a method is *not* scheduled to execute, its unary constraint will have a cost equal to the quality that the method *would have* contributed to G had it been executed. This mapping will produce a DCOP with $|M|$ variables and worst-case $O(|M|^2)$ constraints.

An example of this mapping on a C_TÆMS instance consisting of three tasks and four methods is presented in Figure 5.1.

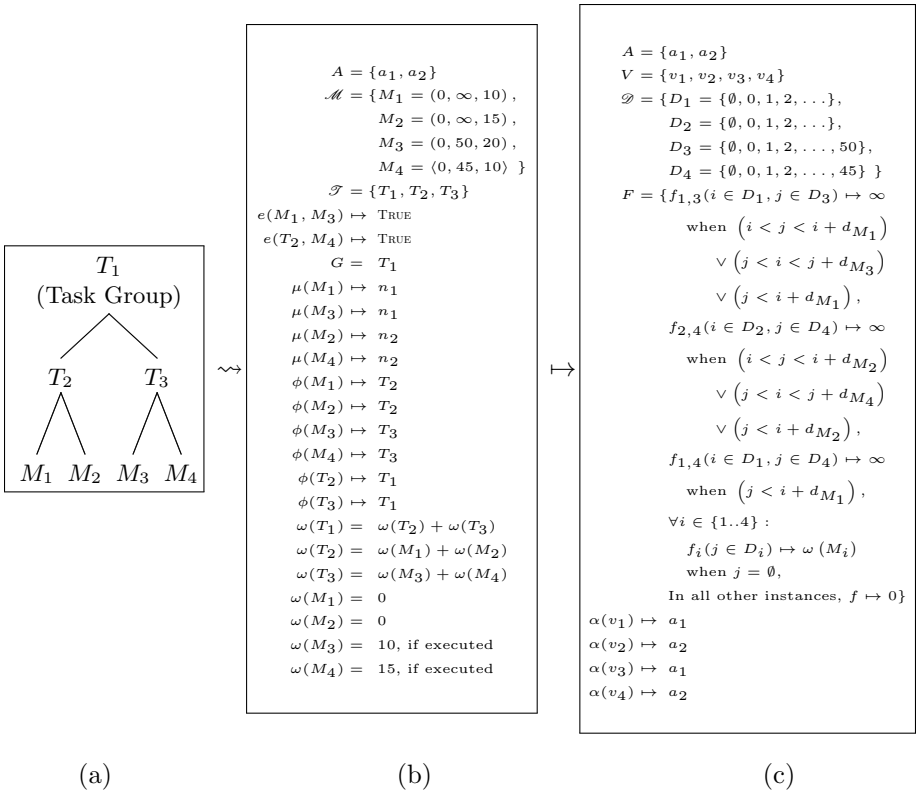


Figure 5.1: An example task hierarchy, (a), with associated representation in C_TÆMS, (b), along with our mapping to a DCOP, (c).

5.2.2 Naïve Domain Bounding

Nothing thus far in the mapping precludes domains from being infinite. The example presented in Figure 5.1 has two such domains: D_1 and D_2. This poses a problem, since the definition of a DCOP forbids infinite domains (see §5.1.2). From a practical standpoint, this is also a problem because most DCOP solution techniques have computational complexity

$$O\left(\left(\frac{\sum_{D\in\mathscr{D}}|D|}{|\mathscr{D}|}\right)^{|V|}\right).$$

Since the number of variables, $|V|$, is generally inflexible, not only do we need to make the domains finite but we ideally need to make them as small as possible while ensuring that the solution space of the DCOP still contains the optimal solution.

Without using any constraint propagation, domain filtering, pruning, or consistency techniques [5], it is still possible to create a finite (although not necessarily tight) upper bound on the start times of the methods. Let us consider a C_TÆMS instance in which all methods have an earliest start time of zero. In this case, assuming all of the methods will be executed, the optimal start time of a method cannot be greater than the sum of the expected durations of all of the other methods. In the general case of heterogeneous earliest start times, we can define an upper bound on the start time of a method M as the maximum finite earliest start time in $\mathscr{M}$ plus the duration of all other methods:

$$\begin{cases} u_M - d_M & \text{if } u_M \in \mathbb{N}_0, \\ \left(\max\limits_{\{M'\in\mathscr{M}:M'\neq M\}} l_{M'}\right) + \left(\sum\limits_{M'\in\mathscr{M}-\{M\}} d_{M'}\right) & \text{if } u_M = \infty. \end{cases}$$

Theorem 2. *The proposed method of naïve domain bounding will not prune all optimal solutions.*

Proof. The longest possible C_TÆMS schedule duration will occur when all of the methods are chosen to execute. A schedule will have a maximal completion time when there exists an enables NLE chain over all of the methods, rooted at the method with maximum earliest start time. There might be an infinite number of optimal solutions to a C_TÆMS instance. All optimal schedules, however, will have the same quality as the schedule in which all of the methods are executed, in order, starting with the method with maximum earliest start time. Let us assume, on the contrary, that the optimal schedule *does not* execute all methods, yet the naïve bounding *does* prune the optimal solution. The only way this can be the case is if some method's optimal execution time were *greater* than the maximum start time plus the longest possible duration, which would

mean such a schedule would be longer than the longest possible duration: a contradiction. □

Take M_1 from Figure 5.1 as an example of a method with an infinite deadline. Using this naïve bounding technique, the maximum start time of M_1 could be set to

$$\max\{u_{M_3} = 50, u_{M_4} = 45\} + \sum\{d_{M_2} = 15, d_{M_3} = 20, d_{M_4} = 10\} = 95,$$

thus making M_1's naïve domain $D_1 = \{\emptyset, 0, 1, 2, \ldots, 95\}$.

5.2.3 Bound Propagation

Although the nature of the Coordinators problem implies that a child's bounds are inherited from (and therefore cannot be looser than) its parent's, the C_TÆMS modeling language neither requires nor enforces this. Assuming each child knows the bounds of its parent, bounds can be propagated down the tree from the root to improve upon the naïve bounding. Formally, the recursion is as follows:

$$l'_X = \begin{cases} l_X & \text{if } X = G \vee l_X \geq l_{\phi(X)}, \\ l_{\phi(X)} & \text{Otherwise.} \end{cases}$$

$$u'_X = \begin{cases} u_X & \text{if } X = G \vee u_X \leq u_{\phi(X)}, \\ u_{\phi(X)} & \text{Otherwise.} \end{cases}$$

A distributed method for implementing this procedure (only requiring local knowledge) is given in Algorithms 4 and 5. To calculate the bounds for the methods of agent a, the algorithm would be invoked as

$$\text{RECURSE-EXECUTION-BOUNDS}(a, G, C, 0, \infty).$$

5.2.4 Constraint Propagation

A binary constraint, f_{ij}, is *arc consistent* if

$$(\forall d_i \in D_i : (\exists d_j \in D_j : f_{ij}(d_i, d_j) \neq \infty)).$$

A DCOP is said to be arc consistent if all $f \in F$ are arc consistent [5]. We use forward constraint propagation down the NLE chains to prune the domains, ensuring arc consistency of the DCOP. Intuitively, if method M_1 enables M_2, then the earliest start time of M_2 cannot possibly be less than the earliest start time of M_1 plus the expected duration of M_1. An example of this procedure

Algorithm 4 RECURSE-EXECUTION-BOUNDS$(a, t, C, \ell, \upsilon)$

Require: a is the agent from whose perspective we will create. the bounds, t is the task rooting the tree whose bounds we will create, C is a C_TÆMS problem instance, ℓ is a lower bound on the bounds of t, and υ is an upper bound on the bounds of t.

Ensure: $\beta_a : T \rightarrow (\mathbb{N}_0 \cup \{\infty\}) \times (\mathbb{N}_0 \cup \{\infty\})$ is a function mapping all tasks in the subtree rooted at t to lower and upper bounds on their start times. The subscript a exists to emphasize the point that each agent has its own β function; the mapping of each β function is contingent upon the extent of knowledge the agent has been given within the problem instance.

1: $l \leftarrow \ell$
2: $u \leftarrow \upsilon$
3: **if** VISIBLE-TO?(C, t, a) **then**
4: $e \leftarrow$EARLIEST-START-TIME(C, t)
5: $d \leftarrow$DEADLINE(C, t)
6: **if** $e \neq \infty \wedge e > l$ **then**
7: $l \leftarrow e$
8: **end if**
9: **if** $d \neq -\infty \wedge d < u$ **then**
10: $u \leftarrow d$
11: **end if**
12: **end if**
13: $\beta_a(t) \mapsto (l, u)$
14: **for all** $s \in$SUBTASKS(C, t) **do**
15: **if** IS-METHOD?(C, s) **then**
16: This means s is a method (*i.e.* a special type of task)
17: **if** VISIBLE-TO?(C, s, a) **then**
18: $l_m \leftarrow l$
19: $u_m \leftarrow u$
20: $e \leftarrow$EARLIEST-START-TIME(C, s)
21: $d \leftarrow$DEADLINE(C, s)
22: $a \leftarrow$EXPECTED-DURATION(C, s)
23: **if** $e \neq \infty \wedge e > l_m$ **then**
24: $l_m \leftarrow e$
25: **end if**
26: **if** $d \neq -\infty \wedge d - a < u_m$ **then**
27: $u_m \leftarrow d$
28: **end if**
29: $\beta_a(s) \mapsto (l_m, u_m)$
30: **end if**
31: **else**
32: This means s is a regular task.
33: RECURSE-EXECUTION-BOUNDS(a, s, C, l, u)
34: **end if**
35: **end for**

Algorithm 5 EXECUTION-BOUNDS(a, C)

Require: a is the agent from whose perspective we will create the bounds and C is a C_TÆMS problem instance.

Ensure: $\beta : T \rightarrow (\mathbb{N}_0 \cup \{\infty\}) \times (\mathbb{N}_0 \cup \{\infty\})$ is a function mapping all tasks in C that are visible to a to to lower and upper bounds on their start times.

1: RECURSE-EXECUTION-BOUNDS(a,TASK-GROUP(C), $C, 0, \infty$)

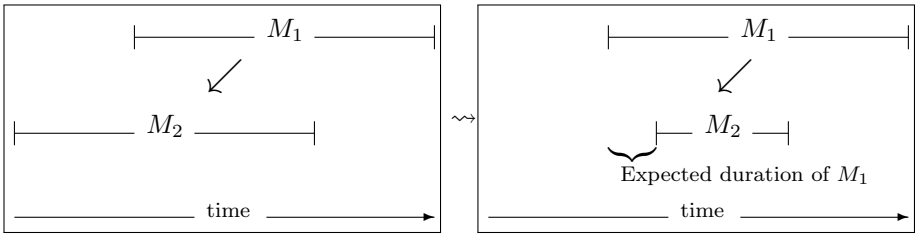


Figure 5.2: Feasible start times for methods M_1 and M_2 both before, (a), and after, (b), constraint propagation. If the C_TÆMS model declares M_1 as enabling M_2, (a), then constraint propagation will increase the lower bound on the start time of M_2 to the earliest start time of M_1 plus the expected duration of M_1, (b).

is given in Figure 5.2. A centralized algorithm for performing constraint propagation is given in Algorithm 6. An equivalent distributed version is given in Algorithm 8. Note that this algorithm makes use of Algorithm 7, BROADCAST-BOUNDS(C, β_a, a), which has the following postcondition: agent a's bounds will be broadcast to all other agents that share an NLE with the method/task associated with the given bound. Algorithm 8 works by having agents continually broadcast their current start time bounds for their methods; if they receive a bound that violates arc consistency, they increase the lower bound on their method's start time until the constraint is arc consistent and re-broadcast the new bounds. Since the lower bounds monotonically increase and are bounded above, the algorithm must terminate. An analysis of the messaging overhead of this algorithm is presented in §5.3.

5.3 Results

Using the Coordinators program scenario generator, we randomly-generated a set of 100 C_TÆMS instances, each with 4 agents, 3-to-4 windows[2], 1-to-3 agents

[2]"Windows" are tasks whose parent is the task group (*i.e.*, they are tasks at the second layer from the root in the HTN).

Algorithm 6 CENTRALIZED-NLE-TIGHTENING(C, B)

Require: C is a C_TÆMS problem instance and B is a set of bound functions $\{\beta_{a_1}, \beta_{a_2}, \beta_{a_3} \ldots\}$, one for each agent (*e.g.* as calculated by the EXECUTION-BOUNDS algorithm).

repeat
  $q \leftarrow$FALSE
  for all $n \in$ENABLES-NLES(C) **do**
    for all $(s,t) \in$SOURCES$(n,C) \times$TARGETS(n,C) **do**
      if VISIBLE-TO?$(C,t,$GET-AGENT$(C,s)) \vee$VISIBLE-TO?$(C,s,$GET-AGENT$(C,t))$ **then**
        $a_s \leftarrow$GET-AGENT(C,s)
        $a_t \leftarrow$GET-AGENT(C,t)
        $(l_s, u_s) \leftarrow \beta_{a_s}(s)$
        $(l_t, u_t) \leftarrow \beta_{a_t}(t)$
        $\delta \leftarrow l_s +$EXPECTED-DURATION$(C,s) - l_t$
        if $\delta > 0 \wedge l_t + \delta \leq u_t$ **then**
          $\beta_{a_s}(t) \mapsto (l_t + \delta, u_t)$
          $\beta_{a_t}(t) \mapsto (l_t + \delta, u_t)$
          $q \leftarrow$TRUE
        end if
      end if
    end for
  end for
until $\neg q$

Algorithm 7 BROADCAST-BOUNDS(C, β_a, a)

Require: C is a C_TÆMS problem instance and β_a is the associated bound function for the agent, a, that is running this instance of the algorithm. Broadcasts are tuples of the form $\langle a, r \in \{\text{SOURCE, TARGET}\}, s, t, (l, u)\rangle$, where f is the agent sending the broadcast, r is the role of the sender in the NLE, s is the source method in the NLE, t is the target method in the NLE, and (l, u) are the bounds for the method a is controlling (as defined by its role, r).

Ensure: Agent a's bounds will be broadcast to all other agents that share an NLE with the method/task associated with the given bound.

1: **for all** $(n, s) \in$SOURCE-OF-ENABLES-NLES(C, a) **do**
2: For each NLE of which a is a source (*i.e.* an enabler)...
3: **for all** $t \in$TARGETS(n, C) **do**
4: SEND-MESSAGE$((a, \text{SOURCE}, s, t, \beta_a(s))$,GET-AGENT$(t))$
5: **end for**
6: **end for**
7: **for all** $(n, t) \in$TARGET-OF-ENABLES-NLES(C, a) **do**
8: For each NLE of which a is a target (*i.e.* the enabled)...
9: **for all** $s \in$SOURCES(n, C) **do**
10: SEND-MESSAGE$((a, \text{TARGET}, s, t, \beta_a(s))$,GET-AGENT$(s))$
11: **end for**
12: **end for**

Algorithm 8 DISTRIBUTED-CONSTRAINT-PROPAGATION(C, β_a, a, Q)

Require: C is a C_TÆMS problem instance and β_a is the associated bound function for the agent, a, that is running this instance of the algorithm. Q is a queue that is continuously updated with incoming broadcasts.

1: BROADCAST-BOUNDS(C, β, a)
2: **while** $Q \neq \emptyset$ **do**
3: $(f, r, s, t, (l, u)) \leftarrow$POP$(Q)$
4: **if** $r =$SOURCE **then**
5: $\beta_a(s) \mapsto (l, u)$
6: $l_s \leftarrow l$
7: $u_s \leftarrow u$
8: $(l_t, u_t) \leftarrow \beta_a(t)$
9: **else**
10: $\beta_a(t) \mapsto (l, u)$
11: $l_t \leftarrow l$
12: $u_t \leftarrow u$
13: $(l_s, u_s) \leftarrow \beta_a(s)$
14: **end if**
15: $\delta \leftarrow l_s +$EXPECTED-DURATION$(C, s) - l_t$
16: **if** $\delta > 0 \wedge l_t + \delta \leq u_t$ **then**
17: $\beta_a(t) \mapsto (l_t + \delta, u_t)$
18: $\beta_a(t) \mapsto (l_t + \delta, u_t)$
19: BROADCAST-BOUNDS(C, β_a, a)
20: **end if**
21: **end while**

Solubility	Avg. Domain Size Reduction	Avg. Final Domain Size	State Space Reduction	Final State Space Size
Solved	8.02%	35.29	97%	2.34×10^{71}
Unsolved	7.61%	35.24	94%	1.47×10^{77}

Table 5.1: Efficiency of Algorithm 8 at reducing average domain size and state space size, in terms of solubility.

per window, and 1-to-3 NLE chains. Note that the scenario generator does not ensure that a feasible schedule exists for its resulting C_TÆMS instances. Even with the small simulation prameters and using all of our domain reduction techniques, the average state space size of these problems was astronomical: on the order of 10^{77}. Therefore, some of the problems inevitably require inordinate amounts of computation time before converging on the optimal solution. There seems to be a phase transition in the problems, such that some are soluble within the first couple thousand cycles of the DCOP algorithm, while the rest keep searching for an optimal solution into the millions of cycles. In terms of computation time, this equates to several orders of magnitude difference: seconds versus days. This necessitated a threshold—that we've set to 10,000 DCOP cycles—above which a C_TÆMS instance is simply declared "insoluble." We have not found a single C_TÆMS instance that has been soluble in greater than 5000 cycles (given a reasonable amount of computation time).

We used the DCOP algorithm Adopt [41] to solve the resulting DCOPs. Of the 100 random problems, none were soluble by the naïve domain bounding approach. Applying bound propagation (Algorithm 4) to the naïve bounding resulted in 2% of the problems becoming soluble. Applying all of our methods resulted in 26% solubility. Using an upper one-sided paired t-test, we can say with 95% certainty that Algorithm 8 made an average domain size reduction of 7.62% over the domains produced from Algorithm 4. If we look at this reduction in terms of state space size, however, it becomes much more significant: an average 94% decrease in state space size. Table 5.1 presents the state space reduction efficiency of our constraint propagation technique in terms of problem solubility. Since constraint propagation was fairly consistent in the percentage of state space reduced between those problems that were soluble and those that were insoluble, this suggests that the 74% of the problems that remained insoluble were due to the large state space size inherent in their structure. For example, the insoluble problems' state spaces were, on average, *one million times* as large as those that were soluble.

We also conducted a battery of tests over C_TÆMS problems of differing complexity (by varying the number of windows and NLE chains). The number of windows is correlated to the number of variables in the resulting DCOP, while

# Windows	# NLE Chains	% Soluble Naïve	% Soluble CP	Avg. # Cycles Naïve	Avg. # Cycles CP	Avg. # Messages Naïve	Avg. # Messages CP
3	0	0	40.00	-	143.87	-	2569.25
3	2	0	36.67	-	143.40	-	3114.14
3	4	0	46.67	-	220.73	-	3854.2
4	0	0	33.33	-	83.89	-	1758.5
4	2	0	11.76	-	66.18	-	2783
4	4	0	33.33	-	108.72	-	3887
5	0	0	60.00	-	122.1	-	1364.5
5	2	0	60.00	-	242.4	-	2634
5	4	0	50.00	-	248.8	-	5748.67
* 6	3	0	41.67	-	196.42	-	4025.08

Table 5.2: Solubility statistics for different complexities of C_TÆMS instances. All simulations were conducted with four agents. None of the problems bounded using the naïve method were soluble. * This is the default configuration for the C_TÆMS scenario generator.

the number of NLEs is correlated to the number of constraints in the resulting DCOP. These data are presented in Table 5.2. Notice that problems bounded naïvely were never soluble. Over the most complex problems with 6 windows and 3 NLEs chains, Algorithm 8 required an average of 144.94 messages (with a standard deviation of 16.13). This was negligible in comparison to the number of messages required to arrive at an optimal solution.

Chapter 6

> “This one’s from the book!”
> — Pál Erdős

Conclusions

This thesis provides three contributions. First of all, MATES: the Macro Agent Transport Event-based Simulator was presented. MATES is the first simulator of its kind and provided the testbed for our subsequent investigations. Secondly, we developed a bottom-up approach to distributing global state information around a dynamic peer-to-peer network. Finally, we used the model generated by the belief of the global state to drive a top-down automated mechanism by which agents can develop globally-optimal low-level schedules.

6.1 Evaluating Multi-Agent Systems on Dynamic Peer-to-Peer Networks

MATES’ “continued execution” architecture was proposed, providing a means for simulators to increase transparency and appear akin to development for real-world agent systems. The validity of MATES’ models was confirmed through correlation between simulated experiments and live, empirical data. Dynamic, peer-to-peer networks and agency are both heavily researched areas of computer science. As these types of networks become more prevalent, efficient distributed algorithms will be required; the need for further research in this area has already been established [39]. Therefore, a system for testing and comparing these algorithms before deployment is a necessity. We believe MATES, or a system like it, is a step toward achieving this.

6.1.1 Estimating Global State

We have presented an approach to service discovery in dynamic network environments. The current literature on SOAs does not yet address these issues directly, nor has there been any substantial theoretical or empirical studies of approaches to solve the discovery problem for SOAs in environments as MANETs. In this context, the location and capabilities of services, agents, and hosts are all part of a global state which can only be partially observed by each agent, host or service provider in the network. The technical approach

proposed herein uses mobile agents and exploits the combinatorial properties of random walks to create a set of service discovery agents that maintain overall state for all nodes on the network.

The principle contributions of this facet of the thesis includes the development of a mathematical formulation of the problem of service discovery by mobile agents in a dynamic network and a set of empirical studies that validate the formulation using MATES. A method for distributing global state information using service discovery agents has been proposed. Possible applications for the service discovery mechanism and its associated model include learning service availability thresholds, "network triage," time-critical reasoning, service composition, and agent population size optimization.

The results also indicate that this proactive approach can be used to maintain accurate state information across a dynamic network while having a limited effect on network messaging. This work represents an important example of how mobile agents can be practically adapted to the constraints posed by real network environments. In addition, this work can provide a basis for enabling multi-agent planning to sense and react to vital network-level events in order to improve plan execution and survivability.

6.1.2 Multi-Agent Coordination

We have presented a mapping from a subset of the C_TÆMS modeling language to an equivalent DCOP. We have shown that the resulting DCOP is soluble using existing techniques, and whose solution is guaranteed to lead to an optimal schedule. We have empirically validated our approach, using various existing techniques from the constraint processing literature, indicating that these problems are in fact soluble using our method.

We hope to and are optimistic in extending our mapping to subsume a larger subset of C_TÆMS, including more types of NLEs and QAFs. There are also various heuristic techniques in the literature, such as variable ordering [14], that can be applied to the mapping while retaining formal guarantees on solution quality. If the resulting schedule need not be optimal (*i.e.* feasibility is sufficient), approximation techniques for DCOPs also exist.

With the groundwork laid in solving distributed multi-agent coordination problems with distributed constraint optimization, we have many extensions in which to investigate. For example, we can exploit the HTN structure of C_TÆMS by solving the problem hierarchically; first choose bounds on the execution time of a task and then recursively choose tighter bounds for its children. When a task has multiple children the problem would then be in intelligently partitioning and allotting the parent's domain to its children. We are also interested in the problem of re-planning, if changes to the original C_TÆMS problem occur after the optimal schedule is already found. There are certain classes of perturbations to the original problem from which we can re-plan in a bounded

amount of time given the optimal schedule from the original problem.

6.1.3 Coordinating Agents in Stochastic Peer-to-Peer Environments

This thesis has laid the groundwork for bootstrapping a distributed system—in which all knowledge is local and communication is not ensured—to the point where multilateral, globally-optimal decisions can be made. As these systems become more prevalent and complex, it is imperative that we advance these techniques to a high level of automation. It is my hope that this thesis has both motivated interest in and progressed our understanding of these problems.

Appendix A

> ⟪ *[C]e miracle de l'Analyse, prodige du monde des idées, objet presque amphibie entre l'Être et le Non-être, que nous appelons racine imaginaire.* ⟫
>
> —Attributed to Gottfried Wilhelm von Leibniz
>
> **Translation**: This miracle of analysis, this marvel of the world of ideas, an almost amphibian object between Being and Non-being that we call the imaginary number.

Notation and Nomenclature

The following sections define the notation and nomenclature used throughout this thesis.

Notation

R^+	The transitive closure of a binary relation R.
$E_{x,y}$	The weight of the edge between vertices x and y in the edge set E of a graph.
$X \times Y$	The Cartesian product (or direct product) of two sets: $\{(x, y) \mid x \in X \wedge y \in Y\}$.
$\lvert X \rvert$	Cardinality, when X is a set; the absolute value of X otherwise.
$\vec{X}$	A vector.
$\vec{X}_i$	The i^{th} element of vector $\vec{X}$.

$\vec{\pi}$	A primary right eigenvector; $\vec{\pi}$ corresponds to the eigenvalue, λ_1, of matrix X such that $\vec{\pi}X = \lambda_1\vec{\pi}$.
$P(X)$	The probability of an event X.
$\langle X \rangle$	The expected value of a random variable X.
$\hat{X}$	An estimator of parameter X.
(a, b, c)	A tuple containing elements a, b, and c.
$\max_{x \in X} f(x)$	The maximum value of $f(x)$ over all elements in the set X.
$\lfloor x \rfloor$	The floor function; $\lfloor x \rfloor = (\max_{n \in \mathbb{Z}}(n) : n < x)$.
$f : A \to B$	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 $f : A \to B$, 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); $p \implies q$ is equivalent to $\neg p \vee q$.
$\iff$	If and only if; $p \iff q$ is equivalent to $(p \implies q) \wedge (q \implies p)$.
$O(g(x))$	Given two functions $f(x)$ and $g(x)$ defined on some subset of $\mathbb{R}$, we say $f(x)$ is $O(g(x))$ as $x \to \infty$ if and only if $\exists x_0 \exists y > 0$ such that $\lvert f(x) \rvert \leq y \lvert g(x) \rvert$ 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 $f : \{0,1\}^* \to \{0,1\}^*$ such that for all $x \in \{0,1\}^*$, $x \in L_1 \iff f(x) \in L_2$.

Nomenclature

$\mathbb{N}_0$	The natural numbers, including zero (*i.e.*, non-negative integers).
$\mathbb{Z}$	The integers.
$\mathbb{R}$	The real numbers.
$\mathbb{B}$	A boolean domain: $\{\textsc{True}, \textsc{False}\}$.
$\mathcal{NP}$	The class of languages (*i.e.*, problems) can be verified by a polynomial-time algorithm. A language L belongs to $\mathcal{NP}$ if and only if there exist a two-input polynomial-time algorithm A and a constant c such that $L = \{x \in \{0,1\}^* : \text{there exists a certificate } y \text{ with } \lvert y\rvert = O\left(\lvert x\rvert^c\right) \text{ such that } A(x,y) = 1\}$.
$\mathcal{NP}$-Hard	Non-deterministic Polynomial-time Hard. A class of languages (*i.e.*, problems) that contains all languages L such that for all $L' \in \mathcal{NP},\ L' \leq_p L$.
A	A set of agents.
H	A set of hosts.
S	A set of services.
δ	A time interval represented as a non-negative real number.
ν	The probability that an agent will be at a host.
η	The number of instances of a service within a network.
ℓ	The average amount of time required for an agent to migrate between two adjacent hosts.

τ	The maximum allowable amount of time since a service discovery agent last saw an instance of a service.
N	A local state description, represented as the tuple $(t, \nu, \eta, \lvert H\rvert, \lvert A\rvert, \ell)$.
$\mathscr{N}$	The set of all possible state descriptions.
$\psi : \mathscr{N} \times S \times H \to [0, 1]$	A function returning the probability that at least one agent $a \in A$ with knowledge of a service $s \in S$ will visit a specific host $h \in H$ in a time interval t.
V	The set of variables in a DCOP.
$\mathscr{D}$	The set of domains for the variables of a DCOP; there is one $D \in \mathscr{D}$ for each $v \in V$.
$\emptyset$	The empty set, or, when an element of a domain $D \in \mathscr{D}$, the choice not to execute a method.
F	A set of cost functions for a DCOP; each f_{ij} in F is a function $f_{ij} : D_i \times D_j \to \mathbb{N}_0 \cup \{\infty\}$.
$\alpha : V \to A$	A function mapping variables in a DCOP to their associated agent.
$\sigma : F \to \mathbb{N}_0$	A function that aggregates the costs in F for a given assignment of the variables of a DCOP.
J	The adjacency matrix of a network. J is a square, $\lvert H\rvert$ by $\lvert H\rvert$ matrix such that J_{ij} equals the link quality from host H_i to host H_j. Note that J might not be symmetric if the radio ranges of the hosts are not homogeneous.

The definitions for $O(g(x))$, $L_1 \leq_p L_2$, $\mathcal{NP}$, and $\mathcal{NP}$-HARD were adapted from [16].

Bibliography

[1] Syed Ali, Sven Koenig, and Milind Tambe. Preprocessing techniques for accelerating the dcop algorithm adopt. In *AAMAS '05: Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems*, pages 1041–1048, New York, NY, USA, 2005. ACM Press.

[2] F. Bai, N. Sadagopan, and A. Helmy. Important: a framework to systematically analyze the impact of mobility on performance of routing protocols for adhoc networks. In *Twenty-Second Annual Joint Conference of the IEEE Computer and Communications Societies*, pages 825–835, 2003.

[3] Christopher L. Barrett, Madhav V. Marathe, D. Charles Engelhart, and Anand Sivasubramaniam. Approximate connectivity graph generation in mobile ad hoc radio networks. In *Proceedings of the* 36^{th} *Annual Simulation Symposium*, page 81. IEEE Computer Society, 2003.

[4] Lali Barrière, Paola Flocchini, Pierre Fraigniaud, and Nicola Santoro. Capture of an intruder by mobile agents. In *Proceedings of the fourteenth annual ACM symposium on Parallel algorithms and architectures*, pages 200–209. ACM Press, 2002.

[5] Roman Barták. Theory and practice of constraint propagation. In J. Figwer, editor, *Proceedings of the* 3^{rd} *Workshop on Constraint Programming in Decision Control*, Poland, June 2001.

[6] BBN Technologies. Cougaar architecture document. http://docs.cougaar.org, February 2003.

[7] C. Bessiere. Arc-consistency and arc-consistency again. *Artificial Intelligence*, 65:179–190, 1994.

[8] C. Bessiere, E. Freuder, and J-Ch. Regin. Using constraint metaknowledge to reduce arc consistency computation. *Artificial Intelligence*, 107:125–148, 1999.

[9] Brandon Bloom, Christopher J. Dugan, Tedd Gimber, Bernard Goren, Moshe Kam, Joseph B. Kopena, Robert N. Lass, Israel Mayk, Spiros Mancoridis, Pragnesh Jay Modi, William M. Mongan, William C. Regli, Randy Reitmeyer, Jeff K. Salvage, Evan A. Sultanik, and Todd Urness. Agent systems reference model. Technical Report Draft 7183, Intelligent Agents Product Sub-Team, Networking Integrated Product Team, Command and Control Directorate, Department of the Army, February 2006.

[10] M. Boddy, B. Horling, J. Phelps, R. Golman, R. Vincent, A. Long, and B. Kohout. C-taems language specification v. 1.06, 2005.

[11] Sergey Brin and Lawrence Page. The anatomy of a large-scale hypertextual Web search engine. *Computer Networks and ISDN Systems*, 30(1–7):107–117, 1998.

[12] Tracy Camp, Jeff Boleng, and Vanessa Davies. A survey of mobility models for ad hoc network research. *Wireless Communication & Mobile Computing (WCMC): Special Issue on Mobile Ad Hoc Networking: Research, Trends and Applications*, 2(5):483–502, 2002.

[13] Dipanjan Chakraborty, Yelena Yesha, and Anupam Joshi. A distributed service composition protocol for pervasive environments. In *IEEE Wireless Communications and Networking Conference*, 2004.

[14] Anton Chechetka and Katia Sycara. A decentralized variable ordering method for distributed constraint optimization. In *AAMAS '05: Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems*, pages 1307–1308, New York, NY, USA, 2005. ACM Press.

[15] Vincent Cicirello, Max Peysakhov, Gustave Anderson, Gaurav Naik, Kenneth Tsang, William Regli, and Moshe Kam. Designing dependable agent systems for mobile wireless networks. *IEEE Intelligent Systems*, 19(5):39–45, September 2004. Special Issue on Dependable Agent Systems.

[16] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms.* McGraw-Hill, second edition, 2001.

[17] Jeffrey S. Cox, Edmund H. Durfee, and Thomas Bartold. A distributed framework for solving the multiagent plan coordination problem. In *AAMAS '05: Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems*, pages 821–827, New York, NY, USA, 2005. ACM Press.

[18] P. Dasgupta. Improving peer-to-peer resource discovery using mobile agent based referrals. In *Proceedings of the* 2^{nd} *International Autonomous Agents*

and Multi-agent Systems Conference (AAMAS), Proceedings of the 2nd *Workshop on Agent Enabled P2P Computing*, pages 41–54, July 2003.

[19] John Davin and Pragnesh Jay Modi. Impact of problem centralization in distributed constraint optimization algorithms. In *Proceedings of Forth International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 1057–1063, July 2005.

[20] Keith Decker. TAEMS: A Framework for Environment Centered Analysis & Design of Coordination Mechanisms. In *Foundations of Distributed Artificial Intelligence, Chapter 16*, pages 429–448. G. O'Hare and N. Jennings (eds.), Wiley Inter-Science, January 1996.

[21] Christos Gkantsidis, Milena Mihail, and Amin Saberi. Random walks in peer-to-peer networks. In *Proceedings of the* 23rd *Annual Joint Conference of the IEEE Computer and Communications Societies*. IEEE, March 2004.

[22] Y. Hamadi. Optimal distributed arc-consistency. *Constraints*, 7:367–385, 2002.

[23] P. Jacquet, P. Muhlethaler, T. Clausen, A. Laouiti, A. Qayyum, and L. Viennot. Optimized link state routing protocol for ad hoc networks. In *IEEE INMIC 01: Technology for the* 21st *Century.*, pages 62–68, 2001.

[24] David B. Johnson, David A. Maltz, and Josh Broch. DSR: The dynamic source routing protocol for multihop wireless ad hoc networks. *Ad Hoc Networking*, pages 139–172, 2001.

[25] M. L. Kahn and C. D. T. Cicalese. The CoABS grid. In *Innovative Concepts of Agent-Based Systems:* 1st *International Workshop on Radical Agent Concepts*, 2002.

[26] Lefteris M. Kirousis, Evangelos Kranakis, Danny Krizanc, and Yannis C. Stamatiou. Locating information with uncertainty in fully interconnected networks. In *International Symposium on Distributed Computing*, pages 283–296, 2000.

[27] Donald E. Knuth. *The TEXbook*. Addison-Wesley Publishing Co., Menlo Park, 1984.

[28] Joseph B. Kopena, Vincent A. Cicirello, Maxim Peysakhov, Kris Malfettone, Andrew Mroczkowski, Gaurav Naik, Evan Sultanik, Moshe Kam, and William C. Regli. Network awareness and the Philadelphia Area Urban Wireless Network Testbed. In *AAAI Spring Symposia on AI in Homeland Security*, 2005.

[29] Joseph B. Kopena, Gaurav Naik, Maxim D. Peysakhov, Evan A. Sultanik, and William C. Regli. Service-based computing for agents on disruption and and delay prone networks. In *Proceedings of the Fourth International Joint Conference on Autonomous Agents & Multi-Agent Systems*, July 2005.

[30] Ulaş C. Kozat and Leandros Tassiulas. Service discovery in mobile ad hoc networks: An overall perspective on architectural choices and network layer support issues. *Ad Hoc Networks*, 2:23–44, 2004.

[31] Leslie Lamport. *LaTeX: A Document Preparation System*. Addison-Wesley Publishing Co., Menlo Park, 1986.

[32] B. Langley, M. Paolucci, and K. Sycara. Discovery of infrastructure in multi-agent systems. In *Agents 2001 Workshop on Infrastructure for Agents, MAS, and Scalable MAS*, 2001.

[33] R. P. Lentini, G. P. Rao, J. N. Thies, and J. Kay. Emaa: An extendable mobile agent architecture. In *AAAI Workshop on Software Tools for Developing Agents*, July 1998.

[34] Sean Luke, Claudio Cioffi-Revilla, Liviu Panait, and Keith Sullivan. MASON: A new multi-agent simulation toolkit. In *Proceedings of the 2004 SwarmFest Workshop*, 2004.

[35] A.K. Mackworth. Consistency in networks of relations. *Artificial Intelligence*, 8(1):99–118, 1977.

[36] Rajiv T. Maheswaran, Milind Tambe, Emma Bowring, Jonathan P. Pearce, and Pradeep Varakantham. Taking DCOP to the real world: Efficient complete solutions for distributed multi-event scheduling. In *Proc. of the Third International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 310–317, 2004.

[37] Roger Mailler and Victor Lesser. Solving distributed constraint optimization problems using cooperative mediation. In *AAMAS '04: Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 438–445, Washington, DC, USA, 2004. IEEE Computer Society.

[38] Shivanajay Marwaha, Chen Khong Tham, and Dipti Srinivasan. Mobile agents based routing protocol for mobile ad hoc networks. In *Proceedings of IEEE International Conference on Networks*, pages 27–30, August 2002.

[39] Nikos Migas, William J. Buchanan, and Kevin A. McArtney. Mobile agents for routing, topology discovery, and automatic network reconfiguration in ad-hoc networks. In *Proceedings of the* 10^{th} *IEEE Conference and Workshop on the Engineering of Computer-Based Systems*, pages 200–206, 2003.

[40] Nelson Minar, Kwindla Hultman Kramer, and Pattie Maes. *Cooperating Mobile Agents for Dynamic Network Routing*, chapter 12. Springer-Verlag, 1999. ISBN: 3-540-65578-6.

[41] P. J. Modi, W. Shen, M. Tambe, and M. Yokoo. Adopt: Asynchronous distributed constraint optimization with quality guarantees. *Artificial Intelligence Journal*, 2005.

[42] David J. Musliner, Edmund H. Durfee, Jianhui Wu, Dmitri A. Dolgov, Robert P. Goldman, and Mark S. Boddy. Coordinated plan management using multiagent MDPs. In *Proceedings of the AAAI Spring Symposium on Distributed Plan and Schedule Management*. AAAI Press, March 2006.

[43] Khaled Nagi, Iman Elghandour, and Birgitta König-Ries. Mobile agents for locating documents in ad-hoc networks. In Claudio Sartori Gianluca Moro and Munindar P. Singh, editors, *Agents and Peer-to-Peer Computing (AP2PC 2003), Second International Workshop, Melbourne, Australia, July, 2003, Revised and Invited Papers*, volume 2872 of *Lecture Notes in Computer Science*, pages 199–205. Springer, 2004.

[44] Srini Narayanan and Sheila A. McIlraith. Simulation, verfication and automated composition of web services. In *Eleventh International Conference on World Wide Web*, pages 77–88, 2002.

[45] Massimo Paolucci, Takahiro Kawamura, Terry R. Payne, and Katia Sycara. Semantic matching of web services capabilities. In *First International Semantic Web Conference*, 2002.

[46] Massimo Paolucci, Julien Soudry, Naveen Srinivasan, and Katia Sycara. A broker for OWL-S web services. In *Semantic Web Services - 2004 AAAI Spring Symposium*, 2004.

[47] Massimo Paolucci and Katia Sycara. Autonomous semantic web services. *IEEE Internet Computing*, 7(5):34–41, 2003.

[48] Charles Perkins and Pravin Bhagwat. Highly dynamic destination-sequenced distance-vector routing (DSDV) for mobile computers. In *Proceedings of the ACM Conference on Communications Architectures, Protocols and Applications*, pages 234–244, 1994.

[49] Charles Perkins and Elizabeth M. Royer. Ad-hoc on-demand distance vector routing. In *IEEE Workshop on Mobile Computer Systems and Applications*, pages 90–100, February 1999.

[50] A. Petcu and B. Faltings. A scalable method for multiagent constraint optimization. In *Proc of International Joint Conference on Artificial Intelligence*, 2005.

[51] Max Peysakhov, Donovan Artz, William Regli, and Evan Sultanik. Network awareness for agent security in mobile ad-hoc networks. In *Proceedings of the Third International Joint Conference on Autonomous Agents and Multi Agent Systems*, pages 368–375. Association for Computing Machinery, 2004.

[52] Max Peysakhov, Vincent A. Cicirello, and William C. Regli. Ecology based decentralized agent management system. In *Proceedings of Formal Approaches to Agent-Based Systems III*, April 2004.

[53] Max Peysakhov and William Regli. Ant inspired server population management in a service based computing environment. In *Proc. of the IEEE Swarm Intelligence Symposium*, June 2005.

[54] Maxim D. Peysakhov, Robert N. Lass, and William C. Regli. Stability and control of agent ecosystems. In *Proceedings of the Fourth International Joint Conference on Autonomous Agents & Multi-Agent Systems*, July 2005.

[55] John Phelps and Jeff Rye. GPGP—a domain-independent implementation. In *Proceedings of the 2006 AAAI Spring Symposium on Distributed Plan and Schedule Management*. AAAI Press, March 2006.

[56] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Shenker. A scalable content addressable network. In *Proceedings of ACM SIGCOMM*, 2001.

[57] Romit Roy Choudhury, S. Bandyopadhyay, and Krishna Paul. A distributed mechanism for topology discovery in ad hoc wireless networks using mobile agents. In *Proceedings of the* 1^{st} *ACM international symposium on Mobile ad hoc networking & computing*, pages 145–146. IEEE Press, 2000.

[58] Kunal Shah. Performance analysis of mobile agents in wireless internet applications using simulation. Master's thesis, Lamar University, August 2003.

[59] M. Silaghi, D. Sam-Haroud, and B. Falting. Asynchronous consistency maintenance. In *Intelligent Agent Technologies*, 2001.

[60] Stephen Smith, Anthony T. Gallagher, Terry Lyle Zimmerman, Laura Barbulescu, and Zack Rubinstein. Multi-agent management of joint schedules. In *Proceedings of the 2006 AAAI Spring Symposium on Distributed Plan and Schedule Management*. AAAI Press, March 2006.

[61] Ion Stoica, Robert Morris, David Karger, M. Francs Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet

applications. In *Proceedings of the 2001 conference on applications, technologies, architectures, and protocols for computer communications*, pages 149–160. ACM Press, 2001.

[62] Evan Sultanik, Donovan Artz, Gustave Anderson, Moshe Kam, William Regli, Max Peysakhov, Jonathan Sevy, Nadya Belov, Nicholas Morizio, and Andrew Mroczkowski. Secure mobile agents on ad hoc wireless networks. In *The Fifteenth Innovative Applications of Artificial Intelligence Conference*. American Association for Artificial Intelligence, August 2003.

[63] Evan A. Sultanik, Maxim D. Peysakhov, and William C. Regli. Agent transport simulation for dynamic peer-to-peer networks. In *Proceedings of the Sixth International Workshop on Multi-Agent-Based Simulation*, July 2005.

[64] Milind Tambe. Implementing agent teams in dynamic multi-agent environments. *Applied Artificial Intelligence*, 12(2–3):189–210, March 1998.

[65] David L. Tennenhouse, Jonathan M. Smith, W. David Sincoskie, David J. Wetherall, and Gary J. Minden. A survey of active network research. *IEEE Communications Magazine*, 35(1):80–86, 1997.

[66] Hiroshi Matsuo Toshihiro Matsui and Akira Iwata. Efficient method for asynchronous distributed constraint optimization algorithm. In *Artificial Intelligence and Applications (AIA2005)*, pages 727–732, 2005.

[67] UCB/USC/LBNL/VINT. Network simulator (NS) version 2. http://www.isi.edu/nsnam/ns/, February 2003.

[68] Dan Wu, Bijan Parsia, Evren Sirin, James Hendler, and Dana Nau. Automating DAML-S web services composition using SHOP2. In *Second International Semantic Web Conference*, 2003.

[69] Jie Wu and Ivan Stojmenovic. Ad hoc networks. *IEEE Computer Magazine*, 37(2):29–31, February 2004.

[70] George Xylomenos, George C. Polyzos, Petri Mähönen, and Mika Saaranen. TCP performance issues over wireless links. *IEEE Communications Magazine*, 39(4):52–58, 2001.

[71] Oussama Kassem Zein and Yvon Kermarrec. An approach for describing/discovering services and for adapting them to the needs of users in distributed systems. In *Semantic Web Services - 2004 AAAI Spring Symposium*, 2004.

[72] H. Zimmerman. OSI reference model - the ISO model of architecture for open system interconnection. *IEEE Transactions on Communications*, 28(4):425–432, April 1980.