Inverse Reinforcement Learning and Routing Metric Discovery

Dmitry E. Shiraev

Thesis submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE
in
Computer Science and Applications

Dr. Naren Ramakrishnan, Co-Chair

Dr. Srinidhi Varadarajan, Co-Chair

Dr. Calvin J. Ribbens

August 22, 2003
Blacksburg, VA

Keywords: Inverse Reinforcement Learning, Routing, Network Metrics

Inverse Reinforcement Learning and Routing Metric Discovery

Dmitry Shiraev

(ABSTRACT)

Uncovering the metrics and procedures employed by an autonomous networking system is an important problem with applications in instrumentation, traffic engineering, and game-theoretic studies of multi-agent environments. This thesis presents a method for utilizing inverse reinforcement learning (IRL) techniques for the purpose of discovering a composite metric used by a dynamic routing algorithm on an Internet Protocol (IP) network. The network and routing algorithm are modeled as a reinforcement learning (RL) agent and a Markov decision process (MDP). The problem of routing metric discovery is then posed as a problem of recovering the reward function, given observed optimal behavior. We show that this approach is empirically suited for determining the relative contributions of factors that constitute a composite metric. Experimental results for many classes of randomly generated networks are presented.

Acknowledgments

I would like to thank my advisers Dr. Naren Ramakrishnan and Dr. Srinidhi Varadarajan. This thesis would not be possible without their guidance and insight. I am in debt to Dr. Ramakrishnan for his immense help with proofreading and structuring this thesis as well as many hours spent with me helping me out of dead ends that I managed to end up in. I must thank Dr. Varadarajan for introducing me to this topic and for continuous ideas on how to approach difficult problems that arose during my research. I would also like to greatly thank Dr. Calvin Ribbens for joining my committee on such a late notice.

I would like to give thanks to my parents Eric and Olga for their continuous encouragement. I wish to say thank you to Nicole Steward, for her never-faltering support through the most difficult days; to John Zaloudek, discussions with whom (on this and other topics) were extraordinarily valuable; and to Evan Hardin, occasional computer game with whom kept me sane.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Competing Internet Services Providers (ISPs) are secretive about the information they share with their competitors. Thus, large networks or autonomous systems that need to route packets through other autonomous systems may not know how their neighbors route information internally. If autonomous systems that belonged to different ISPs shared routing information, each autonomous system would be able to take advantage of the information in constructing its own routing tables. Thus, ISPs could offer their customers services based on that information. For example, if an ISP knows that its neighbor is minimizing delay in its routing of packets, it would be able to offer its customers services that take advantage of short delays, such as streaming voice and video services. Thus it is advantageous to the customers of this ISP if the ISP knew the details of the metric that is utilized by the autonomous systems of other competing ISPs. Because competing ISPs are reluctant to share this information, discovering it is a valuable problem.

We will present a method for an autonomous system to discover the details of the metric that is used by a neighboring autonomous system. This method utilizes inverse reinforcement learning (IRL) techniques presented in [1]. We present a method for modeling routing in an autonomous system on the Internet as a reinforcement learning (RL) problem. We will then show how the IRL techniques can be used with this model to solve the problem of discovering the details of the metric that is used by a particular autonomous system.

If ISPs utilize our method to discover the metric characteristics of the autonomous systems of their competitors, this has interesting implications from a game-theoretic perspective. Autonomous systems may develop methods that may block our techniques and the development of new methods may be necessary. This may continue indefinitely, or it may end in autonomous systems sharing their metric information. Our method is deeply involved with the type of metrics that a network may use and insight gained from its development is beneficial from a metric engineering viewpoint. Our model of the network routing problem that is utilized with our method is useful from a modeling standpoint for the purpose of applying reinforcement learning to network routing.

In this thesis we will first present background information about metrics and dynamic routing algorithms employed in practice. This will be discussed in Chapter 2, in which we will also talk about network topology and link characteristic discovery techniques and tools available

today. In Chapter 3 we will present an overview of the IRL techniques presented in [1], and how they can be applied to our problem. In Chapter 4 we will discuss how the network routing problem may be modeled as an RL problem and show how to use the model with the IRL algorithm. In Chapter 5, we will present the results for experiments performed using a simulated network topology with specific composite metrics. In Chapter 6 we will talk about future work that can be done in both IRL and network aspects of this problem.

Chapter 2

Overview of Routing Algorithms, Metrics, and Network Discovery

In this chapter we present the major routing algorithms that are employed in practice and the metrics they use. For modeling purposes, it is our goal to study routing algorithms from the perspective of maximizing (or minimizing) certain metrics. It is also important to be aware of how the routing algorithms function because this will provide us with information on how much time we have between updates to the state of the network or, how quickly optimal paths may change. This will be important for designing suitable instrumentation for network measurement.

We will first present the terminology that will be used here, then we will go over the different types of metrics that can be and are commonly deployed in a network. Then we present the set of routing algorithms commonly used on the Internet. These algorithms are divided into two types: link-state and distance-vector algorithms. We explain the distinction and present the algorithms that belong to either of these two sets. This chapter also surveys network discovery approaches and associated software.

2.1 Terminology

We will use terms from graph theory as well as terms from networking to describe physical objects being abstracted. In an attempt to make the presentation more intuitive, several terms may be used to describe the same object at different times.

- **Nodes, Edges, Graph** – a **graph** is a nonempty finite set of **vertices**, or **nodes**, and **edges**. An **edge** connects two **vertices** and may have a weight and direction.
- **Router** – a **route**r is a computer that performs the task of forwarding **packets** on a **network**. When talking about a computer network as a **graph**, a **router** corresponds to a **node** in the **graph**.
- **Link** – a **link** is a physical connection between two **routers**. When talking about a computer **network** as a **graph**, a **link** corresponds to an **edge** between two **nodes** (or

routers) in the **graph**.

- **Packet** – a **packet** is an encapsulation of information sent across the **network**. **Packets** are usually associated with a particular protocol and since the routing we are working here takes place over IP, the **packets** we work with are IP **packets**.
- **Port** – a **port** in the networking world may be used to describe a number of entities. Here it is used to enumerate a **link** from a given **router's** point of view. When a **router** sends a **packet** on a given **port**, that **packet** travels over the **link** that **port** represents. This word may be used synonymously with **interface**.
- **Routing Algorithm** – a **routing algorithm** can have several meanings, including a set of instructions that tell a **router** how to forward **packets** to their destination, or a particular instance of a program running on a **router**. When we talk about a **routing algorithm**, we will be considering an abstract entity that governs how the entire **network** behaves with respect to forwarding **packets**. A more detailed explanation can be found in the next section.
- **Network** – a **network** has a definition in the context of graph theory, as well as in the context of a computer network. A computer network is a collection of computers that can communicate by some common protocol. In our discussion, a **network** will be a set of **routers** that communicate via IP, run the same routing protocol, and utilize the same routing metric. This definition is akin to that of an autonomous system [13].

2.2 Metrics

A metric is a value assigned to a route (or path) or link that serves as a means of comparing that link with others. A routing algorithm will pick the path between it and the destination that has the smallest (or largest) value of the metric. The value assigned to a link is usually derived from certain aspects of that link, such as delay and bandwidth, but can also be arbitrarily assigned by the network administrator. The metric can also be determined by combining the values for different attributes of a link. For example, the values for bandwidth and latency may be additively combined to determine the overall value for the link. It is our goal to determine how values for different attributes for a link combine together to form the metric that is used by the network.

2.2.1 Simple Metrics

These metrics are covered in greater detail in [13].

Hop Count

The hop count metric treats every link in the network equally and favors the path that contains the least number of links. Typically a value for a link is 1, but it can be any arbitrary non-negative number, as long as that number is the same for every link in the network. It is usually

easy to determine if a network is NOT using purely a hop count metric since if the path taken by a packet goes through more links than the optimal path with respect to hop count, it becomes obvious that hop count is not used. It should also be noted that hop count is generally not used in a composite metric.

Bandwidth

The bandwidth metric is determined by the bandwidth of the link. However, exactly how it is derived from the physical bandwidth is specific to each routing algorithm. Usually a certain constant, such as 10^8 is divided by the physical bandwidth in kilobits/second to determine the metric. This is done if the algorithm is trying to minimize the value for a path, since then the bandwidth metric is lower for higher bandwidth links. The constant that is used may vary per routing algorithm and between different networks run by the same type of algorithm. This is because this method for determining the bandwidth metric creates a maximum bandwidth that can be distinguished in a network (since if the bandwidth is higher than the constant, the result will be less than 1 and will be rounded to 1). Thus, networks that contain links with higher bandwidths will need a higher constant.

If a composite metric is used, scaling the physical bandwidth to the bandwidth metric component becomes a very difficult problem with no clear solution. Most algorithms thus use ad hoc methods to scale the bandwidth.

Delay (or Latency)

The value for the delay metric represents how long it takes for a packet to arrive at the next router after being sent to a particular interface on the current router. Thus, when calculating the delay metric, dynamic aspects such as the queuing delay and latency of the router sending the packet may be included. However, the delay may also be a static value that represents the propagation delay of the link. The value for delay is usually measured in microseconds or milliseconds. If a routing algorithm uses delay as part of a composite metric, the value usually undergoes some sort of scaling. Similar to the problem with scaling the bandwidth metric, scaling latency in a meaningful way with respect to the other components of the composite metric is a difficult problem. An example of how this is done in practice is CISCO's IGRP and EIGRP algorithms, where they consider the latency in microseconds and then divide it by 10.

Load

Load is a dynamic value for a link which represents the percentage of the bandwidth of the link that is currently being used. How load is measured depends on the algorithm and hence how the numeric value is used as a metric. IGRP and EIGRP represent it as a value between 0 and 255 where 0 is a link under 0% load, and 255 is a 100% loaded link. Load is difficult for us to determine (or observe), but fortunately this metric is only used in IGRP and EIGRP [13] and with the default setting it is disabled.

Reliability

Reliability measures the chance that a given link will fail. The routing algorithm may determine this value dynamically, from observed failures, or it may be set statically per link. Since most Internet backbone links are very reliable, this metric is not very useful and is only used in IGRP and EIGRP. Similar to load, it is also disabled under the default settings.

Cost

Cost is an arbitrary value assigned to a link by the network administrator. This metric may be used to make a certain link more or less preferable over others. Since it may not reflect any physical characteristic of the network, networks that utilize arbitrary cost metrics are difficult to analyze.

2.2.2 Composite Metrics

A composite metric is made of several of the simple metrics that are described above. These metrics may be combined to form a composite metric in an arbitrary way. In this thesis, we focus on composite metrics that are linear combinations of simple metrics:

$$\text{Composite Metric} = \alpha_1 M_1 + \alpha_2 M_2 + \ldots + \alpha_n M_n$$

where M_is are the simple metrics and α_is are constants, normally scaled so that

$$\alpha_1 + \alpha_2 + \ldots + \alpha_n = 1.0$$

It will be our goal to determine the α_i. It should be noted that there will be cases when this is not possible. This can be because of the nature of the network, such as if the network is trivial and there is only one path between a source and a destination, or if one path is better than other paths in every respect. It may also be that a custom metric is being used, such that it does not match any combination of values of links. For example, if the network administrator wanted a particular path to be utilized less, they can increase (or decrease) the value for that path. We have no way of knowing that this is the case or perhaps there is a metric that penalizes certain attributes of that path. However, even if custom metrics are used, we may still get useful results from the analysis because if we determine that the network is, for example, trying to minimize latency, but that happens to be just a coincidence that the custom metrics are favoring paths with lower latency, there is no difference between the two policies from the perspective of how packets get routed.

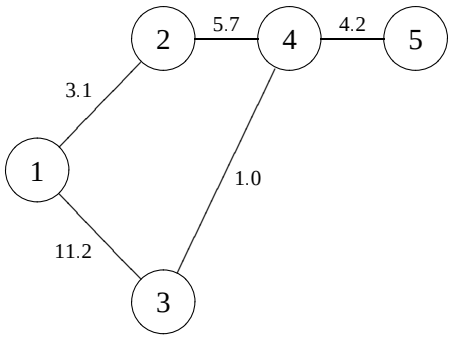
2.3 Routing Algorithms

In this section we will present the major interior routing algorithms that are employed in practice. These algorithms are divided into two categories: distance vector and link state. From a modeling perspective, it does not really matter which category of algorithm is running on the network. This is because both types of algorithms are responsible for routing packets on the

network optimally at all times, but take different approaches to accomplishing that task. As Chapter 3 shows, both algorithms are based on dynamic programming and differ only in the pattern of information exchanges.

2.3.1 Distance Vector Algorithms

The set of distance vector algorithms is one of the two major categories of routing algorithms [13]. In a distance vector algorithm, routers exchange information about routers they can reach. This information is in the form of a vector that contains routers that can be reached, a metric for how much it costs to reach them, and the router that will be used as the next hop.



$$1 \rightarrow 2: \begin{bmatrix} (0.0,1) \\ (3.1,2) \\ (11.2,3) \\ inf \\ inf \end{bmatrix}$$

$$4 \rightarrow 2: \begin{bmatrix} inf \\ (5.7,2) \\ (1.0,3) \\ (0.0,4) \\ (4.2,5) \end{bmatrix}$$

For example, in the network above, router 1 will send a vector to router 2 saying that it knows how to get to 2 and 3 (and itself), how much it costs to get there, and which outgoing interface it will take. Router 4 will also send a vector to 2 with its information, including how to get to 5 and 3. Router 2, in turn, sends a vector to 1,

$$2 \rightarrow 1: \begin{bmatrix} (3.1,1) \\ (0.0,2) \\ (6.7,4) \\ (5.7,4) \\ (9.9,5) \end{bmatrix}$$

causing, the routing table at router 1 to be updated to include a path to 5 and a shorter path to 3. In this example, routers that are unreachable from a particular router were given a value inf. In a real implementation, this symbol will need to be given a numeric value. This value will dictate the maximum length path in a network (which can be $inf - \varepsilon$ where ε is the minimal value a metric can have) and it is up to the designers of the particular algorithm to decide on what that value should be. Vector exchange continues and eventually all routers will know the optimal paths across the network.

Routing Information Protocol (RIP)

The Routing Information Protocol is one of the oldest distance vector protocols still in use today [13]. There are currently two versions in use and they differ in several ways, including how they treat numerical representation of IP networks (v1 is classful, v2 is classless). RIP v2 also provides a means for additional security and authentication. Both versions of RIP use hop count as their metric.

Although not as critical as the metric, it is also good to know the types of timers RIP uses. There is a 30 second update timer which includes a certain amount of jitter, making the timer activate ever $25-35$ seconds. This is to ensure that routers do not synchronize their updates. There is also a timeout timer for a link, which is set to 180 seconds. If a router receives no RIP updates on a link for 180 seconds, the cost for that link is set to inf, which typically has the value of 16. Thus the maximum radius for a network routed by RIP is 15 nodes.

Interior Gateway Routing Protocol (IGRP), and Enhanced Interior Gateway Routing Protocol (EIGRP)

Both Interior Gateway Routing Protocol (IGRP) and Enhanced Interior Gateway Routing Protocol (EIGRP) are CISCO's proprietary routing protocols. IGRP sends updates every 90 seconds, with a 20% random jitter. EIGRP sends updates only when it detects the necessity to do so, such as a link going up or down. EIGRP sends Hello packets to detect these changes every 5 seconds, except on links with link speeds of T1 or slower, then they are sent every 60 seconds. EIGRP uses a timeout value equal to 3 times that of the timer used, so it will detect a link that goes down in 15 seconds if updates are sent every 5 seconds and 180 seconds otherwise.

Both IGRP and EIGRP utilize a composite metric. The components of this metric include bandwidth, delay, load, and reliability. Bandwidth and delay are first scaled before being used in the metric. BW_{IGRP} is calculated by dividing 10^7 by the link bandwidth, in kilobits. Thus

$$BW_{IGRP} = \frac{10^7}{BW_{link}}$$

The constant 10^7 limits the maximum distinguishable bandwidth of a link to 10 Gigabits, but the constant may be changed by the network administrator to allow for higher bandwidth links. The delay component, DLY_{IGRP} is calculated by dividing the delay of the link in microseconds by 10. Thus

$$DLY_{IGRP} = \frac{DLY_{link}}{10}$$

Both load and reliability of a link are expressed in terms of a percentage. That percentage is stored as an unsigned 8 bit integer where 255 is 100% and 1 is 0%. The metric has 5 constants, $k_1, \ldots, k_5$, that can be set by the network administrator. The metric is calculated for an entire path as follows

$$\begin{aligned} metric &= (k_1 * BW_{IGRP(min)} + \frac{k_2 * BW_{IGRP(min)}}{256 - LOAD} \\ &\quad + k_3 * DLY_{IGRP(sum)}) * \frac{k_5}{RELIABILITY + k_4} \end{aligned}$$

The component $DLY_{IGRP(sum)}$ stands for the sum of the DLY_{IGRP} values of all links along the path to the destination. The component $BW_{IGRP(min)}$ stands for the BW_{IGRP} value of the link with the smallest bandwidth along the path to the destination. If we expand the metric, we get

$$\begin{aligned} metric &= k_1 * BW_{IGRP(min)} * \frac{k_5}{RELIABILITY + k_4} \\ &+ \frac{k_2 * BW_{IGRP(min)}}{256 - LOAD} * \frac{k_5}{RELIABILITY + k_4} \\ &+ k_3 * DLY_{IGRP(sum)} * \frac{k_5}{RELIABILITY + k_4} \end{aligned}$$

Since most links are very reliable, it is likely that reliability will be very close to 255 and affect every link for a given router equally. Thus it becomes a constant multiplier that may be factored out in terms of calculating the metric.

It should also be noted that by default, CISCO uses constants $k_1 = k_3 = 1$ and $k_2 = k_4 = k_5 = 0$, thus the default metric is

$$metric = BW_{IGRP(min)} + DLY_{IGRP(sum)}$$

2.3.2 Link State Algorithms

Link state algorithms are the second category of routing algorithms. They are based on Dijkstra's shortest path algorithm [21]. Dijkstra's algorithm requires knowledge of the network topology, so the routing algorithm begins by ensuring that every router knows the entire network topology. During this phase, every router sends out packets to every adjacent router that contain information about what routers it is immediately adjacent to. The routers that receive those packets forward them to adjacent routers. Eventually, every router on the network will receive a packet from every other router, providing the information necessary to reconstruct an accurate topology. Once a router has the entire topology, it runs Dijkstra's algorithm, which will determine the shortest path from that router to every other router on the network.

Open Shortest Path First (OSPF)

Open Shortest Path First (OSPF) is a link state algorithm that was developed by the Internet Engineering Task Force (IETF) through a series of RFCs (1131, 1247, 2328) [18; 19; 20]. The metric OSPF uses is a cost metric, but the RFCs do not define what that cost should represent. The value for the cost can range from 1 to 65535 (a 16-bit number). CISCO uses bandwidth, by default, to represent the cost. The metric is calculated as $\frac{10^8}{BW_{link}}$, where BW_{link} is the bandwidth of the physical link in bits/second. The 10^8 constant can be changed by the network administrator. Some vendors leave the default cost set to 1 [13], reducing the metric to a hop count metric. It is also feasible for a network administrator to change the cost metric to reflect other attributes of the link [13]. OSPF uses a number of timers, including keep-alives for links, but the one most important to us is the Link State Acknowledgment (LSA) packet timer. The LSA packets contain adjacency information and are used by routers to create and maintain their topology information. On CISCO routers, these packets are sent every every 10

to 1800 seconds, with 240 seconds being the default. This number should vary per size of the network; larger networks should have smaller timeouts while smaller networks can have larger intervals.

Integrated Intermediate System to Intermediate System (IS-IS)

Integrated Intermediate System to Intermediate System (IS-IS) is a link state protocol similar to OSPF. IS-IS uses Hello packets to discover neighboring routers and Link State Packets (LSPs) to determine the entire network topology for every router. LSPs are sent as updates every 15 minutes, with a random jitter of up to 25%. ISO 10589 (the standard for IS-IS) defined four metrics that can be used by IS-IS: default, delay, expense, and error. Each metric is an unsigned 6 bit integer. The default metric is required and must be set manually for each link. CISCO sets it to 10 by default. The last three metrics are optional. Delay refers to the latency of the link, cost to a monetary cost of using the link, and error to the error probability of the link. CISCO only supports the default metric and under the default setting this reduces the metric to a hop count metric.

2.4 Network Discovery

From a modeling perspective, network discovery is an important aspect of capturing the essentials of a routing context. We need to have as close to a complete and accurate view of the network as possible in order to situate the operation of specific policies. Specifically, it will be important to capture the location of routers participating in the dynamic routing algorithm and the characteristics of links in the network.

2.4.1 Topology Discovery

[5] present several algorithms for discovering the topology of an IP network. The basic tools they use are `ping`, broadcast ping, `traceroute` [16], and Domain Name System (DNS) zone transfers. `ping` sends a ICMP echo request to an address and uses a response to determine if the hose is active. This may be unreliable with desktop hosts running personal firewalls or networks protected by firewalls at their bounds, since they may not respond to pings. However, routers should still respond to pings, otherwise it will make network administration more cumbersome.

Directed broadcast pings ping an entire subnet of a network, rather than a single machine. Since this technique is also responsible for denial of service attacks (“smurf attacks”) [22], security conscious network administrators will disable broadcast pings. Thus, they are not a reliable means of gathering information, except on poorly administered networks.

`traceroute` is the famous algorithm by Van Jacobson [16] that sends packets with increasing values in the Time to Live (TTL) field in the IP header and records the TTL-expired ICMP messages. When a router receives a packet with a TTL of 1 it should drop the packet and send an ICMP message back to the source of the packet, saying that the TTL for the packet has expired. This behavior is defined in [17]. However, some routers may be configured to not abide by the RFC, or firewalls may be configured to drop these packets at the network boundary. This

should not be the default behavior for large autonomous systems, since this makes investigating and diagnosing network problems more difficult. DNS zone transfers involve querying a DNS server for the mapping of every hostname to IP address on the network it is responsible for. From a security standpoint, it is a bad idea to allow zone transfers from outside of the network, thus only the most poorly maintained networks will be vulnerable to this.

[5] present the results of running four algorithms on the cornell.edu domain. One of the algorithms is the Simple Network Management Protocol (SNMP), and two require the use of DNS zone transfers, and thus are not of very much use to us (both should be blocked at the network boundary). The fourth uses a combination of pings and traceroute, and although it has the worst results of the four, it still boasts a 90% accuarcy rate and discovers 144 of 155 routers on the network.

2.4.2 Link Attribute Discovery

Following the discovery of the network topology, we would like to ascertain more information about the links used in the network. It is important to study the influence of links on the preferential selection of paths, especially their contributions to metrics. Instrumentation for capturing metrics such as delay and link bandwidth are most common. Delay of a link may be determined using traceroute [16]. Determining the bandwidth of a links on a network is the subject of a lot of research. Many methods exist for determining the bottleneck bandwidth of a particular path [6]. However, we are interested in determining the bandwidth of every link in the network. There is a tool by Van Jacobson called pathchar [11]that can be used to accomplish this task. Unfortunately, there are conditions under which it performs poorly. These are discussed in detail in [10].

The load metric causes some amount of trouble. We are not able to reliably discover the value for a load for a particular link. However, the usefulness of the load component may be questioned. It is used as the divisor of $BW_{IGRP(min)}$ in CISCOs IGRP and EIGRP algorithms. Consider the terms that use $BW_{IGRP(min)}$

$$\begin{aligned} k_1 * BW_{IGRP(min)} \quad + \quad & \frac{k2 * BW_{IGRP(min)}}{256 - LOAD} = \\ = \quad & \frac{k_1 * (256 - LOAD) * BW_{IGRP(min)} + k2 * BW_{IGRP(min)}}{256 - LOAD} \\ = \quad & \frac{k_1 * 256 - k_1 * LOAD + k2}{256 - LOAD} * BW_{IGRP(min)} \end{aligned}$$

We see that load and constants k_1 and k_2 determine how much the minimum bandwidth affects the overall metric. However, the load of the link connected directly to the router may have no relation to the $BW_{IGRP(min)}$ value of the path. For example, if the directly connected 1 Gigabit link is 90% loaded, it still has more available bandwidth than a 10 Megabit link that may be the minimum link along the path. Thus, if everything else is equal, the router may choose to route along another path, that may have a 10% loaded 10 Megabit link, instead of the 90% loaded 1 Gigabit link, which is not the desired behavior. Therefore, the usefulness of the load metric in the composite metric may be in question.

Chapter 3

Overview of Inverse Reinforcement Learning

With a view toward routing metric discovery, this chapter presents the Inverse Reinforcement Learning (IRL) techniques described in [1]. We will first present the notation used by Russell and Ng, since similar notation will be used here. We will go over the main theorem in [1] and their suggestions as to how to apply it to solve IRL problems.

3.1 Definitions

Inverse Reinforcement Learning (IRL) is easier to understand by first studying Reinforcement Learning (RL) – the problem of an agent using experience to improve its behavior in a given environment. A Reinforcement Learning problem requires an environment – a set of states the agent can be in, and actions that the agent can take. A state transition matrix provides details on the outcome of specific actions in a given state. The environment should also provide the agent with some sort of feedback when the agent takes a certain action in a certain state. This feedback is called the reward and it is the goal of the agent to maximize the cumulative (possibly discounted) sum of the rewards it earns. The environment is said to have the Markov property if how it responds to a particular action taken by the agent is only based on the state the agent is in and not on any previous state or an outside influence. An environment that satisfies the Markov property is a Markov Decision Process (MDP). If the number of actions and states is finite, it is a finite MDP, or Markov Decision Process.

3.1.1 Notation

Russell and Ng define a finite MDP as a tuple $(S, A, \{P_{sa}\}, \gamma, R)$, where

- S is a finite set of N states
- $A = \{a_1, \ldots, a_k\}$ is a set of k actions
- $P_{sa}(\cdot)$ are the state transition probabilities upon taking action a in state s

- $\gamma \in [0,1)$ is the discount factor
- $R : S \to \mathbb{R}$ is the reinforcement function, bounded in absolute value by $R_{\max}$

There are also several functions defined as part of the reinforcement learning setting. A policy is a mapping $\pi : S \to A$; it defines what action the reinforcement learning agent should take in each state $s \in S$. Notice that a policy is a property of the agent, hence different agents can use different policies in a given environment. A value function for a policy π, when evaluated in state s_1, is

$$V^{\pi}(s_1) = E[R(s_1) + \gamma R(s_2) + \gamma^2 R(s_3) + \ldots | \pi]$$

where $(s_1, s_2, \ldots)$ is the sequence of states passed through when following the policy π. There is also the action-value function, or Q-function, defined as

$$Q^{\pi}(s,a) = R(s) + \gamma E_{s' \sim P_{sa}(\cdot)}[V^{\pi}(s')]$$

The Q function gives the value of taking action a in state s while following the policy π. The notation $s' \sim P_{sa}(\cdot)$ means the expectation with respect to s' is distributed according to $P_{sa}(\cdot)$. Two more functions are defined, the optimal value function,

$$V^*(s) = sup_{\pi} V^{\pi}(s)$$

and the optimal Q-function, as

$$Q^*(s,a) = sup_{\pi} Q^{\pi}(s,a)$$

For finite MDPs, it is well known that V^* and Q^* are independent of policy (unlike V^{π} and Q^{π}).

It is suggested that for discrete finite spaces (if we have a finite number of states and actions), functions V^{π} and R can be represented as vectors, indexed by a particular enumeration of the state space. Since $|S| = N$, these vectors will be N-dimensional. Thus, R becomes an N-dimensional vector $\mathbf{R}$, whose ith value is the reward received at state i and the ith value in $\mathbf{V}^{\pi}$ is the value function evaluated at state i. Also, we can represent state transitions $P_{sa}(\cdot)$ as matrices $\mathbf{P}_a$, indexed by actions $a \in A$. The element in position (i,j) in $\mathbf{P}_a$ is the probability of the state transitioning to state j when action a is taken in state i. Russell and Ng also define $\prec$ and $\preceq$ as strict and non-strict vector inequalities, such that $\mathbf{x} \prec \mathbf{y}$ iff for $\forall i, \mathbf{x}_i \leq \mathbf{y}_i$. There are also two important theorems concerning MDPs that are presented in the paper.

Theorem 3.1 (Bellman Equations from Ng and Russell [1]) *Let an MDP $M = (S, A, \{P_{sa}\}, \gamma, R)$ and a policy $\pi : S \to A$ be given. Then, for all $s \in S, a \in A$, V^{π} and Q^{π} satisfy*

$$V^{\pi}(s) = R(s) + \gamma \sum_{s'} P_{s\pi(s)}(s') V^{\pi}(s') \tag{3.1}$$

$$Q^{\pi}(s,a) = R(s) + \gamma \sum_{s'} P_{sa}(s') V^{\pi}(s') \tag{3.2}$$

Notice that in equation 3.1 there is an implicit marginalization over the space of actions imposed by policy π.

In RL, we are given S, A and provided access to $\mathbf{P}_{sa}$ and $\mathbf{R}$ (either directly or through interactions). The goal of the agent is to find a policy π that maximizes $\mathbf{V}^{\pi}$ (or $\mathbf{Q}^{\pi}$). In IRL, we are given S, A, and a policy that presumably optimizes some value function. The goal is to determine the value function (which is dependent on γ, $\mathbf{P}_{sa}$, and $\mathbf{R}$). Traditionally the goal is posed as one of estimating $\mathbf{R}$, given γ and $\mathbf{P}_{sa}$.

3.2 Characterizing IRL Solutions

The main theorem Russell and Ng present defines the set of solutions that the reward vector $\mathbf{R}$ can belong to, while keeping the current policy optimal.

Theorem 3.2 (Ng and Russell) *Let a finite state space S, a set of actions $A = \{a_1, \ldots, a_k\}$, transition probability matrices $\{\mathbf{P}_a\}$, and a discount factor $\gamma \in [0,1)$ be given. Then the policy π given by $\pi(s) \equiv a_1$ is optimal if and only if, for all $a = a_2, \ldots, a_k$, the reward $\mathbf{R}$ satisfies*

$$(\mathbf{P}_{a_1} - \mathbf{P}_a)(\mathbf{I} - \gamma \mathbf{P}_{a_1})^{-1}\mathbf{R} \succeq 0$$

Ng and Russell point out that if $\succeq$ is replaced with $\succ$, then $\pi(s) \equiv a_1$ is the unique optimal policy. They also point out several deficiencies of this characterization. One is that a the zero vector is always a solution, since then every policy is optimal. Ng and Russell also mention that for a given MDP, there may be many reward vectors $\mathbf{R}$ that make a particular policy optimal. Thus they suggest placing additional requirements on $\mathbf{R}$. Their suggestion is to select solutions which maximize the difference between the best action and the second best action. Thus they want to maximize

$$\sum_{s \in S} (Q^{\pi}(s, a_1) - \max_{a \in A/a_1} Q^{\pi}(s, a))$$

They also add a weight, $-\lambda \|\mathbf{R}\|_1$, so that solutions with smaller rewards are preferred.

There are different ways to characterize or restrict the space of reward functions. The 'shaping' literature [4] is filled with may ideas. See also Ng, Harada, and Russell [7] for a characterization using reward transformation based on potential functions.

3.3 Algorithm when MDP is known

The final formulation becomes

$$\begin{aligned} \text{maximize} \quad & \sum_{i=1}^{N} \min_{a \in A/a_1} \{(\mathbf{P}_{a_1}(i) - \mathbf{P}_a(i))(\mathbf{I} - \gamma \mathbf{P}_{a_1})^{-1}\mathbf{R}\} - \lambda \|\mathbf{R}\|_1 \\ \text{such that} \quad & (\mathbf{P}_{a_1} - \mathbf{P}_a)(\mathbf{I} - \gamma \mathbf{P}_{a_1})^{-1}\mathbf{R} \geq 0, \forall a \in A/a_1 \\ & |\mathbf{R}_i| \leq R_{max}, i = 1, \ldots, N \end{aligned}$$

where $\mathbf{P}_a(i)$ is the ith row of $\mathbf{P}_a$. This is then posed as a linear program in order to find the $\mathbf{R}$ vector:

$$\text{maximize } \mathbf{c} \cdot \mathbf{R} \text{ such that } \mathbf{MR} \succeq \mathbf{b}$$

for a matrix $\mathbf{M}$ and vectors $\mathbf{c}$ and $\mathbf{b}$. But it becomes difficult to factor out $\mathbf{R}$ from the min function,

$$\min_{a \in A/a_1} \{\mathbf{P}_{a_1}(i) - \mathbf{P}_a(i))(\mathbf{I} - \gamma \mathbf{P}_{a_1})^{-1}\mathbf{R}\}$$

since the value of $\mathbf{R}$ determines the result of that function. This problem originates from wanting to maximize

$$\sum_{s \in S}(Q^{\pi}(s,a_1) - \max_{a \in A/a_1} Q^{\pi}(s,a))$$

We have no way of evaluating the maximum $Q^{\pi}(s,a)$ for $a \in A/a_1$, unless we already have $\mathbf{R}$ or are able to see what the second-best policy is by having it performed by the agent. Since our agent is the routing protocol (see Chapter 4), we have no way of making it run sub-optimally, thus we cannot use the criteria suggested by Ng and Russell. Hence we adopt a different function, also stated by Ng and Russell:

$$\text{maximize} \sum_{s \in S} \sum_{a \in A \setminus a_1} Q^{\pi}(s,a_1) - Q^{\pi}(s,a) \tag{3.3}$$

Since $\pi(s) \equiv a_1$, equation 3.1 maybe written in vector notation as

$$\begin{aligned} \mathbf{V}^{\pi} &= \mathbf{R} + \gamma \mathbf{P}_{a_1} \mathbf{V}^{\pi} \\ \mathbf{V}^{\pi} - \gamma \mathbf{P}_{a_1} \mathbf{V}^{\pi} &= \mathbf{R} \\ (\mathbf{I} - \gamma \mathbf{P}_{a_1}) \mathbf{V}^{\pi} &= \mathbf{R} \\ \mathbf{V}^{\pi} &= (\mathbf{I} - \gamma \mathbf{P}_{a_1} \mathbf{V}^{\pi})^{-1} \mathbf{R} \end{aligned}$$

and equation 3.2 can be written as

$$Q^{\pi}(s,a) = \mathbf{R}_s + \gamma \mathbf{P}_a(s) \mathbf{V}^{\pi}$$

Then the equation 3.3 may be written as

$$\begin{aligned} \sum_{s \in S} \sum_{a \in A \setminus a_1} (Q^{\pi}(s,a_1) - Q^{\pi}(s,a)) &= \sum_{i=1}^{N} \sum_{j=2}^{k} (Q^{\pi}(i,a_1) - Q^{\pi}(i,a_j)) \\ &= \sum_{i=1}^{N} \sum_{j=2}^{k} (\mathbf{R}_i + \gamma \mathbf{P}_{a_1}(i) \mathbf{V}^{\pi} - \mathbf{R}_i - \gamma \mathbf{P}_{a_j}(i) \mathbf{V}^{\pi}) \\ &= \sum_{i=1}^{N} \sum_{j=2}^{k} \gamma (\mathbf{P}_{a_1}(i) - \mathbf{P}_{a_j}(i)) \mathbf{V}^{\pi} \\ &= \gamma \sum_{i=1}^{N} \sum_{j=2}^{k} ((\mathbf{P}_{a_1}(i) - \mathbf{P}_{a_j}(i))(\mathbf{I} - \gamma \mathbf{P}_{a_1})^{-1} \mathbf{R} \end{aligned}$$

Since the γ outside the summations is a constant, it can be ignored in a maximization problem. Thus our problem becomes

$$\begin{aligned} \text{maximize} \quad & \sum_{i=1}^{N}\sum_{j=2}^{k} \{(\mathbf{P}_{a_1}(i) - \mathbf{P}_{a_j}(i))(\mathbf{I} - \gamma\mathbf{P}_{a_1})^{-1}\mathbf{R}\} - \lambda\|\mathbf{R}\|_1 \\ \text{such that} \quad & (\mathbf{P}_{a_1} - \mathbf{P}_a)(\mathbf{I} - \gamma\mathbf{P}_{a_1})^{-1}\mathbf{R} \geq 0, \forall a \in A/a_1, \\ & |\mathbf{R}_i| \leq R_{max}, i = 1, \ldots, N \end{aligned}$$

This will be the formulation adopted in this thesis.

3.4 Algorithm when MDP is not known

Russell and Ng also present techniques to be used when the entire MDP is not known. This would be useful, since we would not need to know the entire topology and link characteristics of the network. Unfortunately, their techniques rely on the ability to generate trajectories (sequences of state transitions) under the optimal policy, which we *can* do, and under any policy of our choice, which we *can not* do. This is because we have no way of coercing the routing algorithm into routing sub-optimally. Thus the technique for IRL from sample trajectories does not apply.

Chapter 4

Using Inverse Reinforcement Learning for Network Metric Discovery

In this chapter we will develop a model for the network and routing algorithm such that the routing problem is posed as a Reinforcement Learning problem. We will then use the IRL techniques presented in the previous chapter with the developed model.

4.1 Modeling the Network and Routing Algorithm as a Reinforcement Learning Problem

In this section we will demonstrate our approach to modeling the IP network and the routing algorithm that is run on each router as a reinforcement learning problem. For an overview of reinforcement learning as a feasible and preferable approach to network routing see [8]. We describe a similar approach and show how it can be used with IRL techniques. We will first show how to think of the classical algorithm for finding minimal distance between pairs of nodes on a graph as solving a reinforcement learning problem. We will then present how to set up the final reinforcement learning problem so that it could be used with the IRL algorithm.

4.1.1 Modeling the IP Network as a Graph

A network of routers can be viewed as a directed or undirected weighted graph. The routers are the nodes and the physical links between the routers are the edges. The metric the routing algorithm assigns each link is the weight of the edge. If the link characteristics, such as delay and bandwidth, are symmetric in both directions, then the graph is undirected; otherwise the graph becomes a directed multi-graph, with each physical link represented as two edges going in opposite directions, with different weights for each edge. It will be assumed that the links considered here are symmetric, thus the network is modeled as an undirected weighted graph.

In order to apply the inverse reinforcement learning algorithm, we must conceptualize a reinforcement learning agent and an environment within which it is trying to optimize a given reward function. There are several ways of representing the reinforcement learning agent. One approach is to think of a single router as the agent and thus the routers in the network

will communicate information they learn with each other so as to arrive at an optimal routing policy [2; 3]. This gives rise to a multi-agent reinforcement learning framework. We will take a different approach and think of the same routing algorithm that is run on every router on the network as the agent. This is possible if we think of the network as the environment and every router as implementing the routing algorithm in order to optimally interact in the environment. Thus the routing algorithm is viewed as a single network-wide entity, with each router doing a small part and its actions contributing to the whole result. From the point of view of a single packet being forwarded along in the network, a router makes the decision of which of its interfaces to send the packet along, but that decision is determined by the same algorithm that runs on every other router on the network. Thus, in a sense, there is a single algorithm that governs the whole network according to a particular policy. It is this algorithm that should be considered the reinforcement learning agent. This viewpoint leads to a single-agent reinforcement learning formulation.

Now we need to identify the environment, or the set of states that the agent can be in and the actions the agent may take. The set of states should include everything that can influence the decision of the routing algorithm and also be accessible to the algorithm (observable). It is clear that the physical network is the environment that the agent operates in, but the set of states still needs to be identified, as is the set of actions that the agent can take. It is natural to think of the physical links as being the actions, since the routing algorithm chooses which link to send a packet on so as to minimize the overall cost, for a particular metric. This approach is also taken by [2; 3]. Thus it remains to determine the set of states the agent can be in.

In order to determine the states in the environment, consider the example of a packet traveling through the network on its way from its source to its destination. At a given router, the routing algorithm computes (or has already pre-computed) the minimal path to take to the destination. It then chooses the interface that lies on that path and forwards the packet to the router that is connected by that link. In the case of a single packet, the state of the environment should include the location of the packet, or, which router the packet is currently at. It should also be noted that a router will send a packet along a certain path when it is going to a certain destination, but may forward it along a different path, if it is going to a different destination. Thus, the destination of the packet should also be part of the state, because it influences the action a router takes. Therefore, the state so far includes the current location of the packet and the destination of the packet. There is a third component of the state, but its necessity is dictated by the formulation of the IRL algorithm and will be covered later.

4.1.2 All Node Pairs Shortest Paths

The dynamic programming algorithm for finding the shortest path between every pair of nodes, such as the algorithm presented in [12], is really solving a reinforcement learning problem. We can rewrite the result obtained by the algorithm in reinforcement learning terms. This is akin to how George Cybenko shows the use of dynamic programming to solve the shortest path problem between a given pair pair of nodes [23].

Figure 4.1 presents the algorithm. Here, $Weight[i, j]$ represents the cost for getting between node i and node j. If our state is written in terms of $location \times destination$ nodes, we can write the state where the current location is i and the destination is j as $s_{i,j}$. If we assume that

Input: *Weight* – an $n \times n$ adjacency matrix representing a weighted graph.
Output: *Weight*, containing the lengths of the shortest paths.

```
begin
   for m := 1 to n do
      for x := 1 to n do
         for y := 1 to n do
            if Weight[x,m] + Weight[m,y] < Weight[x,y] then
               Weight[x,y] := Weight[x,m] + Weight[m,y]
end
```

Figure 4.1: All Node Pairs Shortest Paths Algorithm (from [12]

Weight reflects the knowledge of the reinforcement learning agent, then the value function for its policy π is

$$V^{\pi}(s_{i,j}) = \text{Weight}[i, j]$$

From this, we may recover the policy π, for getting between every node to every destination. We define π as

$$\pi(s_{loc,dst}) = a \text{ such that } V^{\pi}(s_{loc,dst}) - cost(a) = V^{\pi}(s_{loc',dst}), \forall a \in \mathscr{A}(s_{loc,dst})$$

where loc' is the node connected to loc by the edge represented by the action a. To relate actions to edges, $\mathscr{A}(s_{loc,dst}) \sim \{e_{loc,loc'} \mid \forall\ loc' \text{ adjacent to } loc\}$, so the set of actions in state $s_{loc,dst}$ is the set of edges that link to loc. After the algorithm in Figure 4.1 is completed, it is clear that π will correspond to π^*, or the optimal policy for reaching a destination node from any other node.

4.1.3 Model used with IRL

Before our model of the environment can be used with the inverse reinforcement learning algorithm, we need to add another component to the set of states in our environment. This component is necessary because we wish to associate rewards with links, or actions. In reinforcement learning, rewards may be associated with states, or arriving in a particular state, or the rewards may be associated with taking a particular action in a given state. Russell and Ng take the former approach in [1], and we will take the latter approach here.

To demonstrate the necessity of this third component, the following example is considered. Figure 4.2 depicts a four node subgraph of a network. Each edge has a negative weight that represents the reward for taking that link, thus if the routing algorithm is trying to maximize the reward for a path, it will pick the edges with the highest values. Each edge is also labeled with two values which represent the port numbers for the link from the perspective of a router.

Consider that an optimal path from node 1 to a certain destination lies through node 4. Then the optimal path to node 4 is $1 \rightarrow 2 \rightarrow 3 \rightarrow 4$. The reward for this sequence of actions is -3,

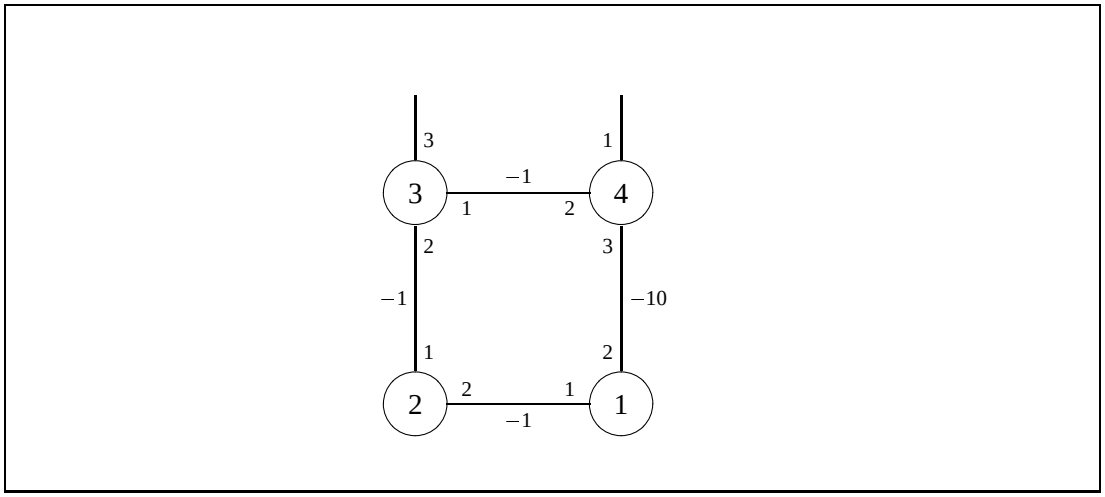


Figure 4.2: Example Network, Action-Value function gives Rewards for Links

as opposed to -10, for taking the direct route $1 \rightarrow 4$. The problem lies in that the notation in the inverse reinforcement learning algorithm associates rewards with states. Thus, assuming a single desination, then when the packet reaches node 4, there is no way of determining whether the reward for taking link 1 at node 3 is -1 or -10 because all is known is that the packet is now at node 4 and it received either a -1 or -10 as the reward. Therefore, the link that the packet arrives from should be included as part of the state. In that case, assuming a single destination, when the packet reaches node 4, there can be two possible states: 1) packet is at node 4 and came on interface 2, or 2) packet is at node 4 and came on interface 3 (assume that the packet will not come on interface 1). The reward for the former is -1 and for the latter is -10. Thus the inverse reinforcement learning algorithm will be able to distinguish between the two cases and assign rewards correctly.

Thus, finally, our definition of state includes three components: the router the packet is currently at, the destination router for the packet, and the port number on the router that the packet came on. Effectively, we "amplify" the definition of state to provide sufficient addressability. The actions are the ports that a router can pick from, and the reinforcement learning agent is the routing algorithm that is running on the network. Since the reward that the agent receives is related to the metric weight of each link, it can be seen that the Markov property holds for this environment. We now have everything we need to be able to utilize the IRL techniques to attempt to recover the reward (metric) used by the algorithm.

4.2 Using Inverse Reinforcement Learning with the Network Model

In this section we will use the model of the network developed in the previous section with the IRL algorithm presented in chapter 3. We will first present an example of how IRL can apply to our model. Then we will present a way for using IRL to recover the structure of a composite

metric.

4.2.1 IRL Without Composite Metric

We will first demonstrate how IRL techniques may be applied to the network model, without attempting to determine the composite metric. This technique cannot ascertain the actual rewards being optimized, since we can only observe the optimal policy and have no way of determining how the suboptimal paths relate to each other, or how much worse one suboptimal path is than another. However, this should give us insight into why and how IRL techniques can determine the relationship between the components in a composite metric.

Setup

We will use, as an example, the topology in Figure 4.3. There are 8 nodes (routers), and each node has 5 edges (links). Each edge has one weight and two ports, one at each end, where it connects to a node. We will designate nodes 7 and 8 as possible destinations. Each node has at least one "unconnected" link. These links serve two purposes. One, for simplicity of notation and construction of the IRL procedure, we would like the same number of actions to be available in every state, and thus every node needs to have the same number of ports. When taken, the unconnected ports lead "back" to the node, with a negative reward, and thus would be avoided by the RL agent (routing algorithm). Also, each node needs at least one unconnected port, to represent it being a potential source for packets. If a node is a source, there is no previous node that a packet came on, thus an unconnected port is used to model a packet being generated at a node. In this example, every node is a potential source.

Now we need to create the state transition matrices (**P** matrices) for each action. There will be 5 **P** matrices and each **P** matrix is 80×80 in dimension (8 nodes $\times$ 2 destinations $\times$ 5 ports $=$ 80 states), are too large to present their contents here. The process for constructing them is simple, since the state transition probabilities for every action are either 100% for nodes connected by a link and 0% for nodes not connected by that link (if a router sends a packet on a link, it will end up at the router at the other end of the link and not another router). So, first we come up with a scheme for enumerating the states. This can be arbitrary and we use the following procedure. First, divide the 80 states into two blocks of 40 states, one block per destination (first for node 7, second for node 8). Then each block of 40 states is subdivided into 8 sub-blocks of 5 states. Each sub-block represents the node the packet is currently at. Each of the 5 states in a sub-block represents the port a packet came on. Thus state 12 means the packet is going to node 7, is located at node 3 and it came on port 2.

There is one subtlety when creating the **P** matrices. For ease of notation, we would like the optimal action to be action 1. In our example, for nodes $1, 2, 3, 4$ and 6, this is true. However, nodes $5, 7, 8$ need to have their ports renumbered, with respect to which of the **P** matrices the transition probabilities for ports 1 and 2 will go to. This is because, if the destination is 7, the optimal path from node 5 is to take port 1, but if the destination is 8, then it must take port 2. The case is similar for nodes 7 and 8, if they are their own destination, they need to be a terminal state, and if not, they need to send the packet on port 1. Thus, to reflect this situation, the transition probabilities need to be modified.

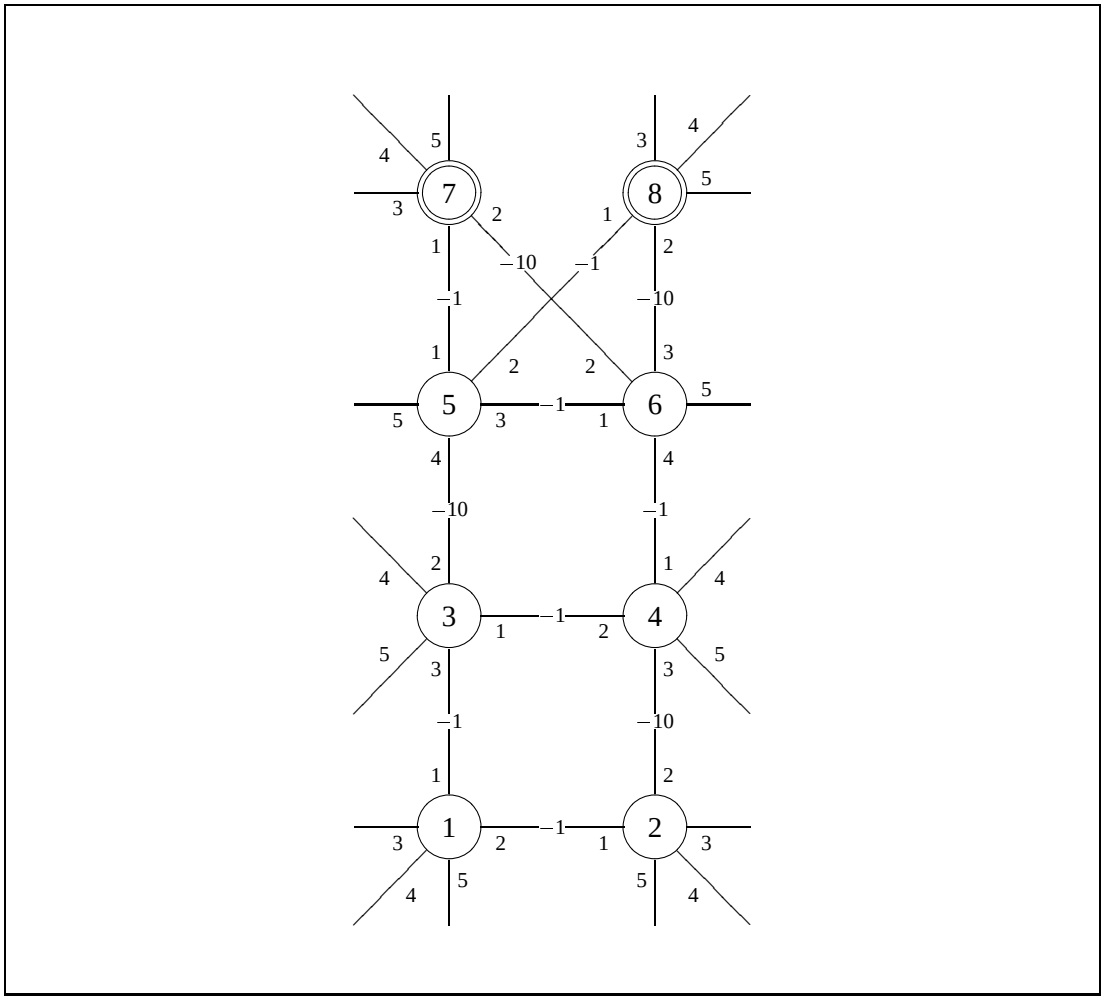


Figure 4.3: Example Network

IRL and Linear Programming

Now that we have the transition probability matrices for each action, we use the IRL method to set up a linear programming problem. Since the values for our rewards need to be negative (or else the network will find a loop and cycle the packet on it to accumulate rewards, which is obviously not the desired behavior), we will want to solve the dual problem of the linear programming problem. This is because as stated, the solution to a linear programming problem cannot have negative values [24]. We will want to set up our problem to find a vector $\mathbf{x}$ such that

$$\mathbf{bx} \text{ is maximized with respect to } \mathbf{Mx} \succeq \mathbf{c}$$

Then we can find the dual solution [24], which is the reward vector we are looking for:

$$\mathbf{R} = \mathbf{b} - \mathbf{Mx}$$

Our **b** vector will be derived from

$$\begin{aligned}\mathbf{bR} &= \sum_{i=1}^{N}\sum_{j=2}^{k}\{(\mathbf{P}_{a_1}(i)-\mathbf{P}_{a_j}(i))(\mathbf{I}-\gamma\mathbf{P}_{a_1})^{-1}R\}-\lambda\|\mathbf{R}\|_1 \\ &= (\sum_{i=1}^{N}\sum_{j=2}^{k}\{(\mathbf{P}_{a_1}(i)-\mathbf{P}_{a_j}(i))(\mathbf{I}-\gamma\mathbf{P}_{a_1})^{-1}\}-\lambda)\mathbf{R}\end{aligned}$$

Thus

$$\mathbf{b}=\sum_{i=1}^{N}\sum_{j=2}^{k}\{(\mathbf{P}_{a_1}(i)-\mathbf{P}_{a_j}(i))(\mathbf{I}-\gamma\mathbf{P}_{a_1})^{-1}\mathbf{R}\}-\lambda$$

We then construct our **M** matrix, which represents the conditions

$$\begin{aligned}&(\mathbf{P}_{a_1}-\mathbf{P}_a)(\mathbf{I}-\gamma\mathbf{P}_{a_1})^{-1}\mathbf{R}\geq 0,\forall a\in A\setminus a_1,\\ &|\mathbf{R}_i|\preceq R_{max}, i=1,\ldots,N\end{aligned}$$

Since there are 4 actions in the set $A\setminus a_1$, and each $(\mathbf{P}_{a_1}-\mathbf{P}_a)(\mathbf{I}-\gamma\mathbf{P}_{a_1})^{-1}$ is an 80×80 matrix, the **M** matrix will become a 320×80 matrix. Notice that the **c** vector at this point is a 320 element 0-vector. Now, we need to satisfy the second condition, $|\mathbf{R}_i|\leq R_{max}, i=1,\ldots,N$. We will satisfy it by appending an 80×80 identity matrix, which is multiplied by -1 to **M**, and a vector of 80 elements, each equal to R_{max} to **c**. Thus, our final **M** matrix and **c** vector will be

$$\mathbf{M} \qquad \mathbf{R} \qquad \geq \qquad \mathbf{c}$$

$$\begin{bmatrix} \begin{bmatrix} (\mathbf{P}_{a_1} - \mathbf{P}_{a_2})(\mathbf{I} - \gamma \mathbf{P}_{a_1})^{-1} \end{bmatrix} \\ \begin{bmatrix} (\mathbf{P}_{a_1} - \mathbf{P}_{a_3})(\mathbf{I} - \gamma \mathbf{P}_{a_1})^{-1} \end{bmatrix} \\ \begin{bmatrix} (\mathbf{P}_{a_1} - \mathbf{P}_{a_4})(\mathbf{I} - \gamma \mathbf{P}_{a_1})^{-1} \end{bmatrix} \\ \begin{bmatrix} (\mathbf{P}_{a_1} - \mathbf{P}_{a_5})(\mathbf{I} - \gamma \mathbf{P}_{a_1})^{-1} \end{bmatrix} \\ \begin{bmatrix} -I \end{bmatrix} \end{bmatrix} \begin{bmatrix} R \end{bmatrix} \geq \begin{bmatrix} 0 \\ \vdots \\ \\ \\ \\ \\ \\ \\ \vdots \\ 0 \\ R_{max} \\ \vdots \\ \\ R_{max} \end{bmatrix}$$

Now we can solve for **x** and use it to get **U**. Our result vector for **U**, organized by router, is

$$\begin{array}{ll} \{-6.05, 0., -1.05, -1.05, -1.05\} & \{-6.05, 0., -1.05, -1.05, -1.05\} \\ \{-6.05, -6.05, -1.05, -1.05, -1.05\} & \{-6.05, -6.05, -1.05, -1.05, -1.05\} \\ \{-6.05, -6.05, 0., -1.05, -1.05\} & \{-6.05, -6.05, 0., -1.05, -1.05\} \\ \{-6.05, 0., -6.05, -1.05, -1.05\} & \{-6.05, 0., -6.05, -1.05, -1.05\} \\ \{-1.05, 0., 0., -6.05, -1.05\} & \{0., -1.05, 0., -6.05, -1.05\} \\ \{-6.05, -1.05, -6.05, 0., -1.05\} & \{-6.05, -6.05, -1.05, 0., -1.05\} \\ \{0., -6.05, -1.05, -1.05, -1.05\} & \{-6.05, -6.05, -1.05, -1.05, -1.05\} \\ \{-6.05, -6.05, -1.05, -1.05, -1.05\} & \{0., -6.05, -1.05, -1.05, -1.05\} \end{array}$$

Now, if we overlay these values on the network topology, we can observe the two sets of rewards determined by the algorithm. Figure 4.4 shows the two graphs and we can see that the algorithm correctly identifies the optimal paths for each destination. However, this is of very limited use, since the values assigned are 0 for the optimal path and a negative number

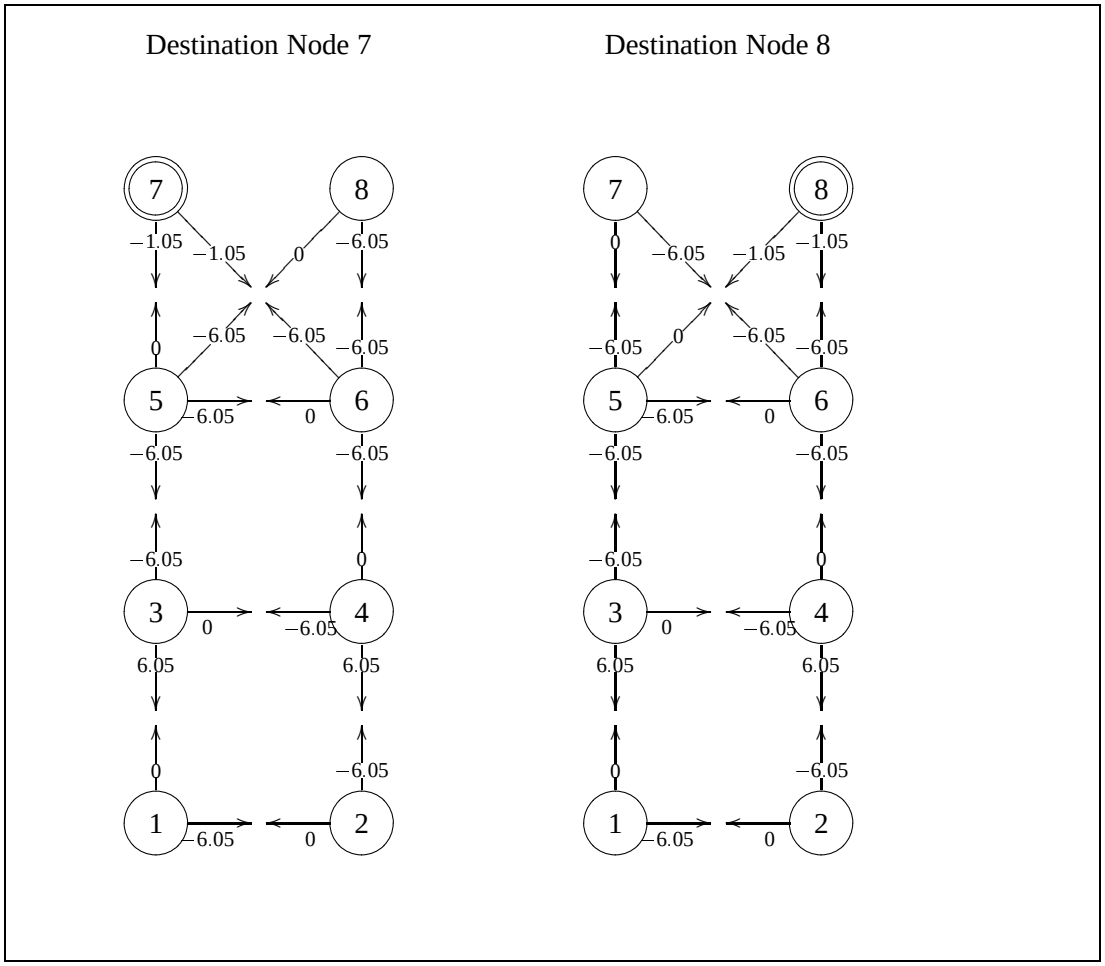


Figure 4.4: Results of IRL Algorithm with Example Network

for non-optimal links. Notice that the negative values can really be assigned arbitrarily. This is not a fault of the IRL algorithm though, since it has no information that would allow it to assign values for suboptimal links because it has no way of rating one suboptimal path against another.

4.2.2 IRL to Determine the Composite Metric

We will now show how to use the IRL techniques to perform the more useful task of discovering the relationships between the components of a composite metric. The IRL formulation sets up an optimization problem that can be solved using linear programming. Given a finite MDP, the optimization problem becomes

$$\begin{aligned} \text{maximize} \quad & \sum_{s\in S}\sum_{a\in A\setminus a_1} Q^{\pi}(s,a_1) - Q^{\pi}(s,a) \\ \text{such that} \quad & (\mathbf{P}_{a_1} - \mathbf{P}_a)(\mathbf{I} - \gamma\mathbf{P}_{a_1})^{-1}\mathbf{R} \geq 0, \forall a \in A \setminus a_1 \end{aligned}$$

Using the vector notation, this can be written as

$$\begin{aligned} \text{maximize} \quad & \sum_{i=1}^{N}\sum_{j=2}^{k}\{\mathbf{P}_{a_1}(i) - \mathbf{P}_{a_j}(i))(\mathbf{I} - \gamma\mathbf{P}_{a_1})^{-1}\mathbf{R}\} - \lambda\|\mathbf{R}\|_1 \\ \text{such that} \quad & (\mathbf{P}_{a_1} - \mathbf{P}_a)(\mathbf{I} - \gamma\mathbf{P}_{a_1})^{-1}\mathbf{R} \geq 0, \forall a \in A \setminus a_1, \\ & |\mathbf{R}_i| \leq R_{max}, i = 1, \ldots, N \end{aligned}$$

The structure of the reward function $R(s)$ is known and can be written as the linear combination $R(s) = \alpha_1 R_1(s) + \alpha_2 R_2(s) + \ldots \alpha_m R_m(s)$ where R_i are the known components of the reward function and α_i are the unknown coefficients that need to be learned. Note that the values of R_i are determined by the state s, while the coefficients α_i remain constant. This function R may be used to model any composite metric that is determined by a linear combination of the attributes of a physical link.

If restricted to metrics that use bandwidth (BW) and latency (Lat), the metric for a link l_i is a function $m(l_i) = \alpha_1 BW(l_i) + \alpha_2 Lat(l_i)$. Note that although when calculating the metric m, α_i can be arbitrary, they can always be scaled such that $\sum \alpha_i = 1$. Then α_is will represent the percentage of the contribution each component of the metric makes to the total value. Now, the metric m can be used as a reward function R. Since the state s has in it implicitly the link that was used to get to it, reward received at state s may be modeled as $R(s) = \alpha_1 BW(l_i) + \alpha_2 Lat(l_i)$ where l_i was the link that was used to get to s. Thus in this case, $R_1(s) = BW(l_i)$ and $R_2(s) = Lat(l_i)$. If the vector notation for functions is used, $R(s)$ can be written as

$$\begin{bmatrix} \mathbf{R}_{1_1} & \mathbf{R}_{1_2} \\ \mathbf{R}_{2_1} & \mathbf{R}_{2_2} \\ \vdots & \vdots \\ \mathbf{R}_{n_1} & \mathbf{R}_{n_2} \end{bmatrix} \begin{bmatrix} \alpha_1 \\ \alpha_2 \end{bmatrix} = \mathbf{R}$$

Now, the optimization problem can be written as

$$\begin{aligned} \text{maximize} \quad & \sum_{i=1}^{N}\sum_{j=2}^{k}\{\mathbf{P}_{a_1}(i) - \mathbf{P}_{a_j}(i))(\mathbf{I} - \gamma\mathbf{P}_{a_1})^{-1} \begin{bmatrix} \mathbf{R}_{1_1} & \mathbf{R}_{1_2} \\ \mathbf{R}_{2_1} & \mathbf{R}_{2_2} \\ \vdots & \vdots \\ \mathbf{R}_{n_1} & \mathbf{R}_{n_2} \end{bmatrix} \begin{bmatrix} \alpha_1 \\ \alpha_2 \end{bmatrix}\} \\ & -\lambda\| \begin{bmatrix} \mathbf{R}_{1_1} & \mathbf{R}_{1_2} \\ \mathbf{R}_{2_1} & \mathbf{R}_{2_2} \\ \vdots & \vdots \\ \mathbf{R}_{n_1} & \mathbf{R}_{n_2} \end{bmatrix} \begin{bmatrix} \alpha_1 \\ \alpha_2 \end{bmatrix} \|_1 \end{aligned}$$

$$\text{such that} \quad (\mathbf{P}_{a_1} - \mathbf{P}_a)(\mathbf{I} - \gamma \mathbf{P}_{a_1})^{-1} \begin{bmatrix} \mathbf{R}_{1_1} & \mathbf{R}_{1_2} \\ \mathbf{R}_{2_1} & \mathbf{R}_{2_2} \\ \vdots & \vdots \\ \mathbf{R}_{n_1} & \mathbf{R}_{n_2} \end{bmatrix} \begin{bmatrix} \alpha_1 \\ \alpha_2 \end{bmatrix} \geq 0, \forall a \in A \setminus a_1,$$

$$\alpha_1 + \alpha_2 = 1$$

Note that the bounding condition $|\mathbf{R}_i| \leq R_{max}, i = 1, \ldots, N$ is no longer necessary since it has been replaced with $\alpha_1 + \alpha_2 = 1$.

This formulation is a novel view of inverse reinforcement learning. The components of the rewards are given, and we are finding their respective contributions, as opposed to finding the rewards themselves.

Chapter 5

Experimental Results

In this chapter we will present the experiments conducted and results gathered. First, we will present the procedure used to set up the experiments. The experiments will be set up to model a situation that might occur in practice. We will then present the results from the experiments.

5.1 Procedure

We now go over the steps to run a particular metric experiment.

1. First the network that will be used is generated by BRITE. We employ a random network topology generator to conduct our experiments. The particular network generator is BRITE [9] developed by a group at the Computer Science Department at Boston University. BRITE is capable of generating a various large of different topologies of very large sizes. There are parameters for the distribution of nodes and links, as well as the characteristics of the links. BRITE can provide bandwidth and latency attributes for each link. When describing the procedure, we will discuss how BRITE will be used to generate the topology. A configuration file, such as the one in Figure 5.1 is used to provide BRITE with the specifications for the network.

   Using BRITE with this input file will create a BRITE output file that contains a randomly generated topology. Figure 5.2 contains an excerpt from a sample output file. The output file is structured into three parts. The first part is two lines that contain information about the file, such as the number of nodes and edges. The next part includes information about nodes, which is not relevant to this procedure. The next part contains information about the links, which includes bandwidth and latency. Latency is the fifth parameter on a line, and bandwidth follows it. For example in Figure 5.2, the bandwidth for the first link is 46.037586 and the latency is 4.891037. The bandwidth is interpreted as megabits/second and the latency is in milliseconds.

2. The topology is generated from the output file. The topology is a three dimensional array. The first two dimensions are the matrix representation of a graph (both dimensions indexed by nodes), the third is the two values for a link (bandwidth and latency) between two routes. This can be viewed as two matrices, one for bandwidth and one for latency.

```
#This config file was generated by the GUI.

BriteConfig

BeginModel
Name =  1  #Router Waxman=2, AS Waxman =3
N = 20  #Number of nodes in graph
HS = 10000  #Size of main plane (number of squares)
LS = 1000  #Size of inner planes (number of squares)
NodePlacement = 2  #Random = 1, Heavy Tailed = 2
GrowthType = 1  #Incremental = 1, All = 2
m = 2  #Number of neighboring node each new node connects to.
alpha = 0.15  #Waxman Parameter
beta = 0.2  #Waxman Parameter
BWDist = 3  #Constant = 1, Uniform =2, HeavyTailed = 3, Exponential =4
BWMin = 10.0
BWMax = 1024.0
EndModel

BeginOutput
BRITE = 1   #1=output in BRITE format, 0=do not output in BRITE format
OTTER = 0   #1=Enable visualization in otter, 0=no visualization
EndOutput
```

Figure 5.1: Example BRITE Input File

```
Topology: ( 20 Nodes, 40 Edges )
Model (1 - RTWaxman):  20 10000 1000 2  2  0.15 0.2 1 3 10.0 1024.0

Nodes: ( 20 )
0 127 168 2 2 -1 RT_NONE
1 888 329 2 2 -1 RT_NONE
2 725 563 3 3 -1 RT_NONE
3 397 320 7 7 -1 RT_NONE
4 1447 372 6 6 -1 RT_NONE
5 2627 677 3 3 -1 RT_NONE
.
.
.
19 6325 186 8 8 -1 RT_NONE


Edges: ( 40 )
0 19 15 1466.296 4.891037 46.037586 -1 -1 E_RT_BACKBONE U
1 19 4 4881.545 16.283081 11.12164 -1 -1 E_RT_BACKBONE U
2 8 19 3483.5168 11.619761 33.808617 -1 -1 E_RT_BACKBONE U
3 8 15 2171.7966 7.2443337 234.85246 -1 -1 E_RT_BACKBONE U
4 11 8 1085.6892 3.6214695 46.754265 -1 -1 E_RT_BACKBONE U
5 11 4 2478.6792 8.267984 10.590906 -1 -1 E_RT_BACKBONE U
6 16 11 504.7237 1.6835771 19.077143 -1 -1 E_RT_BACKBONE U
7 16 19 2341.129 7.8091655 30.948912 -1 -1 E_RT_BACKBONE U
.
.
.
39 4 16 2623.9885 8.752684 16.476728 -1 -1 E_RT_BACKBONE U
```

Figure 5.2: Example BRITE Output File (Excerpt)

3. The source and destination nodes are chosen at random, such that they are not adjacent. These nodes represent the routers that are used to send and receive packets on the network. In a real network, they would represent routers that are connected to the networks that are under our control.

4. Now we need to find the minimum distances between every pair of nodes. This information is stored in the routing table, which is a three-dimensional array. The first dimension is indexed by nodes, it represents which node a particular instance of the routing table belongs to. The second dimension is also indexed by nodes, represents the destination that a node can send packets to. The third dimension is a pair of values, the first is the cost to the destination and the second is the node that is the previous node before the destination along the optimal path to it. The optimal path can be recovered using the previous node information.

5. Next, each link for each node is assigned a port number with respect to that node. For a given node, the link that leads to the next node along the optimal path is assigned port number 0. Then the rest of the links are assigned port numbers 1 to number of edges for that node. This can be done arbitrarily.

6. The **P** matrices are built. The $\mathbf{P}_{a_i}$ matrices are the state transition matrices. When the agent takes action a_i in state r, the position in row r column q gives the probability of the agent going to state q. For us a_i are the ports. The state of the environment is created from the point of view of a packet being forwarded along from its source to its destination. This state is made up of several parts (this is described in more detail in Chapter 3).

   The first part is the current location of the packet, or the router the packet is at. Second, since the destination of the packet influences which path it takes, the destination should be part of the state as well. However, since currently only one destination is considered at a time, this component of the state is ignored. The third component is the port on which the packet arrived on at the current router. This is necessary because the IRL algorithm assigns rewards to states, not state-action pairs. This allows us to distinguish between optimal and sub-optimal links. This means there are $maxports$ P matrices and each P matrix is $nodes \times maxports$ in dimension.

7. The linear programming problem is set up. Since our reward function R is

$$R(s_l) = \alpha_1 BW(s_l) + \alpha_2 DLY(s_l)$$

   where $BW(s_l)$ and $DLY(s_l)$ are the bandwidth and delay of the link used to get to node s, we can write the reward vector **R** as

$$\begin{bmatrix} BW(1_1) & DLY(1_1) \\ \vdots & \vdots \\ BW(n_k) & DLY(n_k) \end{bmatrix} \begin{bmatrix} \alpha_1 \\ \alpha_2 \end{bmatrix} = \mathbf{R}$$

Thus our optimization problem becomes

maximize

$$\sum_{i=1}^{N}\sum_{j=1}^{k}\{(\mathbf{P}_{a_0}(i)-\mathbf{P}_{a_j}(i))(\mathbf{I}-\gamma\mathbf{P}_{a_0})^{-1}\begin{bmatrix} BW(1_1) & DLY(1_1) \\ \vdots & \vdots \\ BW(n_k) & DLY(n_k)\end{bmatrix}\begin{bmatrix}\alpha_1 \\ \alpha_2\end{bmatrix}\}$$

$$-\lambda\|\begin{bmatrix} BW(1_1) & DLY(1_1) \\ \vdots & \vdots \\ BW(n_k) & DLY(n_k)\end{bmatrix}\begin{bmatrix}\alpha_1 \\ \alpha_2\end{bmatrix}\|_1$$

such that

$$(\mathbf{P}_{a_0}-\mathbf{P}_a)(\mathbf{I}-\gamma\mathbf{P}_{a_0})^{-1}\begin{bmatrix} BW(1_1) & DLY(1_1) \\ \vdots & \vdots \\ BW(n_k) & DLY(n_k)\end{bmatrix}\begin{bmatrix}\alpha_1 \\ \alpha_2\end{bmatrix}\geq 0, \forall a\in A\setminus a_1,$$

$$\alpha_1+\alpha_2=1$$

Thus, we calculate the vector **b**, such that

$$\mathbf{b}=(\sum_{i=1}^{N}\sum_{j=1}^{k}\{(\mathbf{P}_{a_0}(i)-\mathbf{P}_{a_j}(i))(\mathbf{I}-\gamma\mathbf{P}_{a_0})^{-1}-\lambda)\begin{bmatrix} BW(1_1) & DLY(1_1) \\ \vdots & \vdots \\ BW(n_k) & DLY(n_k)\end{bmatrix}$$

So our goal is find the values (α_1,α_2) that maximize

$$\mathbf{b}\begin{bmatrix}\alpha_1 \\ \alpha_2\end{bmatrix}$$

Next we calculate the matrix

$$\mathbf{M}=(\mathbf{P}_{a_0}-\mathbf{P}_a)(\mathbf{I}-\gamma\mathbf{P}_{a_0})^{-1}\begin{bmatrix} BW(1_1) & DLY(1_1) \\ \vdots & \vdots \\ BW(n_k) & DLY(n_k)\end{bmatrix}$$

such that

$$\mathbf{M}\begin{bmatrix}\alpha_1 \\ \alpha_2\end{bmatrix}\geq 0, \forall a \text{ in } A\setminus a_1$$

and add a row to it such that

$$\alpha_1+\alpha_2=1$$

8. Now, the information calculated above is output to a file in `lp_solve` [25] format. An excerpt from such a file is shown if Figure 5.3. After running lp_solve, the output is the values for α_1 and α_2.

```
34568.588299 x1 + 69846.960655 x2;
c1: 0.000000 x1 + 0.000000 x2 <= 0.000000;
c2: 0.000000 x1 + 0.000000 x2 <= 0.000000;
c3: 0.000000 x1 + 0.000000 x2 <= 0.000000;
c4: 0.000000 x1 + 0.000000 x2 <= 0.000000;
c5: 0.000000 x1 + 0.000000 x2 <= 0.000000;
c6: 0.000000 x1 + 0.000000 x2 <= 0.000000;
c7: 0.000000 x1 + 0.000000 x2 <= 0.000000;
c8: -42.085111 x1 + -39.624028 x2 <= 0.000000;
c9: -42.085111 x1 + -39.624028 x2 <= 0.000000;
c10: -42.085111 x1 + -39.624028 x2 <= 0.000000;
c11: -42.085111 x1 + -39.624028 x2 <= 0.000000;
c12: -42.085111 x1 + -39.624028 x2 <= 0.000000;
c13: -42.085111 x1 + -39.624028 x2 <= 0.000000;
c14: -42.085111 x1 + -39.624028 x2 <= 0.000000;
c15: -4.750728 x1 + -6.353558 x2 <= 0.000000;
c16: -4.750728 x1 + -6.353558 x2 <= 0.000000;
c17: -4.750728 x1 + -6.353558 x2 <= 0.000000;
c18: -4.750728 x1 + -6.353558 x2 <= 0.000000;
.
.
.
c882: -61.829256 x1 + -156.002754 x2 <= 0.000000;
c882: 1.0 x1 + 1.0 x2 = 1.0;
```

Figure 5.3: Example `lp_solve` input file

5.2 Experiments

The experiments are meant to model the most probable case in a real network. If this technique is used to analyze the network of an ISP, we would have access to only a few machines at the border to the ISP's network. Here, we assume that we only have two machines, one we will use as a source and another as a destination. We will randomly pick non-adjacent nodes on the network and perform our procedure for determining the composite metric. The results for running the procedure several times on a network will represent what is likely to be the outcome in a real situation.

5.2.1 30 Node Networks

For this experiment, we use BRITE to generate 9 random 30 node networks, then pick 10 pairs of routers at random as a source and a destination. We then run the IRL algorithm to set up the linear programming problem, which we solve with `lp_solve`. For each set of 9 networks, we use specific α_1 and α_2's. We ran the experiments with 3 pairs of α_1 and α_2. The values used were $\{0.5, 0.5\}$, $\{0.7, 0.3\}$ and $\{1.0, 0.0\}$. Each trial is a different randomly generated 30 node network, using the BRITE configuration file presented in Figure 5.4.

Table 5.1 contains the averages for α_1 and α_2 for each of the 9 different networks. It is desirable to have the values for the averages to be as close to the actual values of the α's as possible. Most of the values for the discovered α_1's when $\alpha_1 = 0.5$ and $\alpha_2 = 0.5$ are consistently lower than those of the discovered α_2s (except for trials 1 and 7). This may be because the actual values for bandwidths and latencies are distributed according to different distributions (bandwidths are heavy tailed while latencies are linear, as per physical distance between nodes). Thus, it is tough to normalize them such that one unit of bandwidth, such as bit per second is equal to one unit of latency, such as microsecond. These imbalances may affect the perceived percentages of contributions from each of the two components. We observe a similar situation when $\alpha_1 = 0.7$ and $\alpha_2 = 0.3$, where every discovered average α_1 is lower than 0.7, except in trials 2 and 9.

$\alpha_1 = 0.5, \alpha_2 = 0.5$		
Trial	α_1	α_2
1	0.587332	0.412668
2	0.453211	0.546789
3	0.345047	0.654953
4	0.418328	0.581672
5	0.440796	0.559204
6	0.672151	0.327849
7	0.436770	0.563230
8	0.409529	0.590471
9	0.417369	0.582631

$\alpha_1 = 0.7, \alpha_2 = 0.3$		
Trial	α_1	α_2
1	0.657521	0.342479
2	0.721728	0.278272
3	0.624799	0.375201
4	0.617874	0.382126
5	0.646169	0.353831
6	0.589311	0.410689
7	0.659454	0.340546
8	0.752809	0.247191
9	0.589177	0.410823

$\alpha_1 = 1.0, \alpha_2 = 0.0$		
Trial	α_1	α_2
1	0.952694	0.047306
2	1.000000	0.000000
3	0.946939	0.053061
4	0.967559	0.032441
5	0.952913	0.047086
6	1.000000	0.000000
7	1.000000	0.000000
8	0.969895	0.030105
9	0.959484	0.040516

Table 5.1: Averages of 10 samples from 9 random networks (trials), 30 nodes per network

We can also observe in Table 5.1 that for some trials, such as for trials 2 and 3 when $\alpha_1 = 0.5$ and $\alpha_2 = 0.5$, the results are not impressive, with the averages for α_1 being close to 0.3, or trial 7, where the average is close to 0.7, instead of 0.5. A similar situation exists for networks with $\alpha_1 = 0.7$ and $\alpha_2 = 0.3$. However, we will observe that these results will improve drastically when the size of the network increases from 30 nodes to 40, and then 50.

```
#This config file was generated by the GUI.

BriteConfig

BeginModel
Name =  1  #Router Waxman=2, AS Waxman =3
N = 30  #Number of nodes in graph
HS = 10000  #Size of main plane (number of squares)
LS = 1000  #Size of inner planes (number of squares)
NodePlacement = 2  #Random = 1, Heavy Tailed = 2
GrowthType = 1  #Incremental = 1, All = 2
m = 2  #Number of neighboring node each new node connects to.
alpha = 0.15  #Waxman Parameter
beta = 0.2  #Waxman Parameter
BWDist = 3  #Constant=1, Uniform=2, HeavyTailed=3, Exponential=4
BWMin = 10.0
BWMax = 1024.0
EndModel

BeginOutput
BRITE = 1   #1=output in BRITE format, 0=do not output in BRITE format
OTTER = 0   #1=Enable visualization in otter, 0=no visualization
EndOutput
```

Figure 5.4: Example BRITE Input File

For each of the three sets of networks, we will present boxplots for the discovered values of α_1 (the values for α_2 are symmetrical since $\alpha_2 = 1 - \alpha_1$). Figure 5.5 contains the boxplot for the nine random networks networks routed with α_1 and α_2 set to 0.5. The top and bottom of each of the boxes in the plot represent the first and third quartile values for the ten discovered α's. The error bars represent the adjacent values, or data points that are closest to the points f_1 and f_3 where

$$\begin{aligned} f_1 &= q_1 - 1.5(q_3 - q_1) \\ f_3 &= q_1 + 1.5(q_3 - q_1) \end{aligned}$$

Thus, the middle of each box is centered over the average values from Table 5.1. Figures 5.5, 5.6, and 5.7 contain the plots for the three sets of networks with different α_1 and α_2 values.

From Figures 5.5 and 5.6 we can see that the discovered values for α_1 often miss their goals and their variance can differ greatly. This should be compared to the discovered values in Figures 5.8 and 5.9, when the number of nodes in each network is increased to 40 and an improvement in accuracy can be observed.

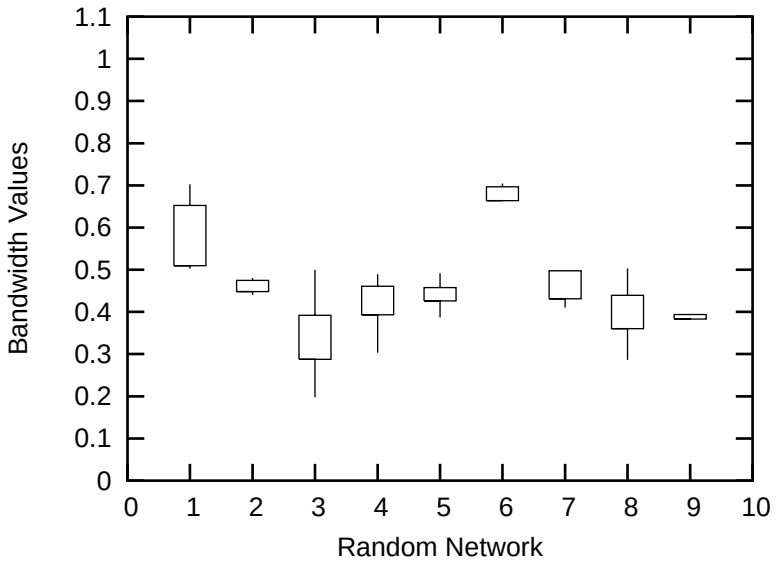


Figure 5.5: Plot of discovered α_1 for 9 random 30 node networks, actual α_1 used was 0.5

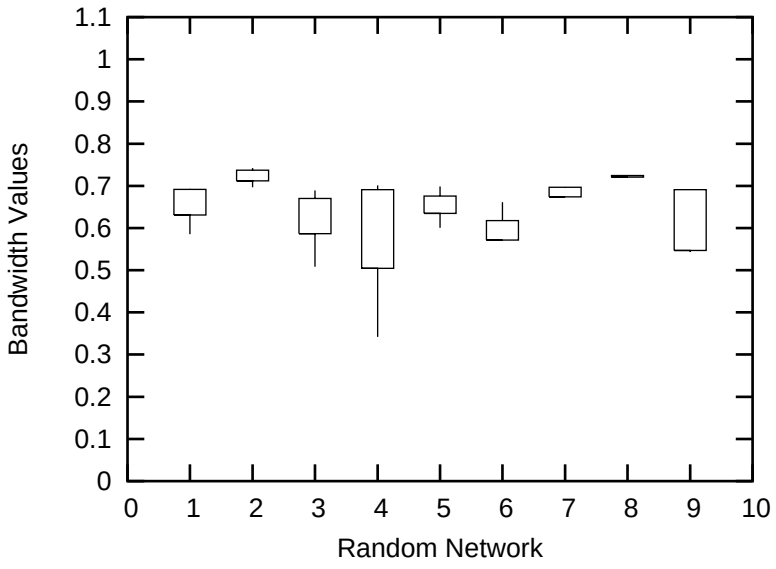


Figure 5.6: Plot of discovered α_1 for 9 random 30 node networks, actual α_1 used was 0.7

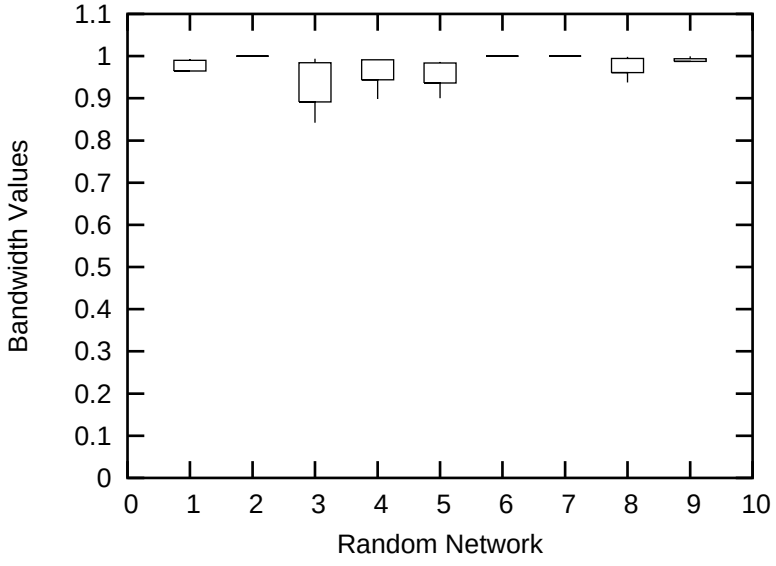


Figure 5.7: Plot of discovered α_1 for 9 random 30 node networks, actual α_1 used was 1.0

5.2.2 40 Node Networks

Results for the networks with 40 nodes were collected in exactly the same way as the results from the previous section. Nine different networks were randomly generated for each pair of α's, and ten random pairs of nodes were chosen as a source and destination for each random network. Table 5.2 contains the results for the average discovered values for the α's for the ten pairs of nodes for each network. Although the values for α_1 are still consistently lower for $\alpha_1 = 0.5$ and $\alpha_1 = 0.7$, they are much closer to the actual values used in the networks. This can also be observed in Figures 5.8 and 5.9, where the boxplots show that the range of the discovered values is smaller and closer to the correct values. This improves further when the number of nodes is increased to 50, as can be seen in the next section.

$\alpha_1 = 0.5, \alpha_2 = 0.5$		
Trial	α_1	α_2
1	0.543630	0.456369
2	0.458908	0.541092
3	0.421309	0.578691
4	0.470915	0.529085
5	0.482138	0.517862
6	0.474102	0.525898
7	0.549825	0.450175
8	0.460848	0.539152
9	0.424842	0.575158

$\alpha_1 = 0.7, \alpha_2 = 0.3$		
Trial	α_1	α_2
1	0.655922	0.344078
2	0.652258	0.347742
3	0.644822	0.355178
4	0.671142	0.328858
5	0.671900	0.328100
6	0.664189	0.335811
7	0.639451	0.360549
8	0.686381	0.313619
9	0.773649	0.226351

$\alpha_1 = 1.0, \alpha_2 = 0.0$		
Trial	α_1	α_2
1	0.975437	0.024564
2	1.000000	0.000000
3	1.000000	0.000000
4	1.000000	0.000000
5	0.985091	0.014909
6	0.967836	0.032164
7	1.000000	0.000000
8	0.992320	0.007680
9	0.980055	0.019945

Table 5.2: Averages of 10 samples from 9 random networks (trials), 40 nodes per network

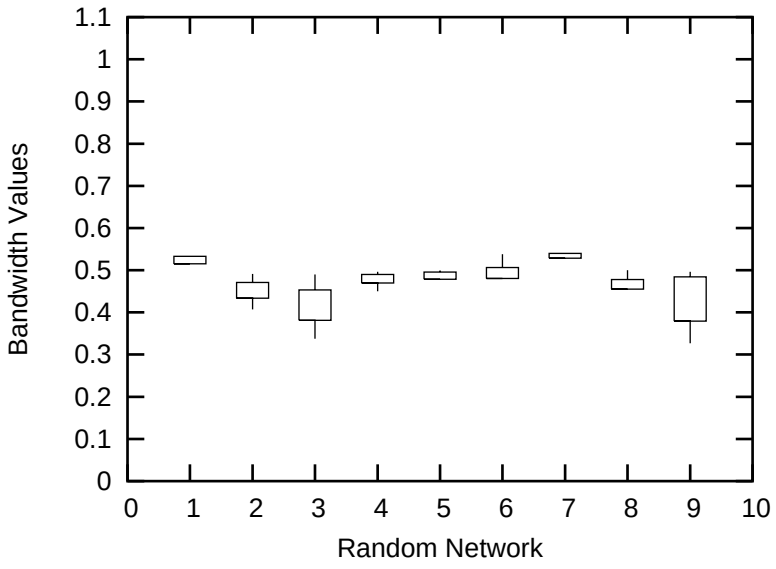


Figure 5.8: Plot of discovered α_1 for 9 random 40 node networks, actual α_1 used was 0.5

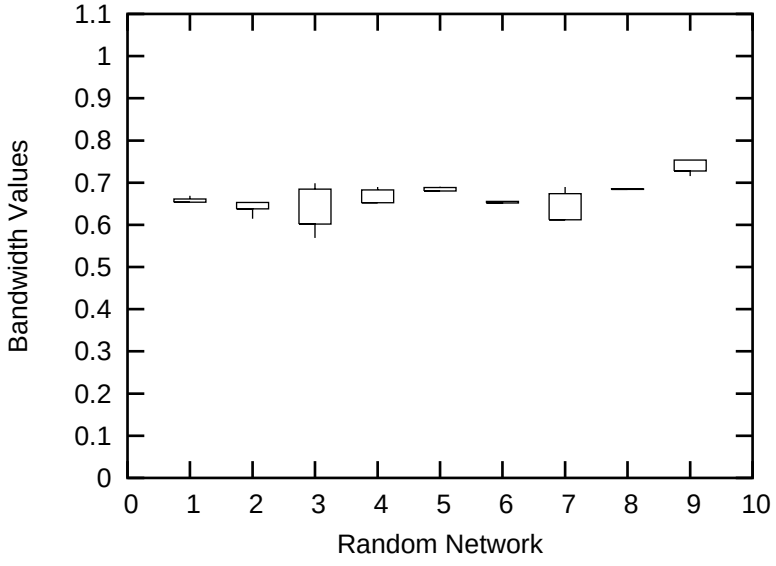


Figure 5.9: Plot of discovered α_1 for 9 random 40 node networks, actual α_1 used was 0.7

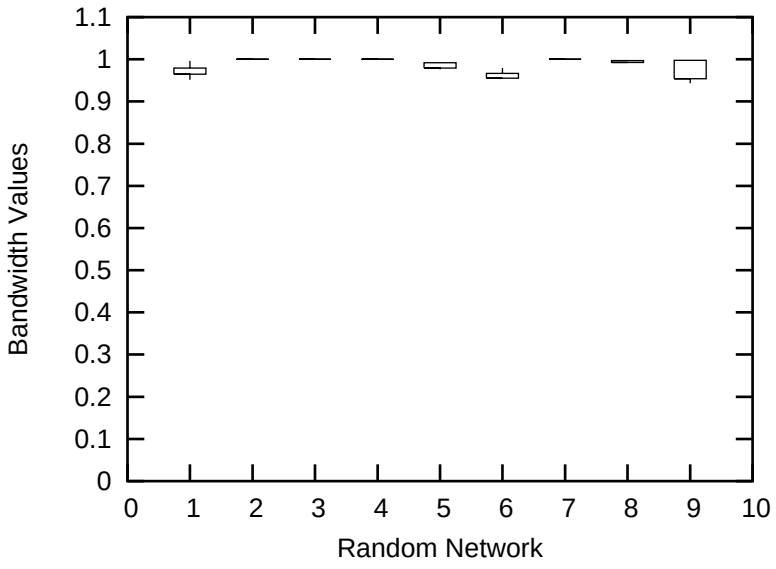


Figure 5.10: Plot of discovered α_1 for 9 random 40 node networks, actual α_1 used was 1.0

5.2.3 50 Node Networks

Here we present the results collected from networks of 50 nodes, set up exactly like the previous two sections. The improvement in the values for the α_1 and α_2 is not as drastic as between networks of 30 nodes and 40 nodes, and the results are comparable to those of the 40 node networks. As can be seen in Table 5.3, the averages of the discovered values are close to the actual values. Figures 5.11, 5.12, and 5.13 show that the ranges over which the values vary are very small.

5.3 Discussion

Our results show that given perfect knowledge of the network, our method is able to reliably determine the values of the coefficients (α_i) of two metric components that are linearly combined to form a composite metric. The random networks used were generated such that they represent actual autonomous systems on the Internet in terms of router and link distributions. The reliability of our method increases with the number of routers present in the autonomous system.

$\alpha_1 = 0.5, \alpha_2 = 0.5$		
Trial	α_1	α_2
1	0.478874	0.521126
2	0.468416	0.531584
3	0.457299	0.542701
4	0.466051	0.533949
5	0.478386	0.521614
6	0.454899	0.545101
7	0.533983	0.466017
8	0.538486	0.461514
9	0.469512	0.530488

$\alpha_1 = 0.7, \alpha_2 = 0.3$		
Trial	α_1	α_2
1	0.666068	0.333932
2	0.624115	0.375885
3	0.660928	0.339072
4	0.669098	0.330902
5	0.680861	0.319139
6	0.666791	0.333209
7	0.688993	0.311006
8	0.682842	0.317158
9	0.662795	0.337205

$\alpha_1 = 1.0, \alpha_2 = 0.0$		
Trial	α_1	α_2
1	0.975020	0.024980
2	0.991583	0.008417
3	0.982289	0.017711
4	1.000000	0.000000
5	0.996175	0.003825
6	0.977311	0.022689
7	0.999988	0.000012
8	1.000000	0.000000
9	0.988685	0.011315

Table 5.3: Averages of 10 samples from 9 random networks (trials), 50 nodes per network

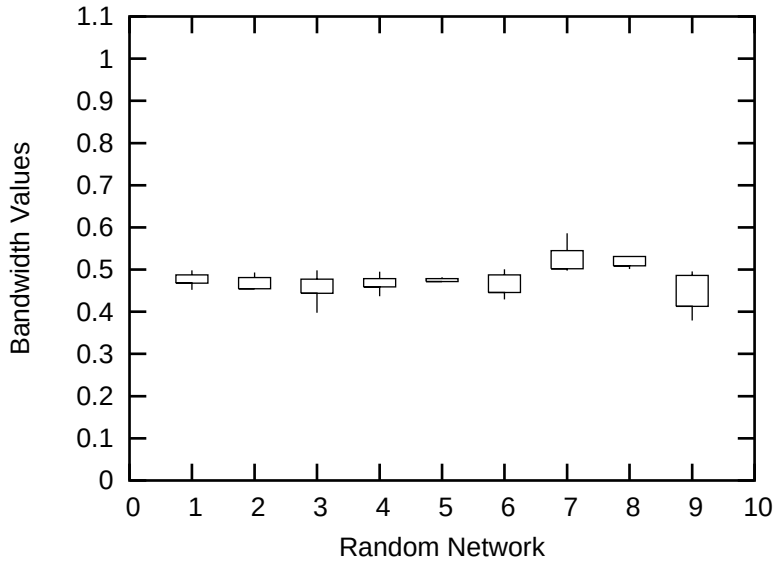


Figure 5.11: Plot of discovered α_1 for 9 random 50 node networks, actual α_1 used was 0.5

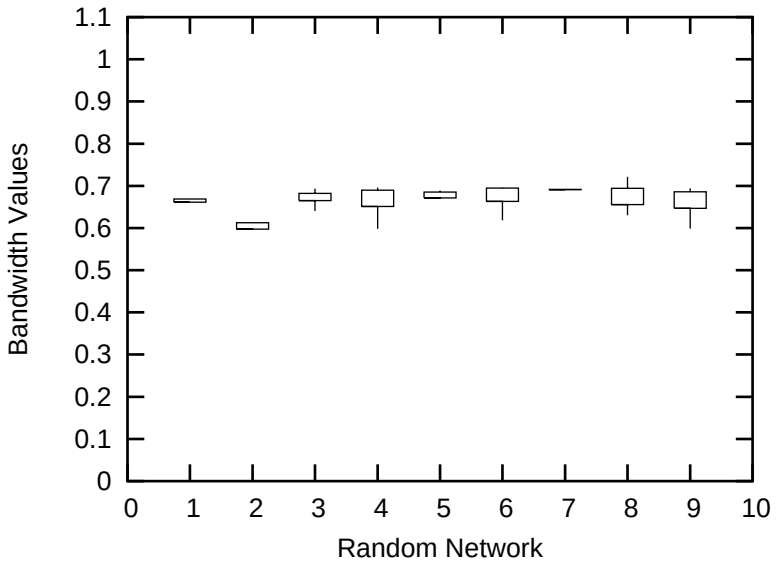


Figure 5.12: Plot of discovered α_1 for 9 random 50 node networks, actual α_1 used was 0.7

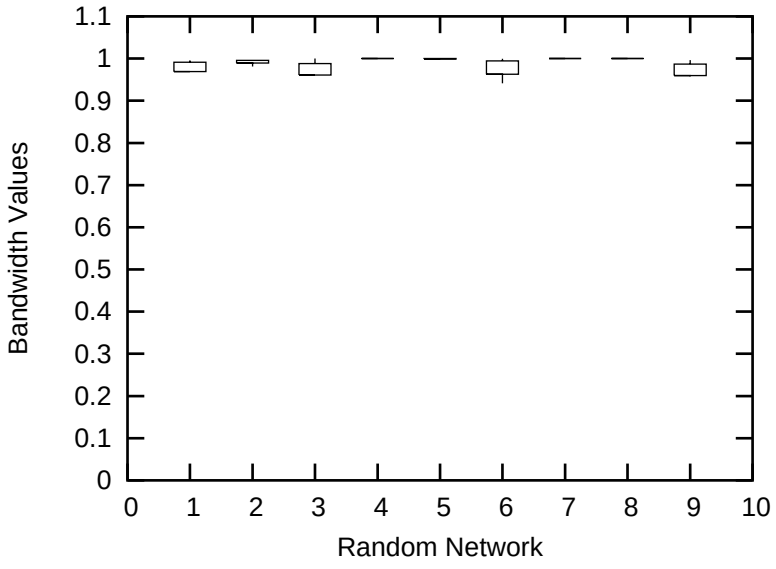


Figure 5.13: Plot of discovered α_1 for 9 random 50 node networks, actual α_1 used was 1.0

Chapter 6

Conclusion and Future Work

6.1 Conclusion

We have presented a method for using IRL techniques to discover the details of a linear composite metric used by a routing protocol. This method includes a specific way of modeling a network and the routing problem as a finite Markov decision process. We have also set up a reinforcement learning agent such that, when it operates optimally, we may recover the linear composite metric it is optimizing. We accomplish this by utilizing inverse reinforcement learning techniques presented by Ng and Russell. We presented results collected when our technique was applied to randomly generated networks (that simulate a real Internet autonomous system).

If applied in practice, our method has the potential of allowing Internet service providers to analyze the networks of their competitors and discover the metrics they use.

6.2 Future Work

- **Non-Linear Composite Metrics**
  We can currently only discover linear composite metrics. This requirement eliminates metrics that multiply their components together, or perform another nonlinear operation, such as taking the minimum of a metric along a path. CISCO's IGRP and EIGRP metrics employ both strategies. Thus, that metric cannot be represented as a reward vector. Further research is necessary to determine if IRL can be used to recover such non-linear metrics.

- **Partially Known Environments**
  Our method for discovering the details of a composite metric relies on accurate knowledge of the network we wish to analyze. This may not be always possible in a real network, thus developing a method that can be used with partially known environments may be better applicable to real life situations.

- **Network and Link Discovery**
  Since knowing the network topology and link attributes is important for our techniques,

new tools and methods for discovering them can be used to improve the results of our IRL approach.

Bibliography

[1] A.Y. Ng and S. Russell, "Algorithms for Inverse Reinforcement Learning." *International Conference on Machine Learning,* pp.663–670, Stanford, California: Morgan Kaufmann, 2000.

[2] J. A. Boyan and M. L. Littman. "Packet Routing in Dynamically Changing Networks: A Reinforcement Learning Approach." *In J. D. Cowan, G. Tesauro, and J. Alspcctor, editors, Advances in Neural Information Processing Systems 6,* pp.671–678, Morgan Kaufmann Publishers, 1994.

[3] D. Subramanian, P. Druschel, and J. Chen. "Ants and Reinforcement Learning: A Case Study in Routing in Dynamic Networks." *In Proceedings of IJCAI'97, International Joint Conference on Artifi cial Intelligence,* pp.832–838, 1997.

[4] A.Y. Ng, D. Harada, and S. Russell, "Policy invarience under reward transformations: Theory and application to reward shaping," *In Proceedings of the Sixteenth International Conference on Machine Learning. Bled, Slovenia: Morgan Kaufmann,* pp. 278–287, 1999.

[5] R. Siamwalla, R. Sharma, and S. Keshav, "Discovering Internet Topology." *Submitted to Infocom '99*

[6] R. L. Carter and M. E. Crovella, "Measuring Bottleneck Link Speed in Packet-Switched Networks." *Performance Evaluation,* vol 27 and 28, pp.297–318, 1996.

[7] U. Chajewska, D. Koller, D. Ormoneit, "Learning an Agent's Utility Function by Observing Behavior," *Eighteenth International Conference on Machine Learning (ICML), Williams College,* June 2001.

[8] S. Varadarajan, and N. Ramakrishnan, "Reinforcing Reachable Routes," to appear in *Computer Networks,* 2003.

[9] A. Medina, A. Lakhina, I. Matta, J. Byers, "BRITE: Universal Topology Generation from a User's Perspective." *Technical Report, BUCS-TR2001-003,* Boston University, 2001.

[10] A. B. Downey, "Using pathchar to estimate Internet link characteristics." *In Proceedings of ACM SIGCOMM '99, Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications,* pp. 241–250, Sept. 1999.

[11] V. Jacobson, "pathchar – a tool to infer characteristics of Internet paths." *Presented at the Mathematical Sciences Research Institute (MSRI);* slides avaialbe from ftp://ftp.ee.lbl.gov/pathchar/, April 1997.

[12] U. Manber, "Introduction to Algorithms," *Addison-Wesley Publishing Company Inc.,* 1989.

[13] J. Doyle, "CCIE Professional Development: Routing TCP/IP," *Macmillian Technical Publishing,* 1998.

[14] R. S. Sutton and A. G. Barto, "Reinforcement Learning" *The MIT Press,* 2002.

[15] L. P. Kaelbling and M. L. Littman, "Reinforcement Learning: A Survey," *Journal of Artificial Intelligence Research,* vol 4, pp. 237–285, 1996.

[16] V. Jacobson, "traceroute," *ftp://ftp.ee.lbl.gov/traceroute.tar.Z.*

[17] J. Postel, "Internet Control Message Protocol," *RFC777, Internet Engineering Task Force,* 1981.

[18] J. Moy, "OSPF" *RFC1131, Internet Engineering Task Force,* 1989.

[19] J. Moy, "OSPF Version 2" *RFC1247, Internet Engineering Task Force,* 1991.

[20] J. Moy, "OSPF Version 2" *RFC2328, Internet Engineering Task Force,* 1998.

[21] J. A. Dossey, A. D. Otto, L. E. Spence, and C. V. Eynden, "Discrete Mathematics" *Addison-Wesley,* 1997.

[22] V. Irwin, and H. Pomeranz, "Intrusion Detection and Packet Filtering: How it Works," *course notes, authors can be reached at virwin@texas.net, and hal@deer-run.com*

[23] G. Cybenko, "Dynamic Programming: A Discrete Calculus of Variations," *IEEE Computational Science and Engineering,* vol 4, pp. 92-97, 1997.

[24] S. I. Gass, "Linear Programming: Methods and Applications," *McGraw-Hill Inc.,* 1964.

[25] M. Berkelaar, "lp_solve," *ftp://ftp.ics.ele.tue.nl/pub/lp_solve/*

Vita

Dmitry Eric Shiraev was born on February 3rd, 1980 in St. Petersburg, Russia (formerly Leningrad, Soviet Union). He enrolled as an undergraduate at Virginia Tech in the Fall of 1998. Dmitry participated in the 5 year BS/MS program, taking graduate classes during his senior year of his undergraduate studies. Dmitry graduated Magna Cum Laude with Bachelor of Science degrees in Computer Science and Mathematics in May 2002. His Mathematics concentration was in Applied Discrete Mathematics. He completed the requirements for Master of Science in Computer Science in August 2003. Dmitry is a member of Phi Beta Kappa, Pi Mu Epsilon, and Upsilon Pi Epsilon.