

Walking the Path

**An Application of Policy-Gradient Reinforcement Learning
to Network Routing.**

Nigel Chi-Yen Tao

A thesis submitted in partial fulfilment of
the requirements of the Bachelor of Science degree, with Honours, at
the Computer Science Department of the Australian National University.

April 2001

© 2000 Nigel Chi-Yen Tao

Typeset in Times by $\text{T}_\text{E}\text{X}$ and $\text{L}^\text{A}\text{T}_\text{E}\text{X} 2_\epsilon$.

Except where otherwise indicated, this thesis is my own original work.

Nigel Chi-Yen Tao
12 April 2001

To Turps, Srema and Hoges.

Acknowledgments

We could have pilots undergo a crash course of three months.

— Thomas Drake-Brockman, Australian Minister for Air

There have been many people who have helped me out throughout the year, and I hope I haven't forgotten any of them.

First of all, I would like to thank Honors Convenor Superstar Jyme Grundy, for being damn cool, and for his 'Gator skateboard. I would also like to thank Brendan McKay, for leaving me an image of him surrounded by bikini-clad women on some Mediterranean beach.

Thanks to Jon Baxter for coming up with such an interesting topic, and for bringing my attention the tourist mecca of Pittsburgh, Pennsylvania. Cheers also to Lex Weaver for taking the time to share many supervisor meetings in his freshly renovated office¹.

I'd like to acknowledge the organizations who have kindly given financial support for my study — without them, I would be slaving at a consulting firm to pay the bills. Thanks go to the Australian National University for its excellent National Undergraduate Scholarship scheme, and Burgmann College for its Rob Swift Honours Scholarship.

Thanks to all the chonz, none of whom I knew 12 months ago, for being so awesome, and for the K'nex™ dude. Trystan “Don't Spell My Name Wrong Again Or I'll Kill You” Upstill, for sharing the corporate interview experience. Derek “Sleet-Slammin” Foster, for the couch to crash on. Vijay “Gourmet Chef” Boyapati, for his unique way with words. Mick “Captain Orthogonal” Compton, for changing my views on the role of women in society. Matt “You Can't Thank ‘Ephemeral’ In Your Acknowledgements” Adcock, for all the SIGGRAPH toys. Tom “Tentacle” Kaiser, for all those weird muffins. Mel “Jedi” Smith, for literally kicking arse. Pete “Pete” O'Brien, for being Pete. Finally, Jason “Honorary Chonz” Ozolins, for being a 1337 h4X0r and knowing everything.

Cheers to Rampaging Roy Slaven, and H. G. Nelson, for the many laughs whilst I was going nuts programming all day, and for introducing Fatso — the Battler's Prince. Thanks to Wrigleys for their bonza spearmint chewing gum, to Zefferelli's, for feeding the Zoo animals, and to Toohey's, for the Quiet Ale. Thanks also to the Calvary

¹ANU Union Bar

emergency ward, twice, for all my football accidents. To the guy (or girl) who invented nerf, thanks to you too. To John Carmack (not Romero) for the excellent *Quack3*, I raise my glass.

Yet another round of thanks, this time to my family, for all the support through my life. Special thanks to Mum, for being the best Mum I've ever had, and to my brother Terry, for the valuable discussions, and the proof that never made it.

A barrel of thanks to Melanie "Pa Tu" Calvert, of the under-appreciated Chonz' Wives Club, for a lift home in the rain, a place to stay, the last-minute thesis swap, and about a million other things. I'm already looking forward to Black Russians and soup when our theses are done.

Finally, thanks to you, for reading this thesis. This has been the biggest thing I've ever done, and it's nice to know that someone out there takes an interest.

Twas the night before thesis, when all through CS,
Not a keyboard was stirring, not even a mouse.
The toner was hung by the printer with care,
In hopes that 2 p.m. soon would be there.
The chonz were nestled, all snug in their beds,
While visions of Jenna J. danced in their heads.
The brand new computers, all tired from Quake,
Had just settled down for a long summer break,
When out of the Zoo there arose such a clatter,
I sprang from the couch to see what was the matter...

Power surge - all data on mehta has been deleted.
Press any key to continue.

10 hours to go...

— N. Tao, 4 a.m., November 24, 2000.

Abstract

The world is a book, and those who do not travel, read only a page.

— Saint Augustine

Reinforcement learning is a general technique which relies on a numerical performance measure as feedback for machine learning. Policy-gradient reinforcement learning incorporates gradient ascent in policy-parameter space to achieve a locally optimum solution from a class of policies.

This thesis details the application and evaluation of OLPOMDP, a recent policy-gradient reinforcement learning algorithm, to simulated network routing. It was found that, in most circumstances, OLPOMDP performed comparably to the benchmark Shortest Path algorithm. It was also found to be more robust, outperforming Shortest Path in some cases when the underlying network model was modified. OLPOMDP routing also avoided Braess' Paradox, which confounds certain greedy algorithms.

Policy-gradient methods were demonstrated to successfully solve a distributed problem, which involved multiple agents having to co-ordinate their actions to reach a globally optimal collective policy. This was achieved without an explicit credit assignment algorithm, or even communication between agents, except for contributions to the numerical reward signal.

Certain patterns of sub-optimal behavior were detectable, and an explicit penalty term (i.e. negative reward signal) for such behavior was shown to dramatically speed up convergence time. This motivates an extension of the Actor-Critic reinforcement learning architecture to incorporate domain knowledge into the training signal.

Contents

Acknowledgments	vii
Abstract	ix
1 Introduction	1
1.1 Purpose	1
1.2 Background Overview	1
1.2.1 Network Routing	1
1.2.2 Machine Learning	2
1.2.3 Reinforcement Learning	2
1.2.4 Policy-Gradient Reinforcement Learning	2
1.2.5 The OLPOMDP Algorithm	4
1.2.6 A New Routing Paradigm	4
1.3 Thesis Organization	7
2 Related Work	9
2.1 Network Routing	9
2.1.1 Problems in Network Routing	10
2.1.2 Alternative Approaches to Network Routing	11
2.2 Co-ordination of Multiple Agents	12
2.2.1 Credit Assignment	14
2.3 Reinforcement Learning	14
2.3.1 Reward Shaping	15
2.3.2 Reinforcement Learning Applied to Routing	17
2.4 Gradient Ascent Methods	17
2.4.1 Policy-Gradient Reinforcement Learning	20
3 Motivation	21
3.1 Why PGRL could be Appropriate for NR	21
3.1.1 Robustness	21
3.1.2 Guaranteed Performance Improvement	21
3.1.3 Implicit Co-ordination	22
3.1.4 Distributable Computation	22
3.1.5 Flexibility	22
3.2 Why PGRL could be Inappropriate for NR	23

3.2.1	Convergence Time	23
3.2.2	Multiple Local Optima	23
3.2.3	Non-Determinism	23
3.2.4	Scalability	24
3.2.5	Reward Signal Distribution	24
3.3	Summary	24
4	Design	25
4.1	Simulation Design	25
4.2	Design of a PGRL Routing Algorithm	26
4.2.1	The Reinforcement Learning Framework	26
4.2.2	The Policy-Gradient Reinforcement Learning Framework	26
4.2.3	The n -Armed Bandit Analogy	28
4.2.4	The OLPOMDP Routing Algorithm	29
4.2.5	Reward Signal Design	30
4.3	Heuristic Optimizations	30
4.3.1	Restricting the Range of θ	30
4.3.2	Packet Time-To-Live	32
4.3.3	Reward Signal Decomposition	33
4.3.4	Cycle Detection	34
5	Results	35
5.1	Simple Networks	36
5.1.1	Convergence to Optimum (the Linear 3-Node Network)	36
5.1.2	Consistency (the Linear 3-Node Network Repeated)	38
5.1.3	Co-ordinating Agents (the Triangular 3-Node Network)	40
5.2	Performance Heuristics	42
5.2.1	Signal Decomposition (the 5-Node Network)	42
5.2.2	Cycle Detection (two 6-Node Networks)	46
5.2.3	Scalability (the 8-Node Network)	50
5.3	Resource Contention and Side-Effects	52
5.3.1	Mixed Strategies (the 2-Node Network)	52
5.3.2	Multiple Optima (the Triangle Network Revisited)	54
5.3.3	Braess' Network Paradox	56
5.4	Distributed Networks	59
5.4.1	Broadcast of Reward Signal Components	59
6	Discussion	61
6.1	Was PGRL Appropriate for NR?	61
6.1.1	Robustness	61
6.1.2	Guaranteed Performance Improvement	61
6.1.3	Implicit Co-ordination	62

6.1.4	Distributable Computation	62
6.1.5	Flexibility	62
6.2	Was PGRL Inappropriate for NR?	62
6.2.1	Convergence Time	62
6.2.2	Multiple Local Optima	63
6.2.3	Non-Determinism	65
6.2.4	Scalability	65
6.2.5	Reward Signal Distribution	66
6.3	Implications	66
6.3.1	An Architecture for Reward Shaping	66
7	Conclusion	71
7.1	Specific Contributions	71
7.2	Future Work	72
7.2.1	Other Application Domains	73
A	Verification	75
A.1	Theory	75
A.2	Practice	77
B	The Moving Average Smoother	81
C	Existence of Multiple Optima	83
	Bibliography	85
	Glossary	89
	Notation	93

Introduction

A tower of nine storeys begins with a heap of earth.
The journey of a thousand *li* starts from where one stands.

— Lao Tzu.

Policy-gradient reinforcement learning (PGRL) is a general purpose approach that is applicable to a variety of problems. This thesis presents an evaluation of a recent PGRL algorithm (OLPOMDP¹, developed by Baxter and Bartlett [1999]) applied to the problem of network routing.

Network routing is a well-studied problem, for which numerous algorithms already exist (see, for example, the survey articles by Deo and Pang [1984] and Yang and Reddy [1995]). These algorithms supply an evaluation benchmark for PGRL.

1.1 Purpose

This thesis aims to demonstrate the effectiveness of policy-gradient reinforcement learning, and the OLPOMDP algorithm in particular. This algorithm is very recent, and this thesis presents the largest application of it to date.

It is also hoped that this thesis will encourage the real-world application of Reinforcement Learning techniques outside of its recognized domains of pattern recognition, data classification, game playing, and systems control.

1.2 Background Overview

1.2.1 Network Routing

Network routing is a real-world problem: the growing use of the Internet demonstrates the need for efficient communication methods. Outside of the Internet domain, routing

¹An acronym for On-line Learning algorithm for Partially Observable Markov Decision Processes. A brief explanation is provided by the glossary on page 89, otherwise see chapter 4.

is important for messaging in multi-processor computers and the design of transport networks (such as vehicle traffic, airline scheduling and water distribution).

1.2.2 Machine Learning

Machine learning is the study of computer algorithms which improve automatically with experience [Mitchell 1997]. Traditionally, there are two distinct categories: supervised learning and unsupervised (or reinforcement) learning. Supervised learning relies on the supply of training data, or a set of examples and the corresponding desired actions. For example, in image classification, a human might supply a series of images as well as their correct identification (e.g. “images 21, 36 and 54 are pictures of a hippopotamus”). The goal would be to develop a program that could classify arbitrary images, based on its experience of the (finite) training data set.

1.2.3 Reinforcement Learning

In some situations, a human may not know the right actions to teach a computer agent. For example, consider a chess playing program. Although there exists a perfect strategy, no human has discovered it, and game-tree methods scale poorly with search depth, or the number of moves looked ahead. Nonetheless, it is possible to measure the performance of a program, such as its win-loss ratio in competition. Reinforcement Learning attempts to create effective programs based only on a performance measure for feedback.

The classic reinforcement learning architecture is depicted in figure 1.1. An agent continually observes and acts in an environment, and its actions can trigger (numerical) rewards to be issued. The agent contains a policy, which determines how it responds to its observations. A policy is characterized by a number of (real-valued) parameters (e.g. a neural network has linkage weights), which form a parameter space \mathbf{R}^k . The agent also contains a learning system, which uses the reward signal, or performance measure, to update the policy via a feedback mechanism.

1.2.4 Policy-Gradient Reinforcement Learning

Traditional reinforcement learning is based on state evaluation [Sutton and Barto 1998], which has its basis in dynamic programming [Bellman 1957]. In this model, an agent divides its environment into a set of states, based on the features it chooses to observe. It then learns the value of each state, which is related to the expected reward signal of achieving that state. An agent attempts to stay at high-valued states, which hopefully corresponds to a good policy.

An alternative approach foregoes the intermediate notions of system state and a state evaluation function, and instead learns by gradient ascent on an agent’s policy.

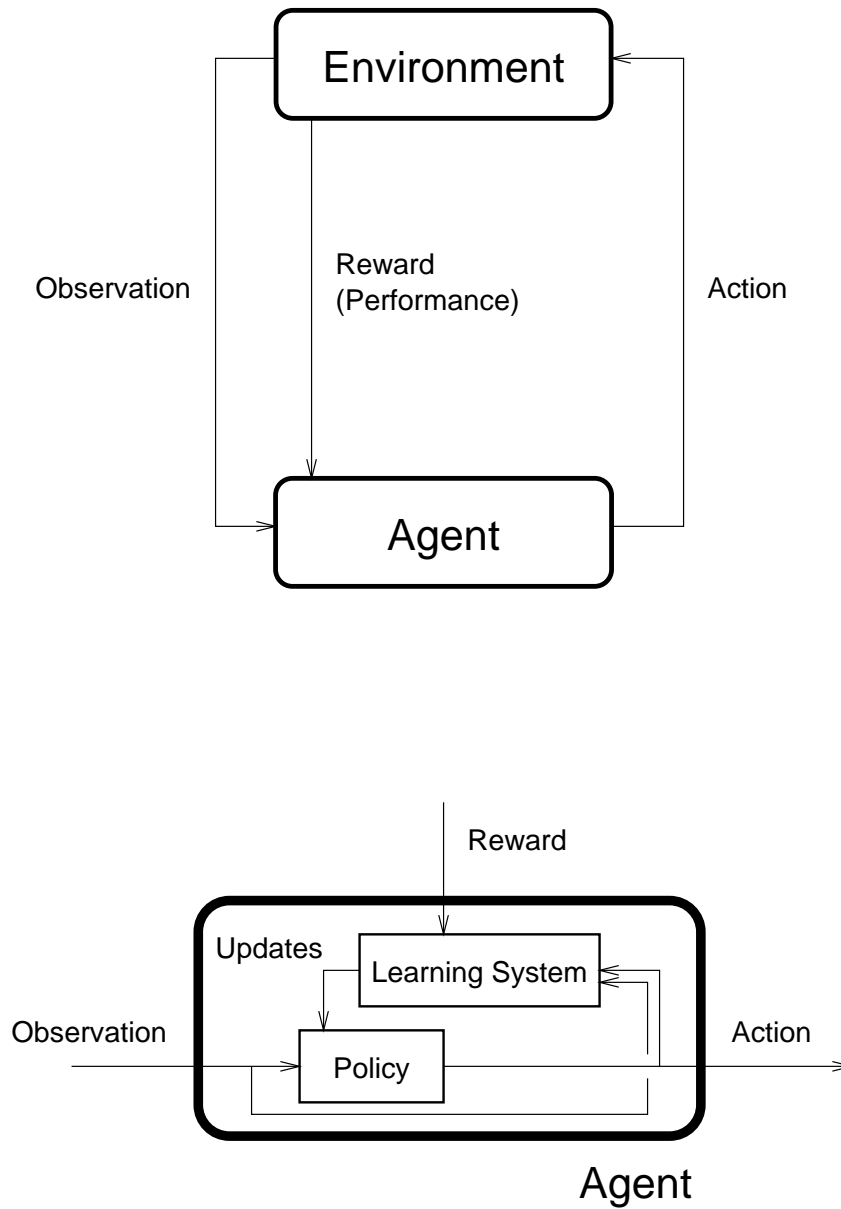


Figure 1.1: The reinforcement learning architecture.

This method is known as policy-gradient, or direct, reinforcement learning. An agent's performance is dependent on its policy, which is dependent on its parameters. Thus, one can assign an expected performance measure to each point in parameter space. Gradient ascent (also known as hill-climbing) methods converge to a local optimum in parameter space. This gives them a (weak) optimality guarantee that other algorithms do not have.

1.2.5 The OLPOMDP Algorithm

The OLPOMDP algorithm is an example of policy-gradient reinforcement learning. It is informally summarized in the following loop:

1. Try actions according to the current policy.
2. Keep a record, known as the eligibility trace \vec{z} , of recent actions.
3. Receive a global reward signal r .
4. Associate recent actions with the reward signal.
5. Adjust policy to favor those actions associated with high rewards.

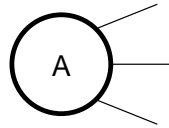
The algorithm has two parameters: β , or how long the memory should be, and γ , or how large the policy updates should be.

The β parameter is to deal with credit assignment. There may be a time delay between an action and its consequences, and so it is difficult to associate the right actions with the right reward signals. If β is too small, then an action which caused a significant improvement in performance may be forgotten, and not recorded as a good action to take. If β is too large, then credit is diluted over a large number of actions, which may inappropriately reward actions performed in the distant past.

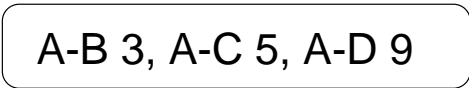
The γ parameter is to deal with noise. A system's performance is not just affected by an agent's actions, but a number of environmental factors, which may include the actions of other agents. If γ is too large, then the direction of policy changes may be determined by mere chance rather than by steady observation. If γ is too small, then it will take a long time for an agent to adapt to changes in the environment, or to learn from its initial state.

1.2.6 A New Routing Paradigm

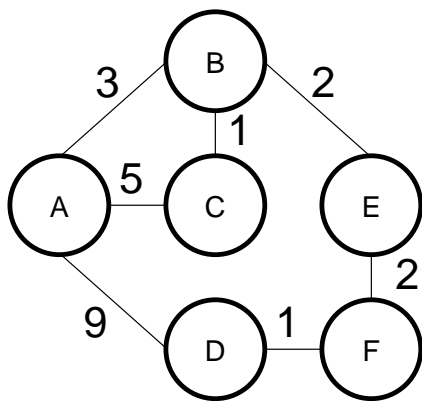
Traditional routing algorithms are based on the construction of a snapshot, or model, of the network. A shortest-path algorithm is run on this model to determine routes. This procedure is illustrated in figure 1.2.



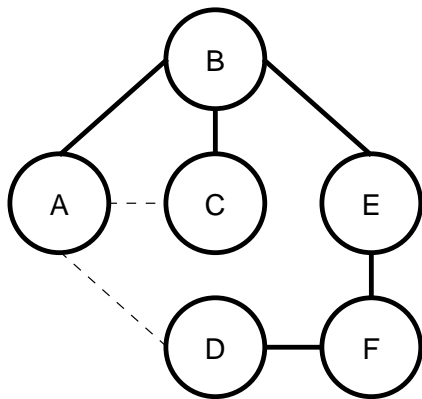
Query links



Collate information and broadcast



Construct 'snapshot' of the network



Find shortest paths with respect to model

Figure 1.2: The traditional routing paradigm.

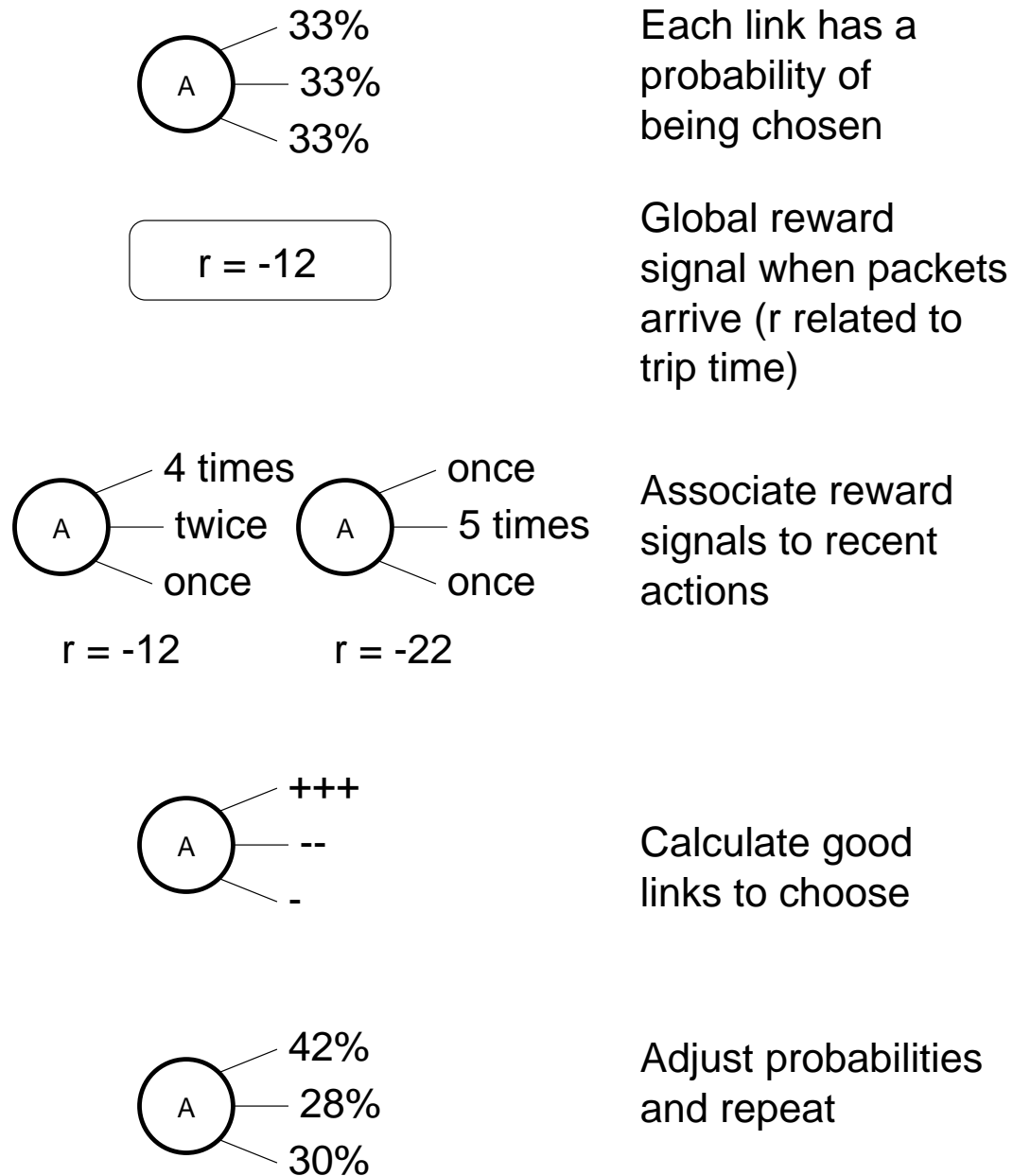


Figure 1.3: The policy-gradient reinforcement learning routing paradigm.

A novel approach is illustrated in figure 1.3. In this case, a routing agent associates a probability of choosing each link (or action). Over time, it tries a number of links, and associates reward signals with those links. Better links are correlated with higher reward values, and worse links with lower reward values. The probabilities of choosing each link are updated to favor higher performing links, and the process is repeated until an equilibrium is achieved.

1.3 Thesis Organization

The remainder of this thesis is split into six chapters. Chapter 2 provides the context for this research, by referring to relevant literature. Chapter 3 discusses the reasons why policy-gradient reinforcement learning may or may not be appropriate for network routing. Chapter 4 describes an application of the generic OLPOMDP framework to a routing simulator. Chapter 5 presents the results of the OLPOMDP routing algorithm to various networks of increasing complexity. Chapter 6 analyzes these results and discusses the effectiveness of the OLPOMDP algorithm. Finally, chapter 7 provides a summary and indicates directions for future work.

Related Work

Do not seek to follow in the footsteps of the wise. Instead, seek what they sought.

— Matsuo Bashō.

2.1 Network Routing

Network routing is the problem of how to efficiently use communication (or traffic) paths. However, there are several interpretations to the term ‘efficiency’, such as maximizing throughput or reliability, or minimizing latency or cost. Another concern is that optimizing one efficiency criterion may be detrimental to another. An example due to Tanenbaum [1996] is shown in figure 2.1. Suppose that there is enough traffic from X to Y to saturate the link $B \asymp C$. In this case, to maximize total flow, traffic from A to D should be shut off altogether, rather than cause congestion. However, this is not a fair allocation.

Current approaches to routing model the network as a graph with nodes representing routers and weighted edges representing links and their associated costs. Finding

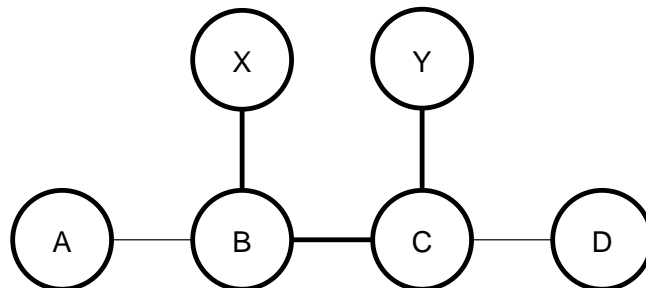


Figure 2.1: The problem with maximizing flow.

good approaches to routing is thus reduced to analytically solving various optimization problems for weighted graphs [Deo and Pang 1984].

With respect to minimizing total path weight, various algorithms exist to calculate exact solutions for point-to-point shortest paths [Dijkstra 1956; Bellman 1958] and all-pairs shortest paths [Cormen et al. 1990]. Such methods are termed Shortest Path (SP) algorithms. In an alternative approach, Ford and Fulkerson [1962] developed an algorithm to maximize network flow from a single source node to a single sink node, given link capacities.

In practice, routers inform their peers of the status of their links (e.g. which router a link connects to, its average delay, etc.), from which each router builds up its own graph model of the network, and then calculates shortest paths. There are a variety of router-to-router communication protocols, such as Distance Vector and Link State routing [Tanenbaum 1996].

2.1.1 Problems in Network Routing

Elegant solutions to the weighted graph models of routing problems have been found. However, as the models approach realism, the increasing complexity can render the problems intractable. In one example, Orda, Rom, and Sidi [1993] consider the problem of graphs where the edge weights can change probabilistically (as a Markov process). For this model, they conclude that the minimal expected delay routing problem is NP-hard, which means that, in practice, any algorithm will scale poorly.

There is also the problem of co-ordination, whether between routers or between packets from one router. Synchronization problems can occur where two routers sharing two links simultaneously notice that one of the links is not congested and both attempt to use it, causing congestion. This problem is worsened when the two routers now notice the other link is now free and both choose to use it, causing congestion again. This situation repeats, with the two routers oscillating between the two links — a phenomenon known as flapping [Tanenbaum 1996].

Packet co-ordination problems are vividly exemplified by Braess' paradox [Braess 1968], applied to queueing networks by Cohen and Kelly [1990]. In this case, each packet routing decision is made independently of the others, and is optimal given the other decisions as fixed. However, the addition of a new link, improving capacity, has the paradoxical effect of worsening overall cost.

Both of these examples demonstrate the problem of externalities. Model-based algorithms yield decisions which are optimal from the individual agent's (i.e. router's) perspective. However, their actions impose external effects (e.g. the cost of link congestion) on other agents' performance. Models that incorporate the choices available from every agent necessarily exhibit a large number of policy dimensions and thus usually scale poorly.

Even if the problem posed by the model is tractable, unrealistic modeling may

mean that the derived solution is sub-optimal. Real world networks exhibit a variety of features which are not captured in the standard weighted-graph model. Such complexities include links having limited bandwidth or capacity, limited buffering at either or both ends of channels, packet collisions, mobile hosts (i.e. changing link delays), deadlock concerns, unreliable links, non-uniform or bursty traffic (e.g. Poisson distributed traffic), minimum Quality-of-Service requirements, and prioritized traffic [Tanenbaum 1996].

The quest is for routing solutions which are robust to the complexities of real-world networks, and model-based algorithms are *prima facie* fragile when their assumptions are broken.

2.1.2 Alternative Approaches to Network Routing

Collective Intelligence

Tumer and Wolpert [1999] have developed a theory of collective intelligence (COIN), and applied it to a routing model. COIN is based on a global utility function, or a measure of collective performance. In this regard, it is similar to reinforcement learning. However, their approach is to analytically derive individual utility functions which are “aligned” to the global utility function. They identify the design of these functions as critical to the success of the system, which implies knowing an accurate model of the problem domain. Furthermore, these individual utility functions still incorporate observations of other agents, so it is not clear if their method distributes well.

In their experiment, they demonstrated that their COIN technique outperformed a particular Shortest Path algorithm for a particular, pathological network. However, they did not address whether COIN is feasible in general. Furthermore, they ran the traditional algorithm first in order to generate training data for the COIN technique, and so the comparison is between the Shortest Path algorithm versus the COIN technique on top of the Shortest Path algorithm. Finally, their experiments require the insertion of virtual zero-cost nodes into the network in order to “simplify the mathematics”, which suggests that their method may not be practical.

Markets for Network Resources

Standard micro-economic theory suggests that in a free market, an “invisible hand” will guide producers and consumers to an equilibrium which is optimal, without requiring each agent having explicit knowledge of the behavior, desires, and costs of any other agent [Varian 1999]. This leads to the idea of markets for network resources, such as bandwidth, or channels, where agents for producers (links) and consumers (traffic generators) agree on mutually beneficial exchanges. In the ideal case, transactions are instantaneous and unrestricted, each agent’s market power is small, and

both consumer's price valuations and producer's cost estimates are accurate. If those conditions are true, then the market mechanism will generate an optimum resource allocation. Even if there is uncertainty in the underlying resources (e.g. unreliable links), agents can incorporate risk preferences into their valuations, and markets for bearing risk can emerge.

A recent experiment to test the market approach to network routing was done by Gibney, Jennings, Vriend, and Griffiths [1999]. They conclude that market mechanisms are as good as static routing methods for their test network simulation. However, their architecture does not scale well, since it requires a market for every link, and an intermediate market for every path. For example, if links exist from $A \asymp B$, $B \asymp C$, and $C \asymp D$, then there is a market for the path $A \asymp B \asymp C \asymp D$, for which the consumers are packets routed on that path, and the producer buys in the market for the component links. The number of different paths in a network grows rapidly as the number of nodes increase, which means that the system as presented is not feasible in a realistic context. Their markets also stray from the ideal, in that in a distributed system, market transactions are not instantaneous, but bids and offers are affected by network latency.

Ant-based Routing

An alternative, unorthodox approach to network routing is based on imitating ant behavior, where shorter ant trails carry stronger pheromone traces. One of the first experiments was done by Schoonderwored, Holland, Bruten, and Rothkrantz [1996], applying the idea to a specific model of circuit switched networks. Caro and Dorigo [1998] developed a more realistic simulator for packet-switched networks, and reported that ant-based routing had outperformed certain algorithms, such as Distance Vector and Link State, under a variety of conditions. Their work is very interesting, although they do not provide optimality guarantees for their method.

2.2 Co-ordination of Multiple Agents

The formal analysis of the strategy of multiple agents, or actors, dates back to game theory [von Neumann and Morgenstern 1944], which was motivated by studies in economics and politics.

One of the fundamental concepts is that of a Nash equilibrium [Nash 1950]. In such a situation, every agent is pursuing the best policy it can, taking all other agent's behavior as given. At Nash equilibria, no agent has an individual incentive to modify its policy. Note that there can be multiple Nash equilibria in a system. Two examples, discussed by Dixit and Skeath [1999] illustrate the point.

First, consider the "co-ordination game", involving two actors, named Alice and Bob. There are two movies screening at the cinema, titled X and Y. Suppose that both

		Robber B's action		
		Confess	Deny	
Years in Prison for Robber A	Confess	5	0	← Robber A's Dominant Strategy
	Deny	12	1	
Robber A's action				

Figure 2.2: The Prisoner's Dilemma: confess, or deny?

Alice and Bob agree to watch a movie that night, but the phone gets cut off before they can agree on a movie. However, they discussed the fact that Alice prefers movie X, and Bob prefers movie Y. Alice's options are to choose to go to movie X, or to movie Y, and Bob's options are the same. Both people like the company of each other, so Alice would prefer to watch Y with Bob rather than watch X by herself, and vice versa for Bob. Now, if Alice knew that Bob was going to watch movie Y regardless, then her best strategy is to also go to movie Y. In this situation, neither person would want to change their strategy and choose movie X, without wanting the other person to change their behavior too. Similarly, the situation where both people choose to go to movie X is a Nash equilibrium.

The second example is the famous "prisoner's dilemma", which again has two actors, but only one Nash equilibria. Two bank-robbers are caught, but there is a lack of hard evidence to convict them. The prosecution offers a deal to each one of them: give evidence implicating the other person (i.e. confess), and you will walk away free. However, if one confesses and one denies, then the implicated robber faces a heavy jail term. If both robbers give evidence, then they share the blame and both end up with medium jail terms. If neither robber confesses, then they will still get prosecuted for car theft and face a light jail term. The pay-off matrix is shown in figure 2.2.

Each robber has to make a decision without being able to talk to the other robber. Consider robber A's choice — he knows that the same offer has been given to his partner B. If B confesses, then either A denies (for a heavy jail term), or confesses (for a medium jail term). If B denies, then either A denies (for a light jail term), or confesses (to walk away a free man). In both cases, A's better option is to confess, and similarly for B. Thus, the only Nash equilibrium is the case where both robbers confess, for a medium sentence each. Note that this is clearly not the optimal overall policy (for the pair of robbers), since if they both deny, then they are both better off with a light sentence. However, by pursuing their individual goals, the only rational

outcome is sub-optimal.

These examples demonstrate the problems of multiple agents having to co-ordinate their actions. A similar story due to Hardin [1968], outside of the game theory literature, is entitled, “the tragedy of the commons”. In this case, there is a lake which is common property, and can be used for fishing. However, if too many fish are caught, then the fish population will slowly die out. Individual fishermen will still over-fish, since their incentive is to maintain their livelihood, and the impact of their individual fishing is small on the fish population. However, because all fishermen over-fish, the fish population eventually dries up.

The problem is in the existence of externalities, or the effects by one agent on another agent’s wellbeing. In the network routing context, if a router sends traffic down a path, then that removes capacity that other routers may have used, and can add to queueing times for other packets.

One solution is to have a central authority to co-ordinate all the agent’s actions, but this scales poorly due to the communication overhead. Coase [1960] noted that it is still possible for decentralized agents to co-operate if external effects can be incorporated into individual incentives, such as imposing a fishing license fee.

2.2.1 Credit Assignment

In the field of artificial intelligence, the problem of co-ordinating multiple agents is linked with the idea of credit assignment. A number of actions are responsible for final performance, and receipt of a reward should be acknowledged to all actions which contributed. For example, in navigating a maze, the final step to the exit triggers the reward for success, but every (correct) step on the way is also important. The main problem is to determine which of the previous actions actually contributed, and which did not. Temporal difference methods [Sutton and Barto 1998] attempt to resolve this issue by presuming more recent actions are more likely to be responsible. An “eligibility trace” is maintained, which determines which actions are eligible to be acknowledged for current performance.

A similar problem, where the results of one agent are processed by another agent, is addressed by Holland [1985] in his work on classifier systems. In this case, the bucket brigade algorithm is used to transfer credit from higher-level agents (whose performance is measurable) to the lower-level agents (whose outputs are merely intermediate products consumed by higher-level agents).

2.3 Reinforcement Learning

Reinforcement learning is the study of algorithms which improve over time, given a numerical performance measure for feedback [Sutton and Barto 1998].

In the classic reinforcement learning model, the agent encapsulates the algorithm (and its learning), and is considered separate to its environment. The agent observes the state of the environment, as well as the reward associated with that state. It then decides upon an action or control, based on its policy. The environment responds to the agent's action, possibly changing its state. The agent associates expected reward values with each state (or each state-action pair, in the case of Q-learning [Watkins 1992]), and learns these values over time with experience. The resultant policy is greedy, choosing actions that will maximize (estimated) expected return. If the model is correct — e.g. in the case of a discretized maze, where each grid cell corresponds to a state and each action, such as “move north”, determines the next state — then the agent will converge upon the optimal policy [Sutton and Barto 1998].

The key problem with classic, or state-evaluation, reinforcement learning is the problem of scalability, also known as the “curse of dimensionality” [Bellman 1957]. The valuations of states can only be updated when those states are experienced by the agent. As the number of states in the system model grows, the time it takes to return to states can grow exponentially. Consequently, the time taken to converge to a correct state evaluation function (and hence an optimal policy) can grow rapidly. The problem is worsened in the case of Q-learning, where if there are n states and m actions possible in each of those states, then there are $n.m$ values to update, rather than merely n .

In real world applications, the degree of detail needed in the input (i.e. the number of states in the model) prohibits classical reinforcement learning. The contemporary approach is thus in approximate value functions, where estimated expected rewards are not maintained for individual states (or state-action pairs, for Q-learning). Instead, a parameterized function (such as a linear combination of feature-detectors) returns the approximate value of a state. This approach is exemplified by Tesauro's world-class backgammon player [Tesauro 1994]. Here, the agent reduces the huge number (10^{20}) of the game's states to two dozen salient features, which are inputs to a neural network. A similar approach has been applied to chess [Baxter et al. 1998] and checkers [Samuel 1959].

2.3.1 Reward Shaping

The reward signal gives an agent feedback on its behavior. The design of a good reward signal (i.e. good feedback) can help facilitate learning. One technique is reward shaping, which has its foundations on earlier work by psychologists such as Thorndike, Pavlov, and Skinner, in the field of animal training and conditioning (discussed in Sternberg [1998]). With shaping, a complex behavior (e.g. jumping through a flaming hoop) is not immediately taught to an agent. Rather, a series of increasingly complicated approximations (e.g. first to jump, then to jump through a hoop, then to jump through a flaming hoop) is taught. This procedure is illustrated diagrammatically in figure 2.3. In machine learning, reward shaping has successfully been applied to

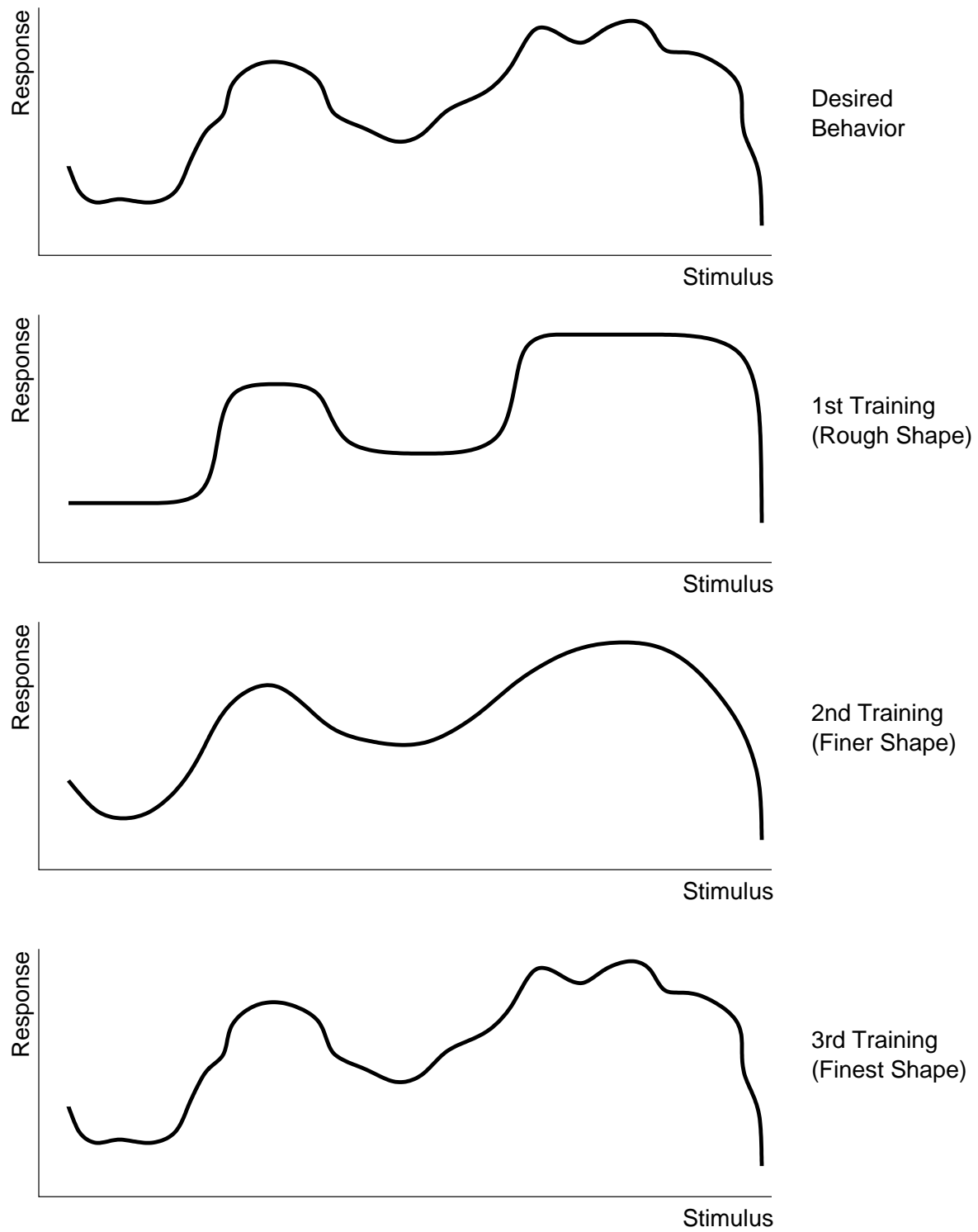


Figure 2.3: Reward (i.e. training signal) shaping.

teaching bike-riding [Randløv and Alstrøm 1998], and a theory for shaping is beginning to emerge [Ng et al. 1999; Mataric 1994].

2.3.2 Reinforcement Learning Applied to Routing

Boyan and Littman [1993] adapt the Q-learning technique to develop Q-routing: a reinforcement learning routing algorithm. In this application, value estimates are communicated between agents, where the value of a state (i.e. the node a packet is at) is basically the expected time-to-arrival at the destination node. This approach is very similar to Distance Vector routing, and in fact they have the same roots. The Distance Vector algorithm is also known as Bellman-Ford routing [Bellman 1958], whereas Q-learning (and state evaluation in general) can be traced to dynamic programming, founded by the same Bellman [1957]. The Q-routing algorithm can suffer from the same count-to-infinity problem as Distance Vector routing [Tanenbaum 1996].

The authors report that Q-routing out-performed static routing, due to the fact that Q-routing is adaptive. As traffic load (and queueing time) increases, the router learns to prefer alternative routes. However, Q-routing is a greedy algorithm, in that the highest valued link is always taken. This means that alternative links are never tried, so changes to the network may be ignored. Furthermore, if the value of the best path is under-estimated and an alternative path has a higher value, then the best path is never taken.

One technique to encourage exploration is to have initial values assigned optimistically [Sutton and Barto 1998], so that the agent tries a range of actions before it settles on the best link. However, this is only a temporary solution and does not allow for adaptability to changes in links not chosen.

2.4 Gradient Ascent Methods

Gradient ascent methods have been in use for some time, in a variety of disciplines. In the field of machine learning, the famous BACKPROPAGATION algorithm for training neural networks is based on gradient ascent [Mitchell 1997].

Figure 2.4 shows the basic gradient ascent model. In this example, there is only one policy parameter, shown on the horizontal axis. Let the vertical axis chart the expected performance as a function of that policy parameter. Suppose we wish to find the best policy — i.e. the parameter value (vertical dashed line) which has the maximum expected performance (the height of the thick curve).

The gradient of a curve is an indication of the upward direction — following that direction will improve expected performance. In this two-dimensional example, the gradient is simply the tangent to the curve. At any point, the gradient vector points left if and only if the optimal policy point also lies left. This leads to the gradient ascent algorithm: continually calculate the gradient, then take a step in the gradient direction,

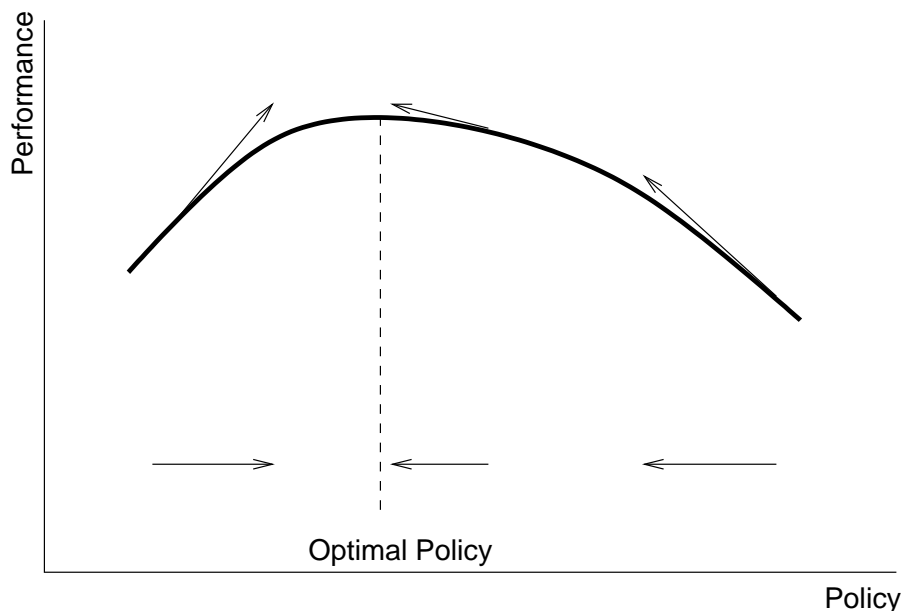


Figure 2.4: Movement in the gradient direction will lead to an optimum.

until you come to a peak. This is also known as the hill-climbing algorithm, which can be used to find the top of a hill in the dark. In this analogy, you continually feel the ground around you to find which direction slopes up, take a step in that direction, and repeat until you can't get any higher.

The advantage of gradient ascent methods is that, provided you can estimate the gradient accurately, and that the size of your steps are sufficiently small, then you will eventually find the top of the hill you are at. This gives a guarantee of optimality that some alternative algorithms cannot provide. The disadvantage is that there is no guarantee that, although you find the top of a hill, there aren't any larger mountains you could have got to. Figure 2.5 shows such a situation, where the policy converged to depends on the starting point. Starting on the left hand side will lead to the globally optimum policy, whereas starting on the right hand side leads to a globally sub-optimum (but locally optimum) policy.

The problem of sub-optimal convergence has also been identified in the field of genetic algorithms [Holland 1975]. In this context, the performance measure is the fitness of individuals, and gradient ascent is achieved by genetic cross-over of fitter individuals. The standard work-around is to have a number of different populations, or demes, starting from different points in policy space. Each deme evolves independently. Although some populations may converge to mere local optima, it is presumed that some populations will converge to the global optimum. A simple comparison of fitness across populations will reveal the best policy to take, which will hopefully

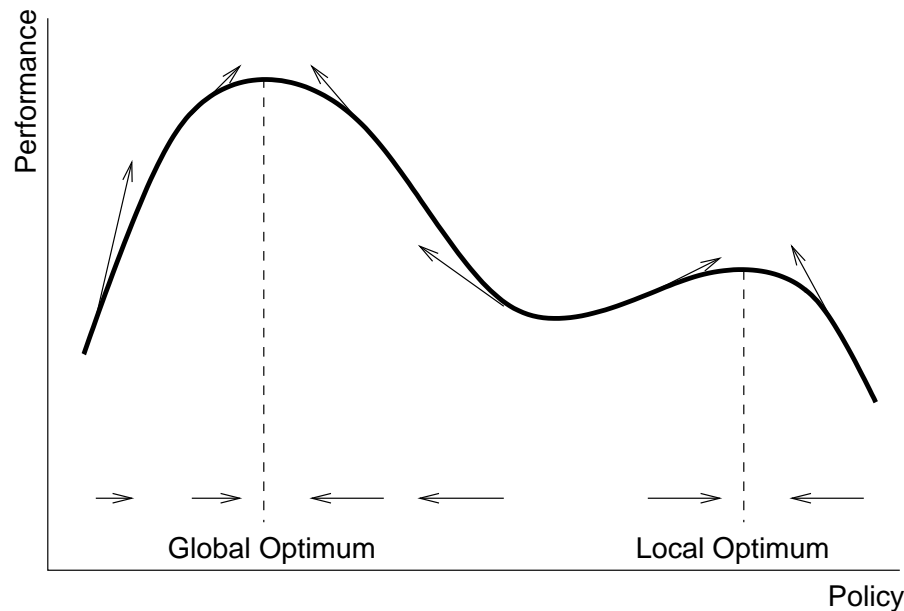


Figure 2.5: The problem of multiple local optima.

be the global optimum. This method, of running a number of parallel, non-interacting systems, can translate into machine learning — a number of agents, each with different starting parameters, can be trained on identical systems. If there are enough agents, at least one of them will be likely to converge to the global optimum, and thus discover the best policy. Note that this strategy is only applicable when the environment the agent lives in can be repeated, and reset, as is the case with computer simulations.

Two other approaches are momentum [Mitchell 1997] and simulated annealing [Kirkpatrick et al. 1983]. With momentum, parameters are updated not only by the current gradient estimate, but also by some memory of past gradient estimates. The hope is that mere local optima are smaller and flatter than global optima, and a learning agent with momentum will run up a hill, keep traveling through the small valley without stopping at the hill-top, until it reaches the mountain. With simulated annealing, there is a notion of system ‘temperature’, which ‘cools’ over time. When the system is ‘hot’, the step size in the gradient direction is very large, which means that the agent tend to overshoot the local optima, and instead jump around the entire policy space. However, as the system ‘cools’, the agent takes smaller and smaller steps. Presumably, the global optimum will have a larger basin of attraction, and so still be reached in medium sized steps, whilst the smaller local optima will be over-stepped. Consequently, the highest ‘temperature’ at which convergence still occurs will lead to a global optimum.

2.4.1 Policy-Gradient Reinforcement Learning

The modern attack on the curse of dimensionality is through approximate value functions. However, by restricting the range of possible state evaluation combinations, the agent is constrained in the class of policies it represents. In particular, optimizing a state evaluation function approximation under this constraint may not result in the optimal policy. In a vivid example [Weaver and Baxter 1999] an agent, starting from evaluation function parameters implementing the optimal policy, converged upon parameters giving the worst policy.

To avoid this problem, more recent approaches are not based on state evaluation, but instead modify the parameters of the policy directly. With policy-gradient methods, learning is seen as a gradient search through policy space. The aim is to reach the point in policy space which yields the maximum expected average reward, which is a function of the policy. By changing the policy parameters incrementally in the direction of the gradient, the agent converges to a local (and hopefully global) optimum.

The difficulty is in estimating the gradient with respect to policy parameter space. Williams [1992] developed one of the earliest algorithms for gradient estimation and ascent, which was restricted to limited class of problems, and required a reward baseline as a reference. Later work by Marbach and Tsitsiklis [1998] derived an algorithm which only relies on a single sample path, or trajectory, through the environment, and thus can be the basis of an agent which learns on-line. However, the estimates relied upon the notion of a recurrent state, which the system would return to infinitely often. The time between recurrences could be quite large, especially for systems such as the Acrobot control problem [Dejong and Spong 1994], in which desirable policies avoid the states that the system spends its learning period in. The OLPOMDP algorithm, developed by Baxter and Bartlett [1999], removed the recurrent state criterion, instead introducing a bias-variance trade-off in gradient estimation. Various successful experiments regarding the applicability of that algorithm are presented by Baxter, Weaver, and Bartlett [1999].

Motivation

If a man write a better book, preach a better sermon, or make a better mouse-trap than his neighbor, though he build his house in the woods, the world will make a beaten path to his door.

— Ralph Waldo Emerson.

Routing is still an important real-world problem. Labovitz, Malan, and Jahanian [1997] report that Internet backbone routers (operating the Border Gateway Protocol) receive up to 700,000 network update messages a day. They link this instability to packet loss, increased network latency, and longer time to convergence.

This thesis presents the first known application of policy-gradient reinforcement learning (PGRL) techniques to the domain of network routing. There are several reasons why PGRL methods could be suitable or unsuitable for such a domain. These reasons are outlined below.

3.1 Why PGRL could be Appropriate for NR

3.1.1 Robustness

Traditional model-based algorithms attempt to optimize some cost function with respect to a model of reality, such as weighted graphs representing link costs. Current theory can provide analytical solutions, provided the model is simple enough. However, real world computer networks are complicated. Correctness or optimality of PGRL methods do not rely upon a particular model, and consequently are quite robust.

3.1.2 Guaranteed Performance Improvement

Gradient ascent methods are guaranteed to improve in performance, on average, over time, unless they are at an optimum. This guarantee requires that certain conditions are

met, such as the step size being sufficiently small. Other machine learning methods may not provide such a guarantee, and model-based methods are sensitive to their model's underlying assumptions.

3.1.3 Implicit Co-ordination

Policy-gradient methods can be used to maximize a global performance measure, without the need for explicit communication between individual agents. If each agent receives the same global reward signal, then they each strive to maximize that signal, and thus implicitly co-operate for the common good.

Alternative methods may require a centralized managerial agent (which becomes a bottleneck as the system scales), or else agents are faced with individual incentives. Such an incentive scheme (or utility function) may not accurately capture external effects on other agents. For example, if an agent chooses to saturate a particular link, then no other agent can use that link effectively.

3.1.4 Distributable Computation

Network routing is inherently a distributed system. Each router must make its own routing decisions, and there is no central and timely record of network state. Model based algorithms require a knowledge of network state, and so modifications to the network must be distributed as messages to each router. This can cause inconsistencies in different router's models, such the "count to infinity" problem in Distance Vector routing [Tanenbaum 1996].

The PGRL framework does not need to maintain a model of the network, and so does not have the problems associated with distributed shared memory. Furthermore, it does not have to explicitly send co-ordination messages to its peers. Provided a global reward signal is broadcast, multiple PGRL agents will work together to maximize the global reward.

3.1.5 Flexibility

For a gradient to be calculated, the range of representable policies must be smooth. Consequently, PGRL methods are necessarily flexible in the class of policies they can learn. Traditional routing algorithms decide on one best link and solely choose that link. It may be that the optimum policy is to choose a range of links. For example, if two link costs are equal, and they are both bandwidth limited, then it may be best to use each link equally, rather than either one or the other exclusively.

3.2 Why PGRL could be Inappropriate for NR

3.2.1 Convergence Time

Any reinforcement learning technique relies upon improvement through experience. If the data is noisy, then the each update must be small enough so that the long term trend is allowed to show through the short term noise. Noticeable improvement may therefore require a large number of updates, and a long convergence time.

In network routing, model-based algorithms calculate the optimal path as soon as the relevant information is available. Routers are regularly informed of changes in network topology, and can re-calculate their policy accordingly. For frequently updating networks (e.g. with physically mobile hosts), PGRL methods may be inappropriate if convergence times are longer than the times between network changes.

3.2.2 Multiple Local Optima

Gradient ascent methods continually improve their performance by small adjustments until an equilibrium is reached. At this point, no small adjustment can improve performance, but improvement may still be possible by a radical shift. Policy-gradient methods may converge to locally optimal (but globally sub-optimal) solutions.

In the context of distributed or multi-agent decision-making, this frequently arises as the problem of multiple Nash equilibria [Friedman 1994]. A Nash equilibrium exists where each agent is acting optimally when taking the other agent's behavior as fixed. In the case of multiple Nash equilibria, the system may converge to any one Nash equilibrium. In this state, no agent can unilaterally move the system to another (better performing) Nash equilibrium. Finding a globally optimal policy may require explicit co-ordination between agents.

3.2.3 Non-Determinism

PGRL agents do not rely on a system model, and so they can only make inferences through their experiences, which may be subject to noise. Regardless, the information they derive, such as estimates of the reward gradient in policy space, is based on a single sample path (its experience), and so it is inherently probabilistic. Although there exist theoretical performance guarantees, they are not absolute. For example, the convergence guarantee states that the probability that an optimum is achieved can be made arbitrarily close to 1, given sufficient time. However, it is possible that, by bad luck, a PGRL system may perform poorly. A non-deterministic algorithm cannot give worst-case guarantees.

3.2.4 Scalability

As the number of agents increases, the effect of an individual agent's actions is a smaller proportion of the global performance measure. Consequently, it becomes increasingly difficult for each agent to distinguish which changes in the reward signal it is responsible for, and which changes are due to the choices of other agents.

Furthermore, as the network size increases, optimal solutions may require the co-ordination of many agents. For example, a packet is routed by a number of agents on the way from A to B , and optimal performance requires those agents to co-operate. As the number of co-operating agents required increases, the learning task is presumably harder, and may become intractable.

3.2.5 Reward Signal Distribution

Reinforcement learning agents rely on a reward signal in order to differentiate between good outcomes and bad outcomes. In policy-gradient reinforcement learning, this reward signal is used to generate gradient estimates.

However, by the necessarily distributed nature of computer networks, there is no global reward signal broadcaster. Instead, reward signals have to be added to router-to-router messages, such as in network update notifications, or piggy-backed on data packets. Alternatively, reward signals will be flooded throughout the network. Either technique will induce a lag between a router's actions and its receipt of that action's consequence on performance (through changes in its reward), which may impact on its convergence properties.

3.3 Summary

The advantages of policy-gradient reinforcement learning methods were identified as their robustness, guaranteed performance improvement, implicit co-ordination, distributable computation, and flexibility. The disadvantages were identified as their convergence time, multiple local optima, non-determinism, scalability, and the need for reward signal distribution. Each of these factors are assessed in chapter 6.

Design

Two roads diverged in a wood and I —
I took the one less traveled by,
And that has made all the difference.

— Robert Frost

A discrete-time routing simulator was developed, and its object-oriented design is detailed in section 4.1. The OLPOMDP algorithm was adapted to network routing, and is described in section 4.2.

Each simulation run was non-deterministic, in that packets were generated for random destinations, and for the OLPOMDP routing algorithm, links were chosen according to a probability distribution. Consequently, it is difficult to test a simulator implementation against expected output. Nonetheless, statistical tests were performed to increase confidence in the implementation's correctness. Details of the verification procedure are presented in appendix A. The verification experiments also provided a guide for the parameters (e.g. β , γ) of the OLPOMDP algorithm.

4.1 Simulation Design

Each `Network` consisted of a set of `Nodes` and a set of `Links`, and each `Link` joined exactly two `Nodes`. A `Network` generated a `RoutingPolicy` for each `Node` according to a `Factory Method` [Gamma et al. 1995]. Similarly, each `Node` had a `PacketGenerator`, which created the traffic to be routed in the `Network`. Each `Link` had a delay, which is the number of time steps between when a packet is placed on a `Link` and when it arrives at the `Node` at the other end of it. Further properties, such as limited bandwidth and one-way channels, were modeled by different subclasses of `Link`. Upon arriving at its destination, a `Packet` notified the `Network's Statistician`, which calculated metrics such as average trip time and hop count. A `RoutingPolicy` is either deterministic (e.g. `Static Shortest-Path`) or non-deterministic (e.g. `OLPOMDP`), with those cases implemented by separate `RoutingPolicy` sub-classes.

4.2 Design of a PGRL Routing Algorithm

4.2.1 The Reinforcement Learning Framework

The reinforcement learning paradigm is depicted in figure 4.1. An artificial agent continually interacts with an environment, in a discrete time system, by making observations y_t and executing actions u_t . Part of the interaction process also involves the receipt of a numerical reward value r_t . It is not necessary that the observations y_t are perfectly accurate. Instead, we model the observations as drawn from a probability distribution dependent on the environment state x_t , i.e. that $y_t \sim v(x_t)$, for some function v . This assumption is also known as partial observability. We presume that the agent's actions influence the state of the environment, i.e. $x_{t+1} \sim P(u_t, \dots)$, for some function P .

An agent's policy represents its 'intelligence', or how it chooses its actions. At the simplest level, it is merely reactive: a policy maps observations to immediate actions, $u_t = \mu(y_t)$. A broader class of agents maps observations to probability distributions over actions ($u_t \sim \mu(y_t)$). Policies are adaptive, in the sense that they are parameterized by a vector $\vec{\theta}$, and that this vector may change over time as the agent learns from its experiences.

A common simplifying assumption, which enables proofs of optimality, is that the state of the environment is a Markov decision process, viz. that x_{t+1} merely depends on x_t and u_t , but not any previous states x_{t-1}, x_{t-2}, \dots or actions u_{t-1}, u_{t-2}, \dots . In mathematical terms, this allows us to write $x_{t+1} \sim P(x_t, u_t)$.

An agent attempts to maximize the expected long-term average reward value:

$$\eta = \lim_{T \rightarrow \infty} \mathbf{E} \left[\frac{1}{T} \sum_{t=1}^T r_t \right]$$

The reward r_t depends on the state of the system x_t , which is influenced by the agent's previous action u_{t-1} , which is a result of its policy μ , which is parameterized by $\vec{\theta}$. Consequently, the expected average reward η is a function of $\vec{\theta}$. Learning can be seen as the process of adjusting $\vec{\theta}$, in response to the stream of observations y_t , rewards r_t , and actions u_t , in order to maximize $\eta = \eta(\vec{\theta})$.

4.2.2 The Policy-Gradient Reinforcement Learning Framework

Policy-gradient reinforcement learning involves updating $\vec{\theta}$ via gradient ascent, i.e. maintaining an estimate of $\nabla \eta(\vec{\theta})$, and adjusting $\vec{\theta}$ in that direction.

The On-line Learning algorithm for Partially Observable Markov Decision Processes [Baxter and Bartlett 1999] maintains an eligibility trace \vec{z} and updates $\vec{\theta}$ according to

$$\vec{\theta}_{t+1} = \vec{\theta}_t + \gamma_{t+1} \cdot r_{t+1} \cdot \vec{z}_{t+1} \quad (4.1)$$

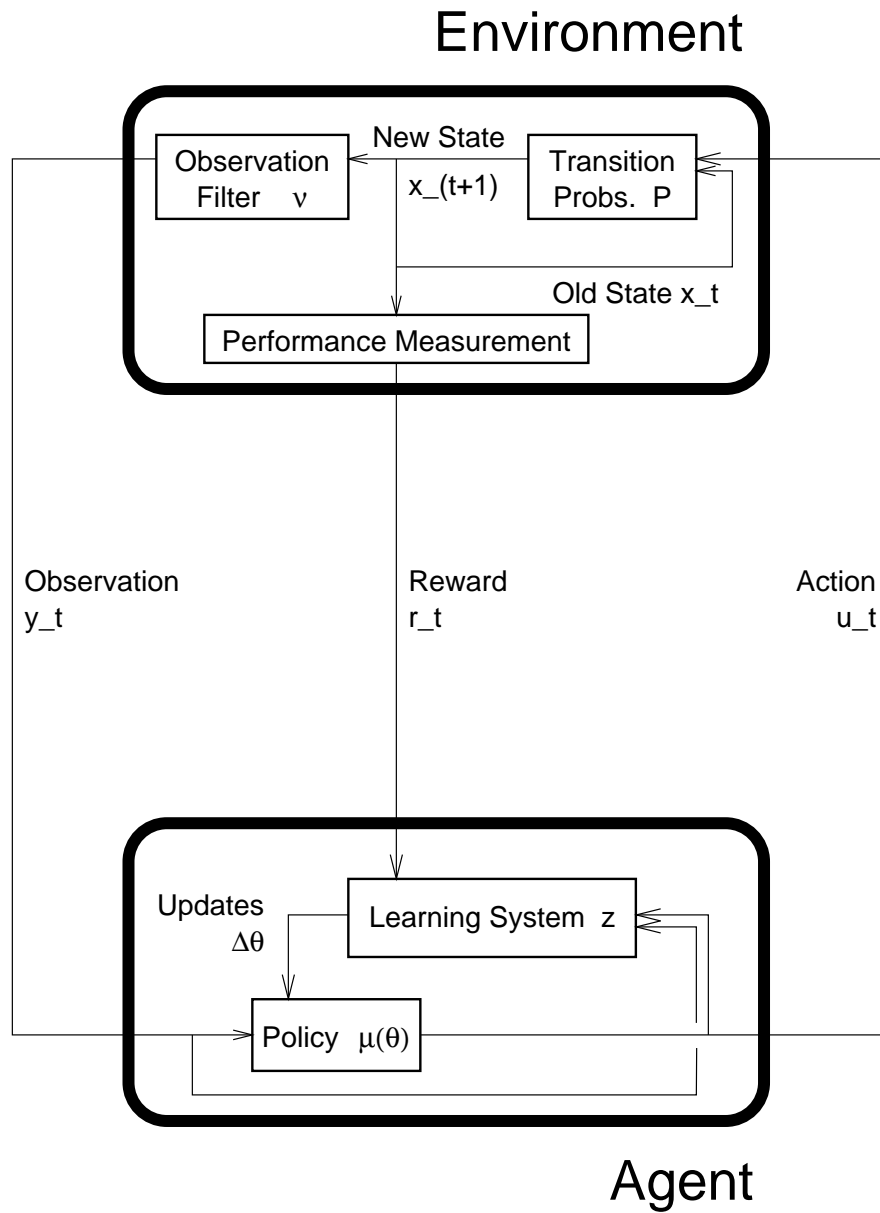


Figure 4.1: The reinforcement learning architecture, in detail.

where γ_t is a series of step sizes which parameterize the algorithm. Provided γ_t is sufficiently small, then the accumulated vector $\sum r_t \cdot \vec{z}_t$ approximates the policy-gradient, given sufficient time.

Convergence requires that $\sum \gamma_t = \infty$ and $\sum \gamma_t^2 < \infty$ [Kushner and Yin 1997], which leads to a decreasing series of step sizes. However, the environment (network) may be constantly changing, so step sizes which decay over time imply slower convergence rates to a moving target. For simplicity, we choose:

$$\gamma_t = \gamma_0 \tag{4.2}$$

Informally, the eligibility trace \vec{z}_t is an average of the ‘good’ directions in $\vec{\theta}$ space, with the direction vectors normalized and the average weighted towards recent measurements:

$$\vec{z}_{t+1} = \beta \cdot \vec{z}_t + \frac{\nabla \mu_{u_t, y_t}(\vec{\theta})}{\mu_{u_t, y_t}(\vec{\theta})} \tag{4.3}$$

where μ_{u_t, y_t} is parameterized by $\vec{\theta}$ and denotes the probability of choosing action u_t on observing y_t . $\beta \in [0, 1)$ is a parameter which can be interpreted as how strongly past inputs (y_t, r_t) influence future updates. As $\beta \rightarrow 1$, then $\mathbf{E}[r_t \cdot \vec{z}_t]$ approaches the gradient of η with respect to $\vec{\theta}$. On the other hand, as $\beta \rightarrow 1$, the variance of \vec{z}_t increases. Choice of β thus involves a bias–variance trade–off.

4.2.3 The n -Armed Bandit Analogy

Sutton and Barto [1998] give the analogy of the n -armed bandit, which motivates the PGRL routing algorithm below. A “one-armed bandit” is the colloquial term for a slot machine, which costs a certain amount of money to play, and has a variable monetary return. The one-armed bandit problem is to estimate what is a fair cost to pay for one play. The best estimate is simply the average of all past returns.

The n -armed bandit problem is where the slot machine has n levers to pull, rather than just one. The return from the slot machine is still probabilistic, but it also depends on which lever was pulled. In this case, the problem is to maximize the average return over a (possibly infinite) number of plays.

The dilemma is in choosing between exploration and exploitation. Initially, one should try each of the levers a number of times, to “get a feel” of the returns from each lever. This exploratory behavior builds up a knowledge base for the exploitation stage. Once the value of each lever has been estimated to a sufficient degree of confidence, then that knowledge can be exploited by always taking the lever of highest value.

However, the transition from exploratory to exploitative behavior is difficult to time. If it is too early, then a steady but under-performing lever might be favored whilst another high-risk, higher-return lever sits idle. In this case, the best policy will not be discovered. If it is too late, then the agent will spend too long choosing among

sub-optimal actions. Again, the best policy is not chosen.

A further problem is when the values of the levers change throughout the experiment. Suppose that the returns are dependent on a small cog in the machinery. Over time, this cog could wear down, or even break, which could change the expected rewards from each lever. A greedy strategy (i.e. sticking to only one lever after an exploratory phase) will not adapt to a changing environment, and another approach is needed.

4.2.4 The OLPOMDP Routing Algorithm

The OLPOMDP algorithm is straightforward to apply to packet routing, with one agent for every source-destination pair. Let $A \leftrightarrow B$ denote the routing agent for packets at A destined for B . Each agent faces an n -armed bandit problem, having to continually choose one link from the n at that agent's node.

For simplicity, we elect to not to observe the environment. Future work may extend the algorithm by incorporating measures of link utilization, or keep some memory of performance as part of the observations. However, by ignoring y_t , we will keep μ simple, i.e. $\mu_{u_t, y_t}(\vec{\theta}) = \mu_{u_t}(\vec{\theta})$.

Each agent keeps a parameter θ_l for each link l . The actions available to the agent are denoted u_l : one for each link l to route packets down. The probability of choosing action u_l is simply proportional to e^{θ_l} :

$$\mu_{u_l} = \frac{1}{s} \cdot e^{\theta_l} \quad (4.4)$$

where s is a normalization term to ensure that $\sum_{\lambda} \mu_{u_{\lambda}} = 1$. In particular, $s = \sum_{\lambda} e^{\theta_{\lambda}}$, so that $\partial s / \partial \theta_{\lambda} = e^{\theta_{\lambda}}$. In both cases, the sum is over all links λ attached to this node.

The advantage of using the normalized exponential is simplicity:

$$\frac{\partial \mu_{u_l}}{\partial \theta_{\lambda}} = \frac{(\partial e^{\theta_l} / \partial \theta_{\lambda}) \cdot s - (e^{\theta_l}) \cdot (e^{\theta_{\lambda}})}{s^2}$$

so that

$$\begin{aligned} \frac{\partial \mu_{u_l} / \partial \theta_{\lambda}}{\mu_{u_l}} &= \frac{(\partial e^{\theta_l} / \partial \theta_{\lambda}) \cdot s - (e^{\theta_l}) \cdot (e^{\theta_{\lambda}})}{s^2} \cdot \frac{s}{e^{\theta_l}} \\ &= \begin{cases} (s - e^{\theta_{\lambda}}) / s & \text{if } \lambda = l \\ -e^{\theta_{\lambda}} / s & \text{if } \lambda \neq l \end{cases} \\ &= \begin{cases} -\mu_{u_{\lambda}} + 1 & \text{if } \lambda = l \\ -\mu_{u_{\lambda}} & \text{if } \lambda \neq l \end{cases} \end{aligned}$$

This means that our update rule is captured by the equations 4.1, 4.2, 4.3, 4.4, and:

$$\nabla\mu/\mu = \left[-\mu_{u_1}, -\mu_{u_2}, \dots, -\mu_{u_k} + 1, \dots, -\mu_{u_{|\mathcal{L}|}} \right] \quad (4.5)$$

where $|\mathcal{L}|$ is the number of links on this node, and u_k was the link chosen. The OLPOMDP routing algorithm is summarized on page 31.

4.2.5 Reward Signal Design

Higher r should indicate better performance. A natural performance metric is packet trip time — how long it takes for a packet to get from its source to its destination. Shorter trip times are better, so a reward signal would be based around the negative of the trip times. Each packet should be weighted equally, and so should be counted only once in the reward signal. To achieve this, we set packets to trigger rewards only in the time step they arrive at their destination. A simple reward signal would then be the sum of the negative of the trip times of all such packets:

$$r_t = \sum_{p \in P_t} -trip_time(p)$$

where P_t is the set of all packets which arrived at their destinations at time t .

In a distributed system, a global reward signal cannot be instantaneously delivered to every agent. However, the basic idea is easily adapted to broadcast rewards. Each destination node A simply computes its own reward component per time unit:

$$r_t^A = \sum_{p \in P_t^A} -trip_time(p)$$

where P_t^A is the set of all packets which arrived at their destination A at time t . Each node then broadcasts this component r_t^A to every other node. The agent's training signal is simply equal to the sum of all reward components received for its host node in that time step.

4.3 Heuristic Optimizations

Although the above method is guaranteed to converge to a (local) optimum *in the long run*, performance can be improved by refining the algorithm.

4.3.1 Restricting the Range of θ

For an agent to learn to prefer action u , it first must receive positive reinforcement after trying u . Consequently, if the agent does not try u often in its exploration, it will take a

The OLPOMDP routing algorithm.

The agent $A \leftrightarrow B$ has:

- Location: Node A , which has $|\mathcal{L}|$ links.
- Parameters: $\beta, \gamma \in \mathbf{R}$.
- Variables: arrays $\vec{\theta}, \vec{z} \in \mathbf{R}^{|\mathcal{L}|}$
- Input: Reward signal $r \in \mathbf{R}$ every time step.

Initialization step:

- Set all variables $\vec{\theta}, \vec{z}$ to zero.

Repeat indefinitely:

1. Set the probability of choosing the i^{th} link as $\vec{\mu}[i] := e^{\vec{\theta}[i]} / \sum_j e^{\vec{\theta}[j]}$.
2. For each packet at A , destined for B , choose a link (denoted k) according to the probability distribution $\vec{\mu}$.
3. For each packet (and the link k it was sent on), increment \vec{z} by:

$$\vec{z}[i] := \begin{cases} \vec{z}[i] - \vec{\mu}[i] + 1 & \text{if } i = k \\ \vec{z}[i] - \vec{\mu}[i] & \text{if } i \neq k \end{cases}$$

4. On each time step, receive the reward signal r , and update $\vec{\theta}$ and \vec{z} by:

$$\begin{aligned} \vec{\theta} &:= \vec{\theta} + \gamma.r.\vec{z} \\ \vec{z} &:= \beta.\vec{z} \end{aligned}$$

long time to learn the correct value of action u . In network routing, an optimal policy may involve choosing one link as often as possible. This means that the probabilities of choosing other links are as small as possible. However, if the underlying network changes (e.g. new routers appear, links are broken), then the agent will take a long time to adapt to a different policy, simply because it does not explore very often.

In order to sustain some level of exploration whilst maintaining a near-optimal policy, we choose to place an upper bound on $\vec{\mu}[i]$ – the probability of choosing link i . Now, $\vec{\mu}$ is a monotonic, continuous function of $\vec{\theta}[i]$, so this is equivalent to restricting $\vec{\theta}[i]$ to a certain range. In particular, we choose the range $[-2, 2]$, since this allows for a maximum $\vec{\mu}[i]$ of $\frac{e^2}{e^2+e^{-2}} = 0.98$, a reasonable trade-off between exploiting an optimal policy and responsiveness to changes in conditions. Note that by requiring a minimum degree of exploration, it is no longer possible to represent *the* optimal policy, if it involves taking one link 100% of the time. However, reasonably good policies (i.e. 98%) are still representable.

4.3.2 Packet Time-To-Live

It is possible for packets to survive indefinitely. For example, a packet destined for A could be stuck in a cycle B to C to D to B . Over the course of time (and millions of packets being routed), it is inevitable that some packets will take a long time to arrive.

Although this does not affect the convergence of the algorithm *in the long run*, it is undesirable for two reasons. First, if there is limited bandwidth, packets which have drifted for a long time may well be far from their destination, and it may be better to discard that packet rather than let it take up valuable bandwidth.

Second, if a packet arrives (and thus generates a reward signal) long after the original routing decision, then the agent will have difficulty in deciding which of its previous actions was responsible for generating that reward, and thus have difficulty in evaluating which actions are better. Algorithmically, this corresponds to requiring a large β parameter, in order to maintain a longer memory of recent actions. The drawback of large β values is that it increases the variance in $\nabla_{\beta}\eta$, the estimate of the gradient, which implies longer convergence times.

In fact, if β is set too low, then it may be that a large negative reward, a long way into the future, is discounted so heavily that it is preferred to a smaller negative reward sooner. In that case, the agent will learn to route packets in cycles, rather than take the shortest route (which leads to an immediate negative reinforcement).

In response to this, we add a maximum packet Time-To-Live (TTL). If a packet has not arrived at its destination TTL time steps after it was generated, then it is dropped. Obviously, this is not a desirable outcome in the long run, and so a penalty signal (i.e. negative reward signal) is broadcast. For simplicity, we choose a penalty term equal to the negative of the TTL .

This imposes an upper bound on the time between an action and its effect. A

packet routed at time t will generate the entirety of its reward when it either arrives at its destination or it times out. The first case can not happen after time $t + TTL$, and the second case happens at time $t + TTL$. This upper bound is entitled the *mixing time* of a partially observable Markov decision process — the minimum time length after an action before it has no further influence on the system. Baxter and Bartlett [1999] show that if the term $\frac{1}{1-\beta}$ is large compared to the mixing time, then $\nabla_{\beta}\eta$ will be an accurate approximation¹ to $\nabla\eta$. Consequently, a knowledge of the mixing time ($= TTL$) allows us to choose a low β whilst still maintaining a reasonable estimate of the true gradient, $\nabla\eta$.

Note that if we choose TTL to be longer than the maximum time length of all point-to-point shortest paths, the system will still learn the optimal policy, since there remains incentive to choose the shortest path. Calculating the maximum length out of all shortest paths relies on calculating actual shortest paths, which defeats the purpose of not using an explicit shortest path algorithm. However, the diameter of the network (i.e. the sum of the weights of the longest path in the graph) is an upper bound to the maximum length of all point-to-point shortest paths. The network diameter may be available to an agent, either by system design, or by assumption. Establishing an upper bound for the mixing time will lead to a good choice of β , and thus good performance.

4.3.3 Reward Signal Decomposition

The agent $A \leftrightarrow B$ only makes choices for packets destined to B . Ignoring interaction effects, such as contention for limited bandwidth, packets not destined for B contribute to that agent's reward signal without that agent being able to influence them. This additional component to the reward signal hinders the agent's ability to interpret signal fluctuations as either a reflection on how good its actions were, or simply noise in the signal. This is detrimental to performance, requiring longer running times to smooth out noise, which is achieved by taking a smaller γ , or step size.

In order to provide better accountability for actions, the global reward signal can be decomposed into the packet trip times per destination node. The agent $A \leftrightarrow B$ would only listen to the reward signal for B , rather than the global one. This new reward signal, being more selective, accounts for all actions which that agent takes, but excludes a vast number of results for which that agent was not responsible. The improvement with this approach is demonstrated in the next chapter.

Of course, by following local rather than global incentives in a multi-agent system, the possibility of converging to a globally sub-optimum equilibrium arises. However, if there are no such concerns, then this approach should improve performance without any adverse consequences.

¹Technically, it provides an upper bound on $\|\nabla\eta - \nabla_{\beta}\eta\|$.

4.3.4 Cycle Detection

Packets which travel in cycles (e.g. A to B to C to A whilst destined for D) are exhibiting sub-optimal behavior. The basic OLPOMDP routing algorithm will eventually converge to an optimum solution, and therefore avoid cycles, but this may take a long time.

Cycles can be detected, if each packet maintains a history of nodes it has visited (or at least, the last n nodes it has visited, if packets have a fixed or maximum size). A penalty signal (i.e. a negative reward signal) can be issued upon cycle detection, which will discourage that behavior. Note that penalties are not issued for optimal behavior (since no optimal path contains cycles). Consequently, the introduction of a penalty system will not affect which policy is actually best.

The problem with cycle detection is the difficulty in appropriating blame. For example, suppose a packet destined for D was routed from A to B to C back to A again. It may be that the best route from B to D starts by going through C , or then again, it may not be. It is difficult to decide which decision was to blame without knowing shortest paths beforehand. However, if we knew shortest paths, then we might as well route by shortest path. By penalizing all participating routers upon detecting a cycle, agents may be penalized for taking the best decision. Thus, this 'optimization' could very well impede performance, rather than improve it.

An evaluation of whether or not cycle penalties are effective is undertaken in the next chapter.

Results

When you're going through hell, keep going.

— Winston Churchill.

This chapter describes the application of the PGRL routing algorithm (developed in chapter 4) to a number of networks. The networks in section 5.1 demonstrate that PGRL routing agents can achieve the optimum strategy, and can co-operate to do so. The networks in section 5.2 introduce performance heuristics in order to speed up the rate of convergence. The networks in section 5.3 are more complicated, and show that PGRL is robust, but also that it can be trapped in sub-optimal behavior. Finally, the network in section 5.4 address the concerns about the practicality of an instantaneous broadcast of the reward signal.

The natural measure of success of a PGRL application is the reward signal itself. Provided that the mathematical formula used to construct the reward signal accurately captures the desirability of the behavior, then a higher-valued reward signal implies a better policy. The reward signal charts in this chapter have been smoothed through a moving average filter (see appendix B). They have also been overlaid with the expected long-term average reward of the optimum policy. Note that the (short-term) actual performance may temporarily go above this line, since packet destinations are chosen randomly.

The other aspect charted is the actual policy of one agent, in particular the probability of it choosing a particular link. In the simple network model, there is one best link, and the optimal policy is to choose that link 100% of the time. In this case, the optimal probability line is merely the top of the chart.

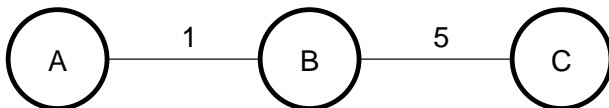


Figure 5.1: The linear 3-node network.

5.1 Simple Networks

5.1.1 Convergence to Optimum (the Linear 3-Node Network)

The first network examined is the the linear 3 node network in figure 5.1. In this simulation, one packet was generated at each node, to a random destination, every time step. The Time-To-Live was set at 70, and out of the 300,000 packets sent, only 26 were timed out. For packets generated at A , the expected reward signal under optimal routing is -1 for packets destined for B , and -6 for packets destined for C , giving an average (denoted $-\tau_A$) of -3.5 . Similarly, at B the expected reward is -3 and at C it is -5.5 , under optimal routing. Thus, under an optimum policy, the average¹ reward signal is $-3.5 - 3 - 5.5 = -12$.

Each agent was initialized with $\vec{\theta} = \vec{0}$, $\beta = 0.99$ and $\gamma = 10^{-5}$. The top graph in figure 5.2 shows the reward signal over 10^5 time steps for a typical run. It is evident that the system learns and maintains a near-optimal policy over time. The bottom graph of figure 5.2 shows the policy of one particular agent — the one responsible for routing packets from B to A . This agent has learned to route packets destined for A on the $B \asymp A$ link, which is clearly the best.

¹Here the average is per time step.

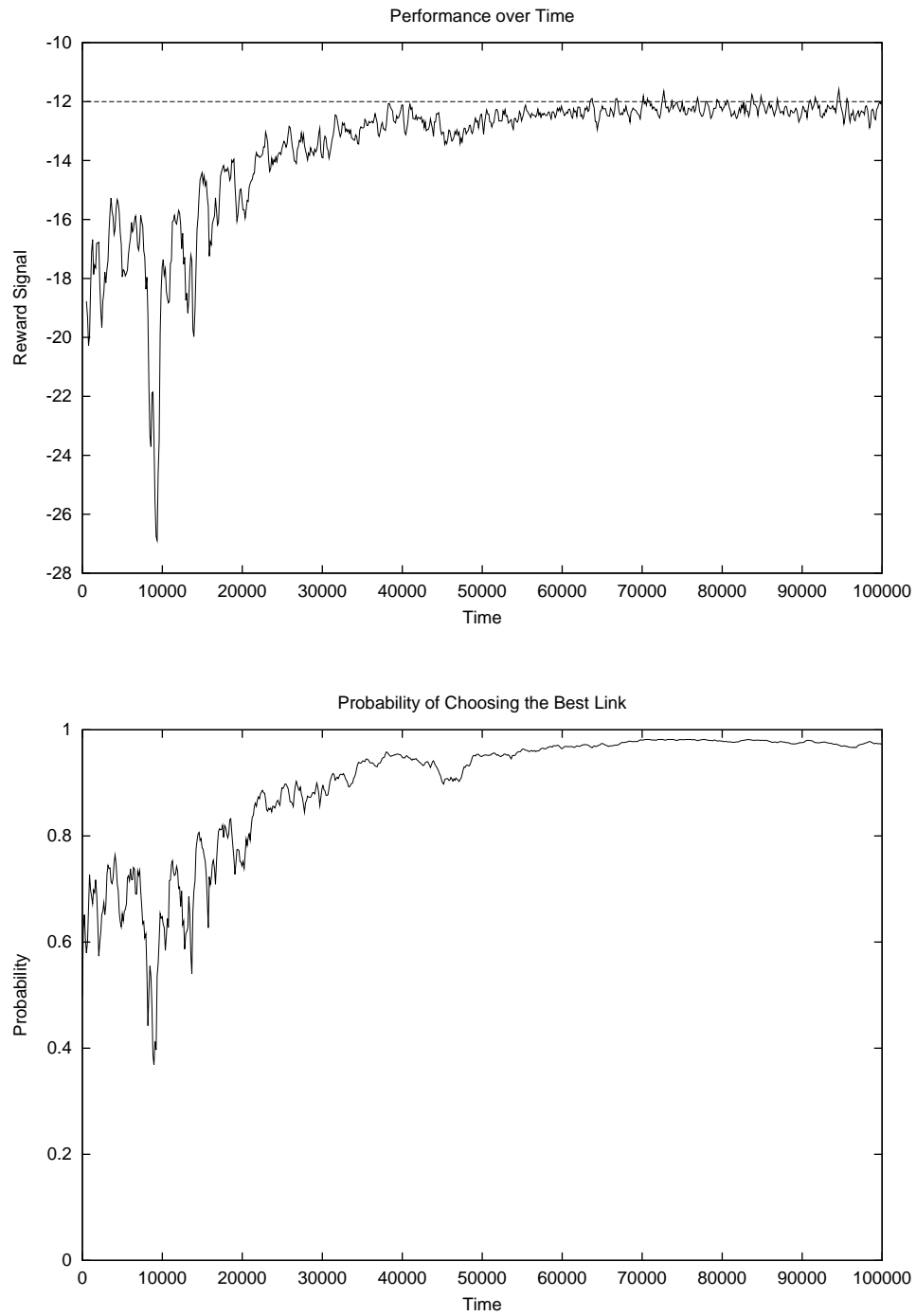


Figure 5.2: Reward signal r and probability $\mu_{B \succ A}^{B \rightarrow A}$ for the linear 3-node network.

5.1.2 Consistency (the Linear 3-Node Network Repeated)

The previous sub-section discussed the results of a single trial. The same experiment was performed 50 times, in order to assess the algorithm's consistency in achieving the optimal solution.

The reward signal is bounded above (by the performance of an optimal router), and highly skewed, since favoring the bad link rapidly leads to long trip times due to loops. Similarly, the probability of choosing the best link is bounded between 0 and 1. Consequently, the average and standard deviation are not the best ways to assess consistency. Instead, percentiles are displayed in figure 5.3, which accurately summarize skewed and bounded data.

In these charts, it is important to note that the continuous lines (e.g. the 75th percentile line) are *not the results of an individual trial*. No single trial spent a long time at a low $\mu_{B \succ A}^{B \rightarrow A}$. The proper interpretation is that, at each point in time, the worst case at that point in 50 trials (i.e. the 2nd percentile) was poor. The identity of the worst-performing trial changed from time to time.

Nonetheless, the overall picture is positive — most trials converged to the same optimal policy. Over time, an increasing proportion of trials were close to optimal. Given sufficient time, the system will converge.

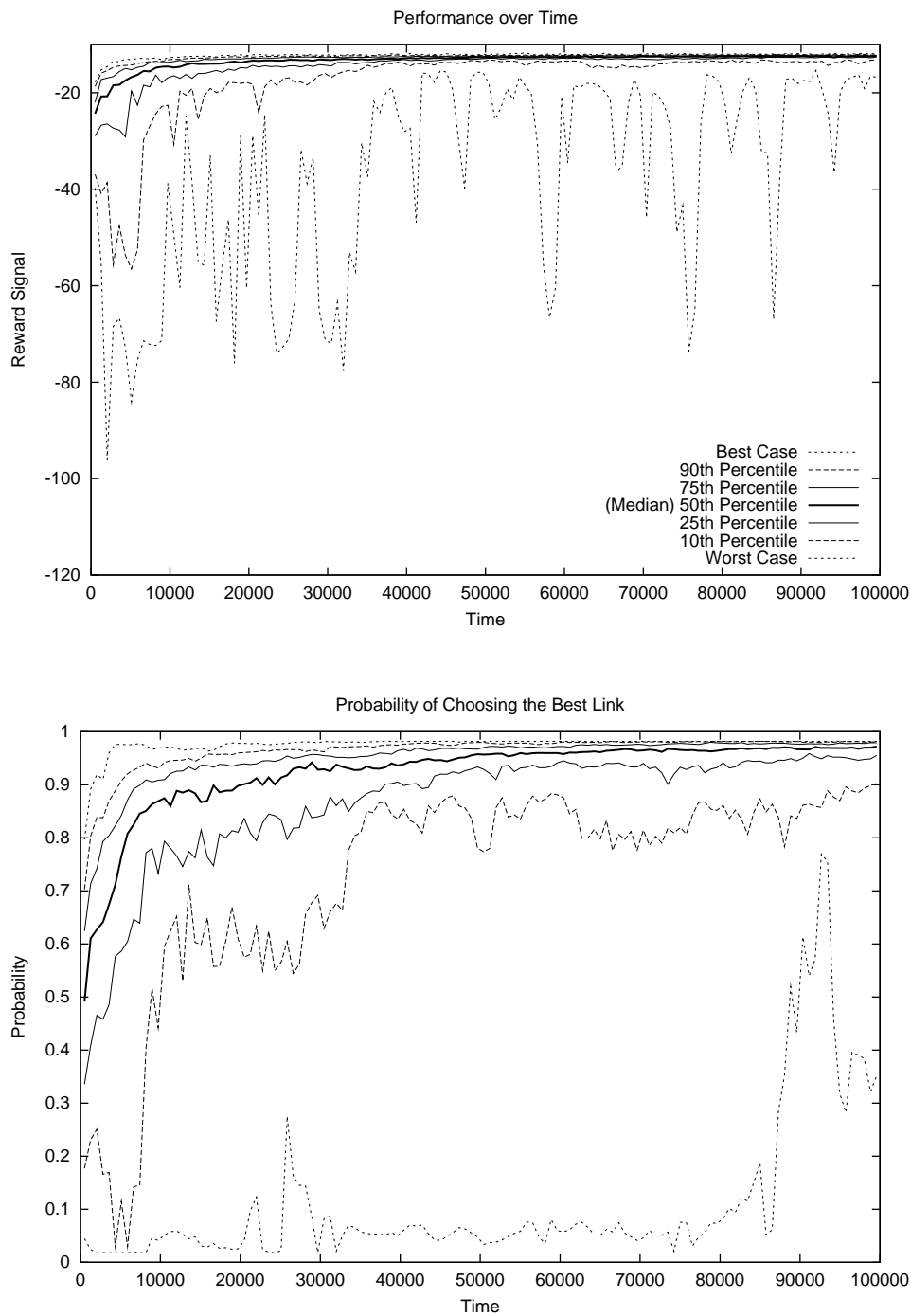


Figure 5.3: Reward signal r and probability $\mu_{B \succ A}^{B \rightarrow A}$, averaged for 50 independent trials.

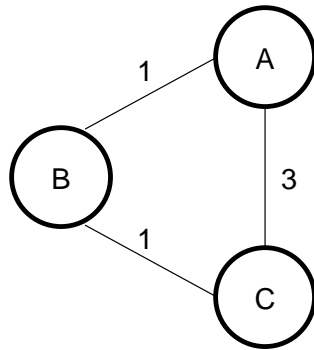


Figure 5.4: The triangular 3-node network.

5.1.3 Co-ordinating Agents (the Triangular 3-Node Network)

Figure 5.4 represents a more complicated network, where the shortest path from A to C is not via the direct link, but rather through B . Like the previous simulation, the traffic rate was one packet per node per time step, and the Time-To-Live was set at 70. This time, the expected reward under an optimal policy is $-\tau_A - \tau_B - \tau_C = -1.5 - 1 - 1.5 = -4$.

Again, $\vec{\theta}$ was initialized to zero, and $\beta = 0.99$ and $\gamma = 10^{-5}$. The top graph in figure 5.5 shows the successful learning process — the improvement in global performance towards optimum. The bottom graph shows the probability of the agent $A \leftrightarrow C$ choosing the link $A \asymp B$. Initially, the agent prefers the link $A \asymp C$ (the direct path), but eventually prefers the $A \asymp B$ link (leading to the shortest path). This is as to be expected, since initially the direct path (for a trip time of 3 with certainty) is preferable to the indirect path via B , which may route incorrectly. However, as $B \leftrightarrow C$ learns, packets sent from A to B are reliably forwarded on the link $B \asymp C$. The indirect path $A \asymp B \asymp C$ becomes a better option than the direct path $A \asymp C$.

Note that in this case, optimal behavior requires co-ordination: the agent $A \leftrightarrow C$ requires the $B \leftrightarrow C$ to adapt its behavior before the former learns the best policy. Thus, PGRL routing agents can learn to co-ordinate their behavior.

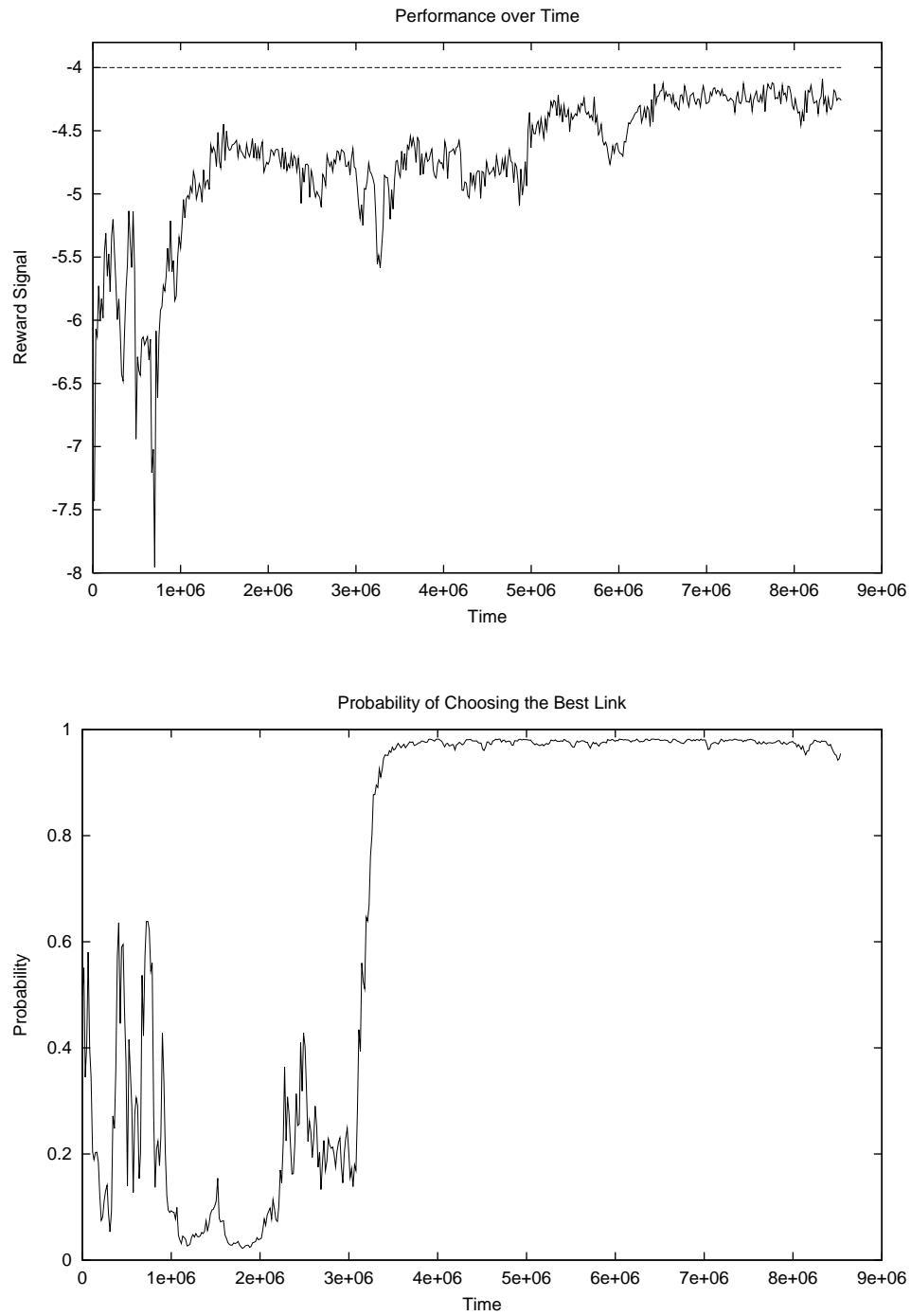


Figure 5.5: Reward signal r and probability $\mu_{A \succ B}^{A \leftrightarrow C}$ for the triangular 3-node network.

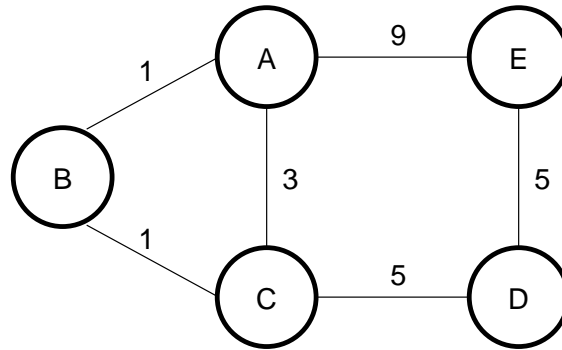


Figure 5.6: The 5-node network.

5.2 Performance Heuristics

5.2.1 Signal Decomposition (the 5-Node Network)

Figure 5.6 depicts a larger network. Note that the subgraph (A, B, C) is exactly the triangular three-node network discussed previously. We again used $\beta = 0.99$ and $\gamma = 10^{-5}$, and the results are given in figure 5.7. The top chart shows the reward signal, and the bottom chart shows the probability of node A choosing the best link when routing to C .

In this case, the reward signal fluctuates wildly, and the agent’s policy (bottom chart) is clearly not settling on a convergence point. A closer inspection reveals that the size of the reward signal is around an order of magnitude greater than the triangular three-node case, and presumably the variance in the reward signal is greater. Consequently the choice of $\gamma = 10^{-5}$ may be too large to allow the gradient estimate to converge.

Figure 5.8 shows a second run on the same network, this time with $\gamma = 10^{-7}$. Clearly, performance is much better, with the system approaching optimal performance. The smaller γ is needed because of the increased noise in the system. As the number of agents increases, the effect of one agent’s actions becomes smaller compared to the combined effects of all other agents’ actions.

An alternative approach is implement reward signal decomposition (§4.3.3). The results are shown in figure 5.9, which appears to converge a little faster than the previous case ($\gamma = 10^{-7}$). Note that in this simple network, there are no side-effects, and so reward signal decomposition will not affect which policy is optimal.

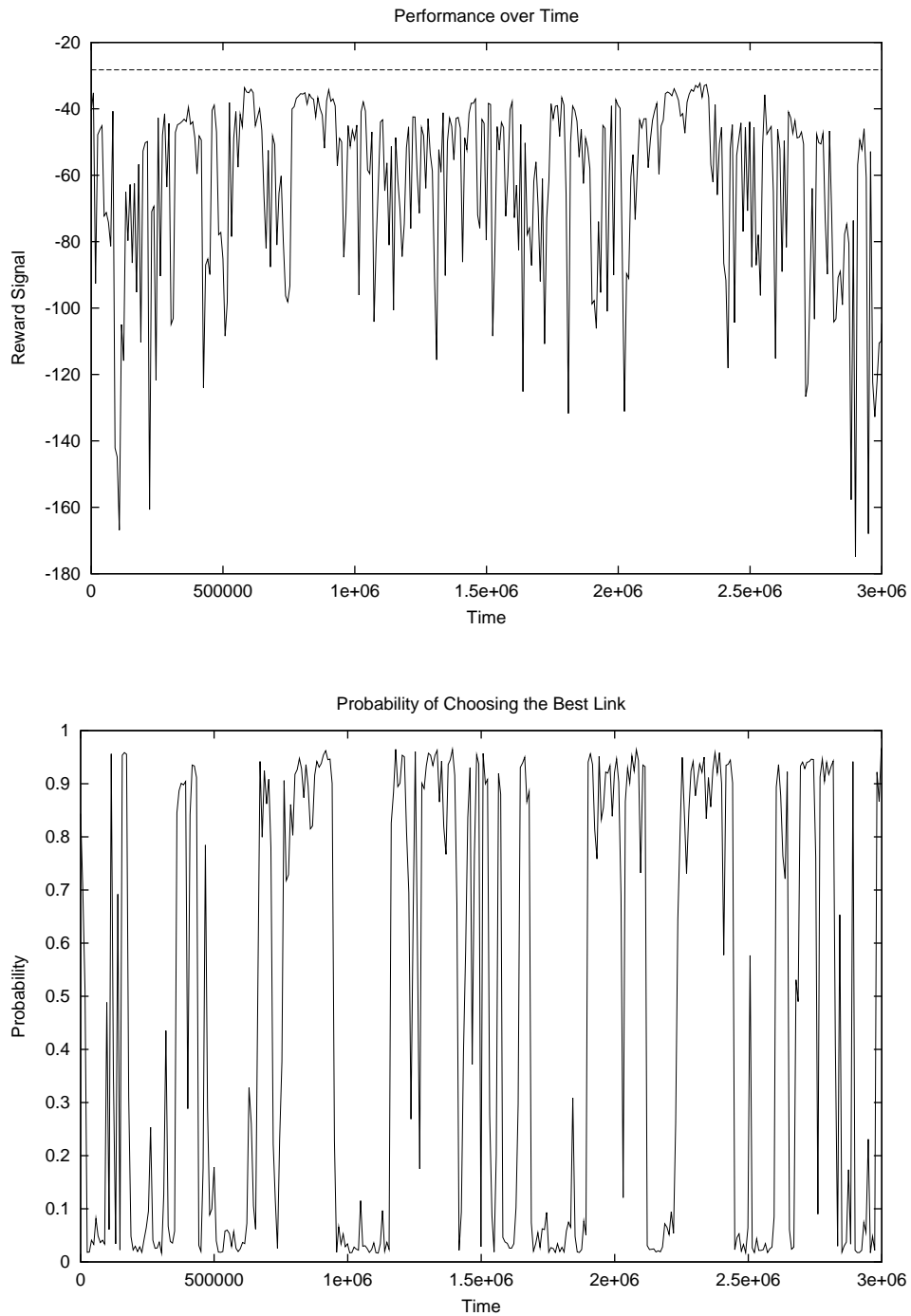


Figure 5.7: Reward signal r and probability $\mu_{A \approx B}^{A \leftrightarrow C}$ for the 5-node network ($\gamma = 10^{-5}$).

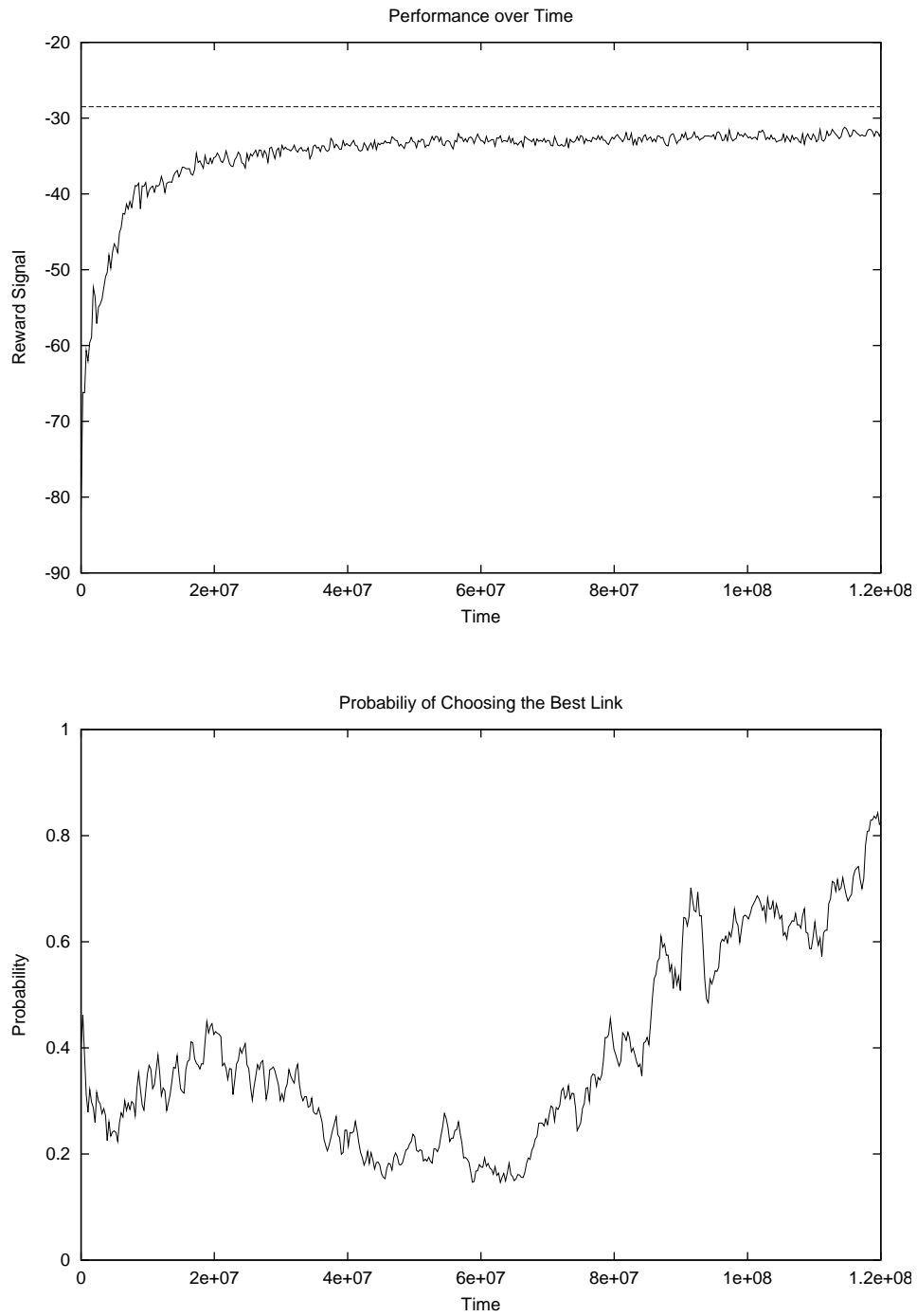


Figure 5.8: Reward signal r and probability $\mu_{A \approx B}^{A \leftrightarrow C}$ with a smaller $\gamma (= 10^{-7})$.

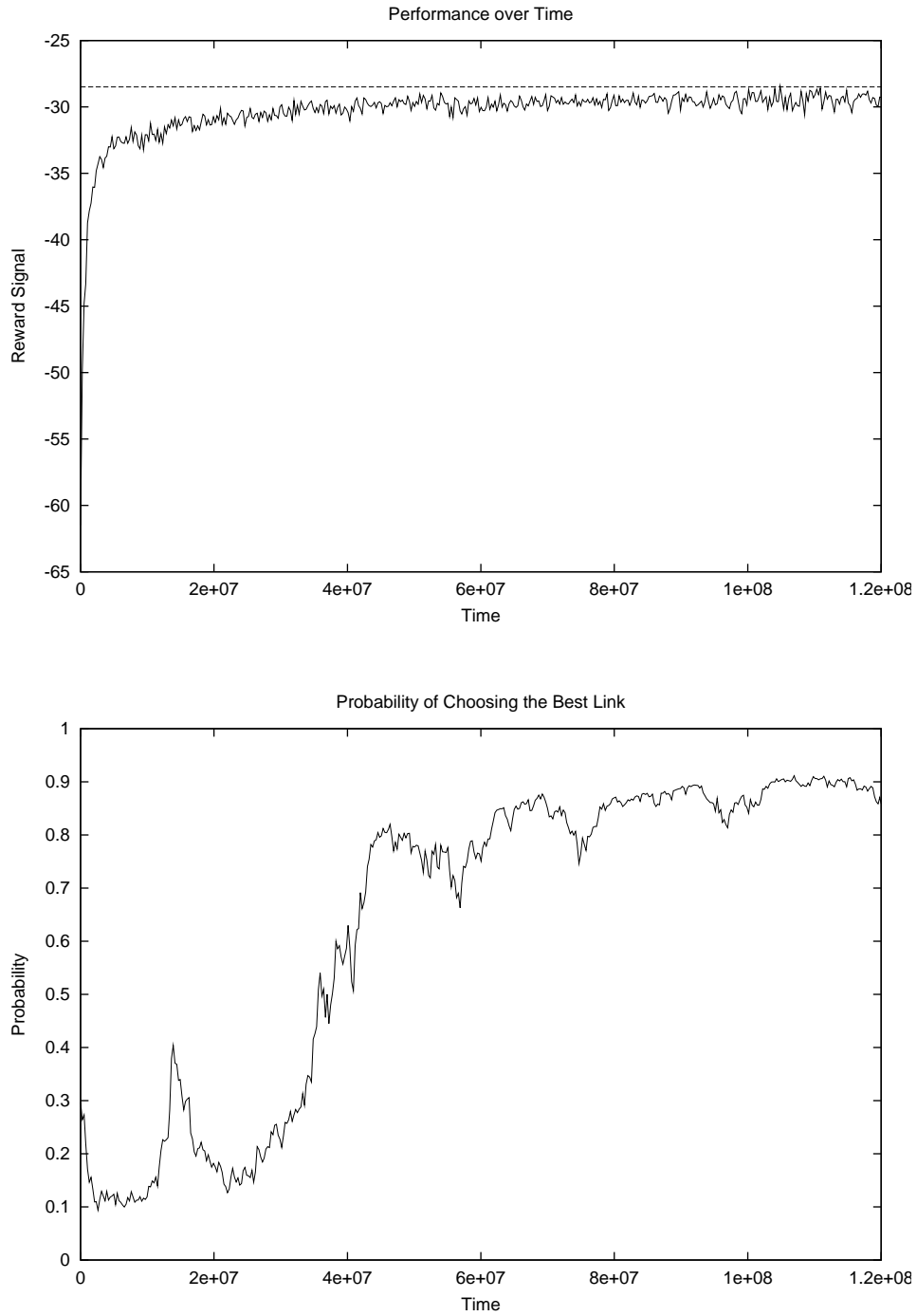


Figure 5.9: Reward signal r and probability $\mu_{A \succ B}^{A \rightarrow C}$ with signal decomposition ($\gamma = 10^{-5}$).

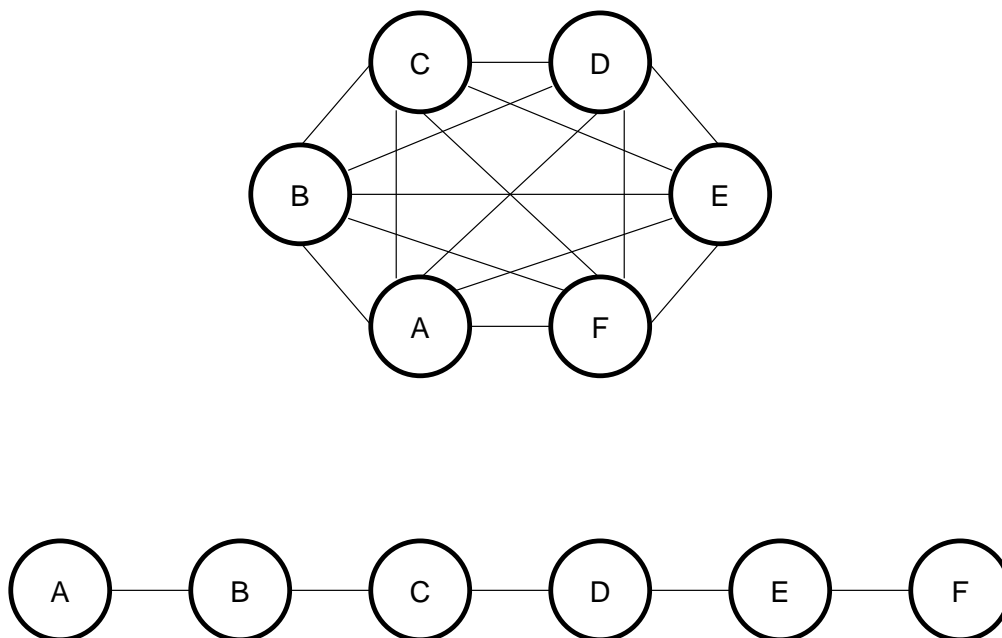


Figure 5.10: Two 6-node networks. All link costs are 1.

5.2.2 Cycle Detection (two 6-Node Networks)

If there are no side-effects of placing a packet on a link, then no shortest path can contain a cycle. A 2-cycle (e.g. $A \asymp B \asymp A$) is easy to detect, and a global penalty can be issued on detection (§4.3.4).

The impact of penalizing 2-cycles was examined for the two 6-Node networks shown in figure 5.10. The cycle penalty was set to -100 , with β set to 0.9 , γ set to 10^{-6} , and the TTL set to 10 . The difference between the actual performance measure (the negative of the total trip times of all arriving packets in a time step) and the reward signal (the performance measure plus the cycle penalties) is demonstrated in figure 5.11. Over time, the difference between the two lines narrows, indicating that penalties are issued less often. It appears that the system rapidly learns to avoid those behaviors which trigger penalties.

The next graph (figure 5.12) shows the difference in performance only, with and without cycle penalties. Clearly, convergence is much faster with the introduction of the cycle penalty. Measuring convergence speed as the number of time steps needed to achieve an average performance of -15 suggests a speed-up factor of around 400 .

The performance improvement is more dramatic with the linear network, with the results shown in figure 5.13 on two horizontal scales. In this case, convergence is around 2000 times faster, measured by the time to achieve an average performance of -25 . The higher speed-up factor for the linear network is easily explained. Penalties

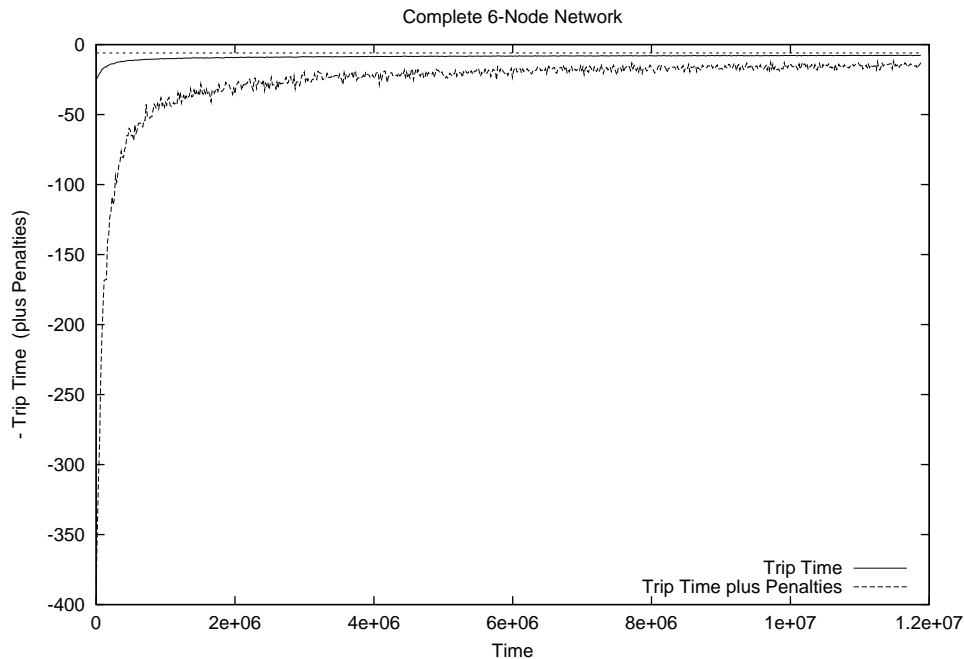


Figure 5.11: Adding a penalty signal to the performance measure.

for 2-cycles are much more likely to occur in the linear network (with each node having at most two links, and only one non-cyclical path from any two nodes) than in the complete network (with each node having six links, and numerous non-cyclical paths from any two nodes). Cyclic behavior is therefore penalized more often in the linear network.

Note that the penalty term is a global signal, which does not identify the agent or agents which acted poorly. A number of agents contribute to the path taken by a packet, and all agents are penalized even if only one of them makes a wrong choice. Nonetheless, the introduction of a global penalty signal was very effective, even without an explicit credit assignment procedure.

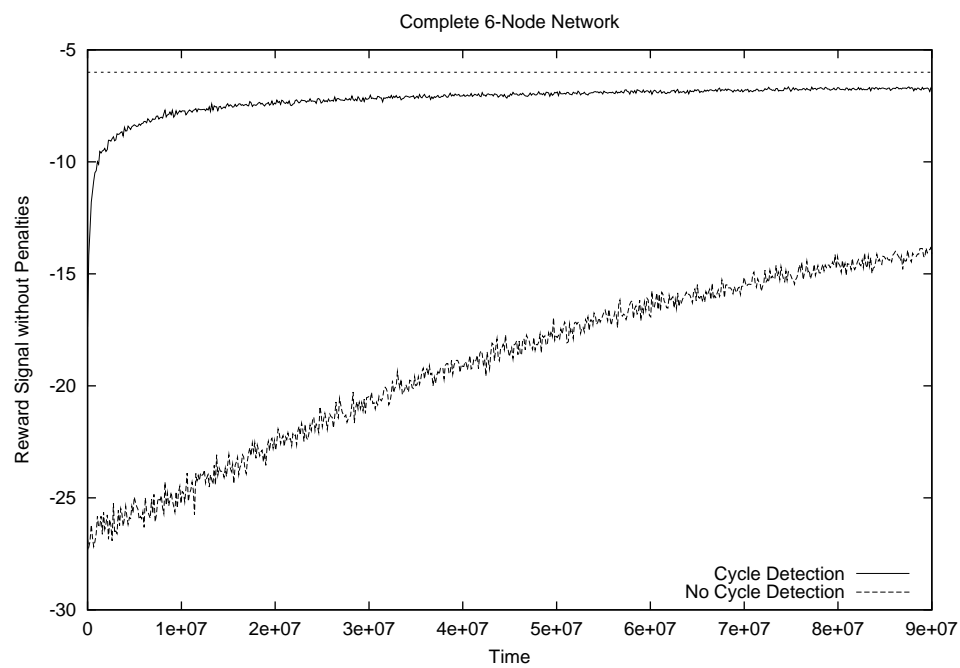


Figure 5.12: Performance gain in the complete 6-node network.

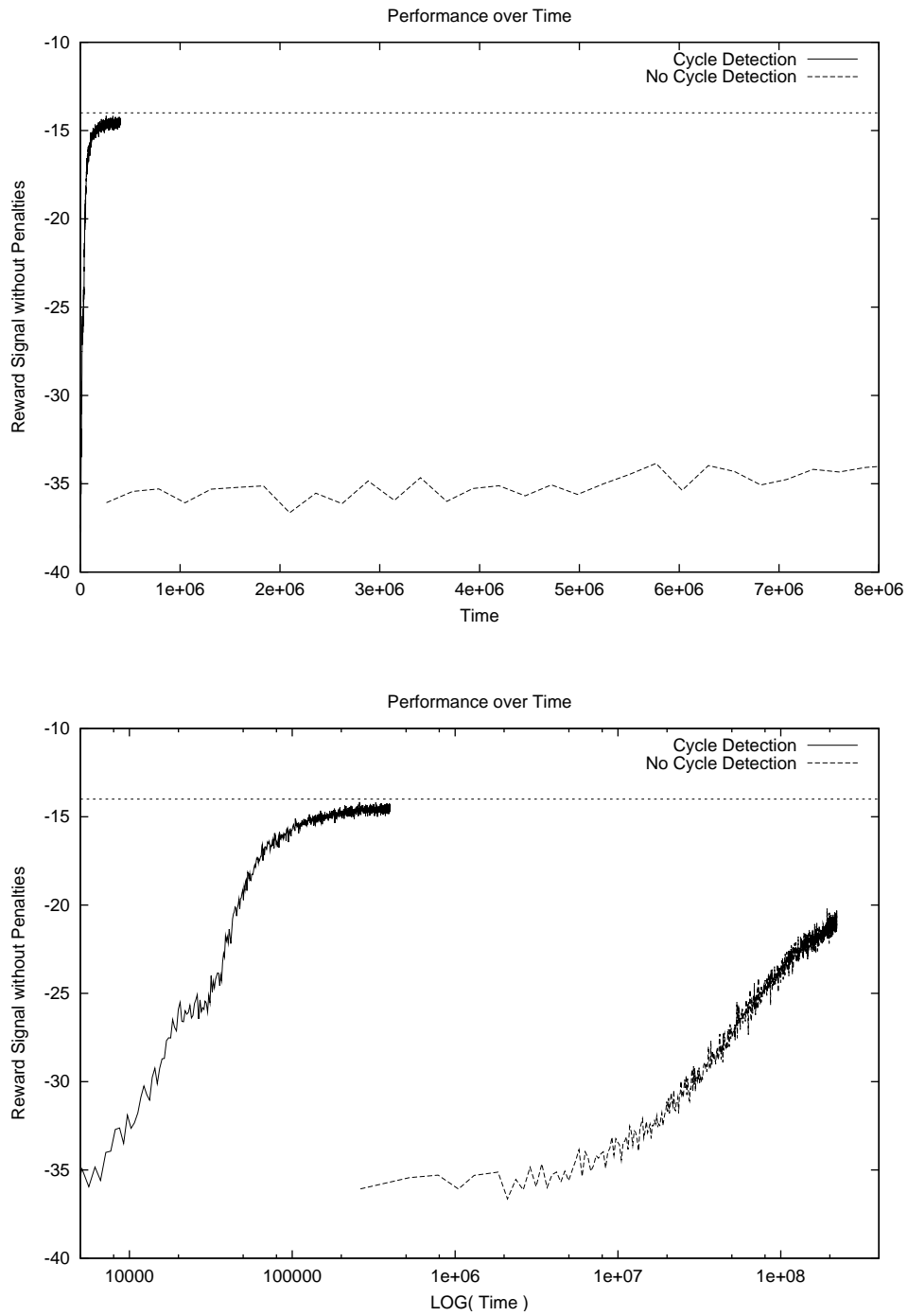


Figure 5.13: Performance gain in the linear 6-node network. Note the logarithmic scale on the second chart.

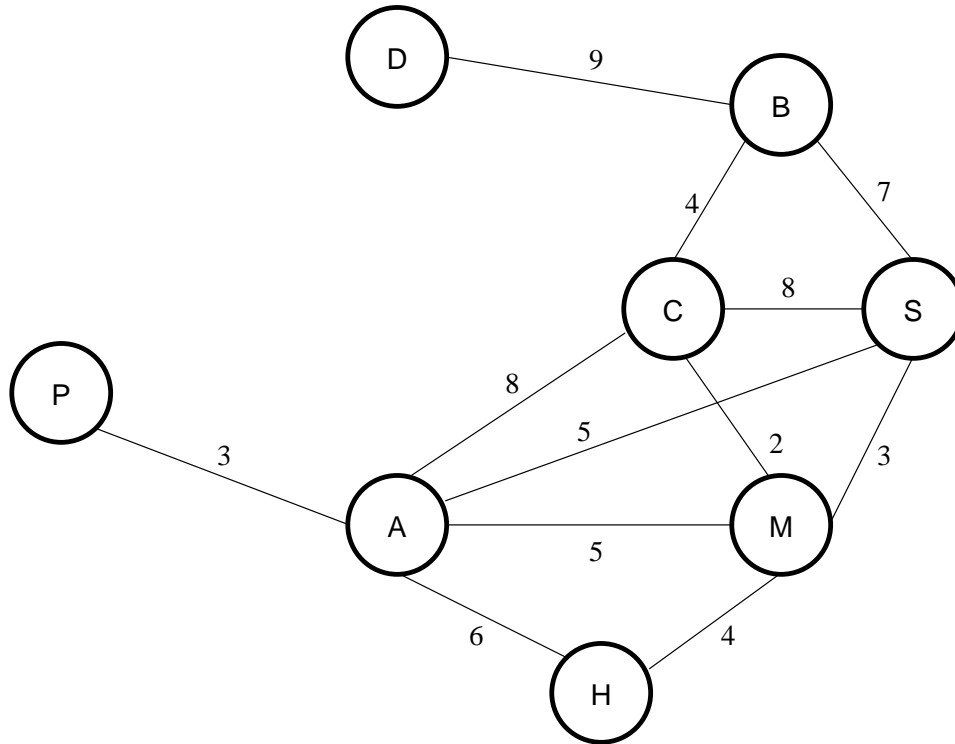


Figure 5.14: The 8-node network.

5.2.3 Scalability (the 8-Node Network)

Figure 5.14 depicts the biggest network we considered, whose topology is a loose interpretation of Australian capital cities. This network has 8 nodes, each having 7 agents, and the average node degree is 3. This gives $8 \times 7 \times 3$ parameters, or a parameter space with 168 dimensions.

Nonetheless, by incorporating both performance heuristics described earlier, clear performance improvement was observable (see figure 5.15) within a reasonable length of time (with $\beta = 0.99$, and $\gamma = 10^{-8}$). The underlying performance measure (i.e. reward signal without penalties) was plotted on two vertical scales. The first scale demonstrates improvement over time, on average. The second scale demonstrates that the performance improvement rate slows markedly, and the noise in the reward signal becomes large compared to the trend. It appears that the easier decisions are made relatively quickly, such as not to route from *B* to *S* via *D*. However, more subtle improvements, such as routing from *A* to *C* via *M*, rather than the direct link, are a lot harder to find.

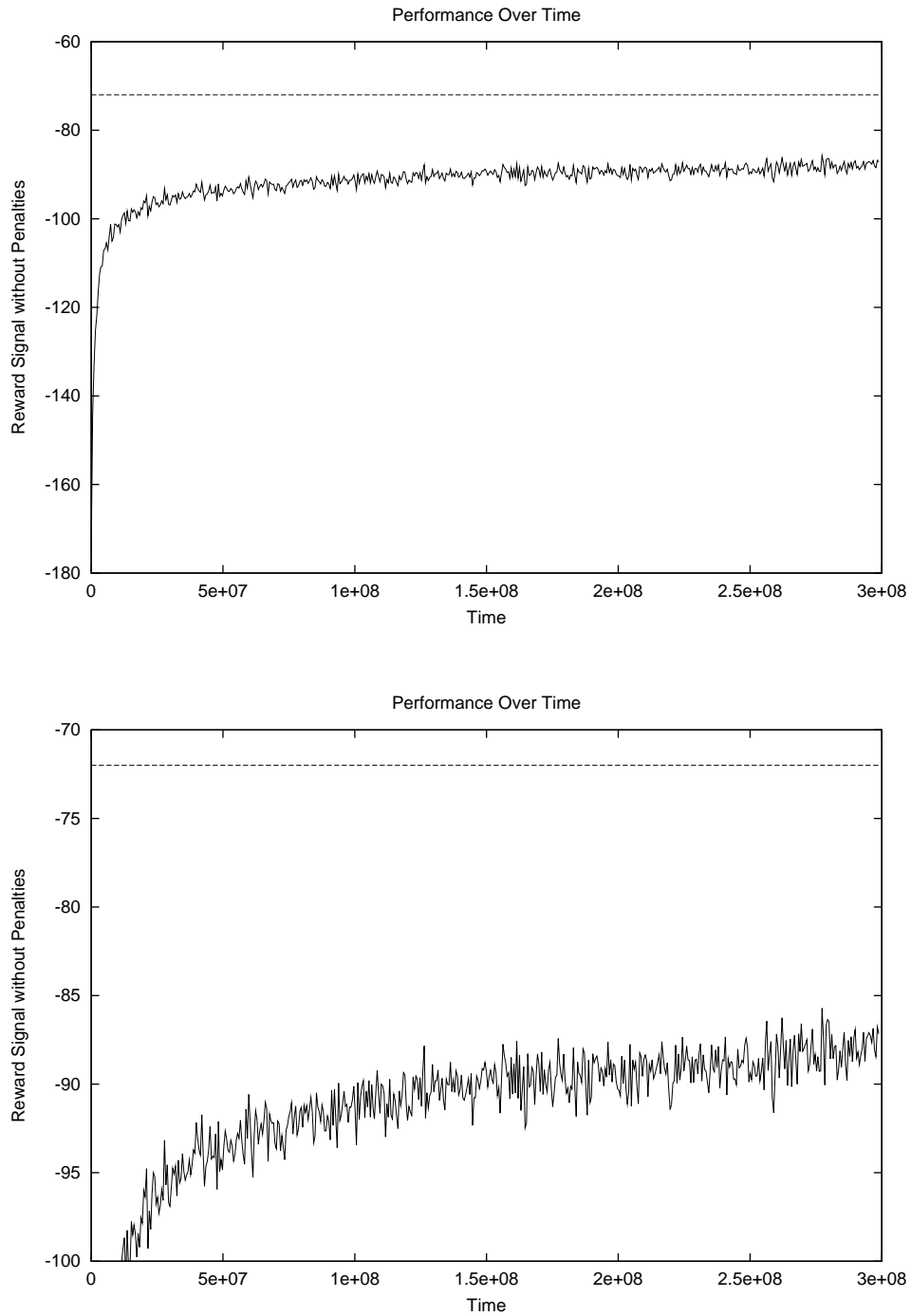


Figure 5.15: Reward signal r in the 8-node network. Note the two vertical scales.

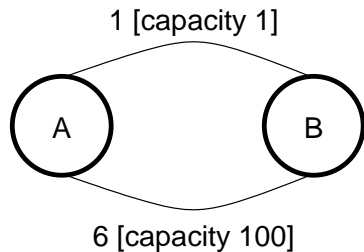


Figure 5.16: The 2-node contention network.

5.3 Resource Contention and Side-Effects

5.3.1 Mixed Strategies (the 2-Node Network)

Figure 5.16 represents a contention network. In this simulation, each link can only carry as many packets per time step as its capacity. When a link has reached its capacity, any extra packets are simply dropped, and a penalty term, *penalty*, is added to the reward signal.

In this simulation, we set the node *A* to generate 2 packets to *B* per time step, and no traffic to be generated at *B*. Note that regardless of which link was taken, every packet will arrive at its destination (i.e. *B*) in exactly one hop. In order to present a simple example (see the analysis below), we set *penalty* equal to -21 .

In one time step, there are 2 packets at *A* to be routed. If they are both put on the shorter link, one of those packets is dropped, and the other takes 1 time step to arrive, leading to a reward signal of $(-1) + (-21) = -22$. If each link takes one packet each, then no packets are dropped, and the reward is $(-1) + (-6) = -7$. If both packets take the longer link, then the reward is $(-6) + (-6) = -12$.

If the probability of *A* routing a packet on the shorter link is denoted μ , then the expected reward signal from the two packets generated per time step is:

$$\begin{aligned}\eta &= \mu^2 \cdot (-22) + 2 \cdot \mu \cdot (1 - \mu) \cdot (-7) + (1 - \mu)^2 \cdot (-12) \\ &= (-20) \cdot \mu^2 + 10 \cdot \mu - 12\end{aligned}$$

The optimal μ is therefore $\frac{-10}{2 \cdot (-20)} = \frac{1}{4}$, and at that policy, $\eta = -10\frac{3}{4}$.

A naïve Shortest Path (SP) router always places packets on the shortest path, measured by link delay. In this case, the shortest path from *A* to *B* is the top link, which will hold one packet (trip time 1) and drop one packet (penalty -21). Consequently, the SP router would achieve an average performance of -22 .

The parameters of the OLPOMDP router was set at $\beta = 0.99$ and $\gamma = 10^{-7}$. The performance is charted in figure 5.17. In this case, policy-gradient routing has outperformed the naïve SP router, due to the the latter's model's assumptions being broken.

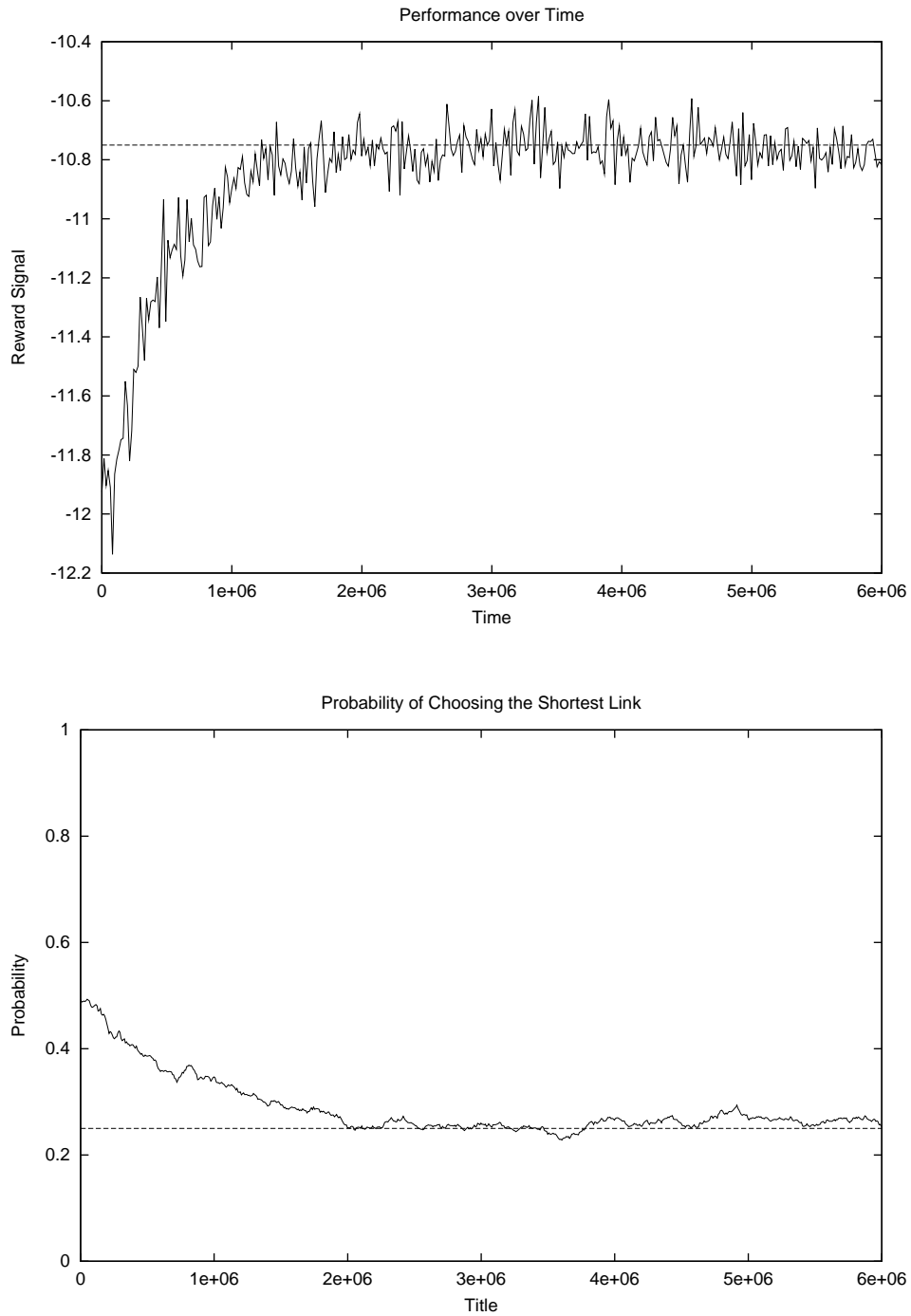


Figure 5.17: Reward signal r and probability μ_{short_link} for the 2-node contention network.

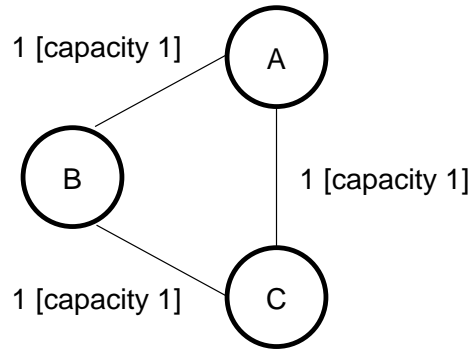


Figure 5.18: The 3-node contention Network.

5.3.2 Multiple Optima (the Triangle Network Revisited)

Appendix C describes a system (shown in figure 5.18) where multiple optima can occur — situations in which each agent is pursuing the best policy it can, assuming that all other agent’s policies are fixed. The basic cause is the limited bandwidth — packets pushed onto a link in use will be dropped, which is worse than taking an unused, but longer route. A situation can occur when all traffic from node A take the wrong route, if no single agent can change its policy without causing packet loss.

The results of two separate runs in applying the PGRL router to this network are shown in figure 5.19. In this simulation, the routers were set with $\beta = 0.9$, and $\gamma = 10^{-5}$, with the packet $TTL = 10$, and the packet loss penalty set at -10 per packet dropped.

Clearly, the two runs converge to a different equilibrium, and one of these is not globally optimal. Thus, it is possible for PGRL routing to fail to learn the best policy.

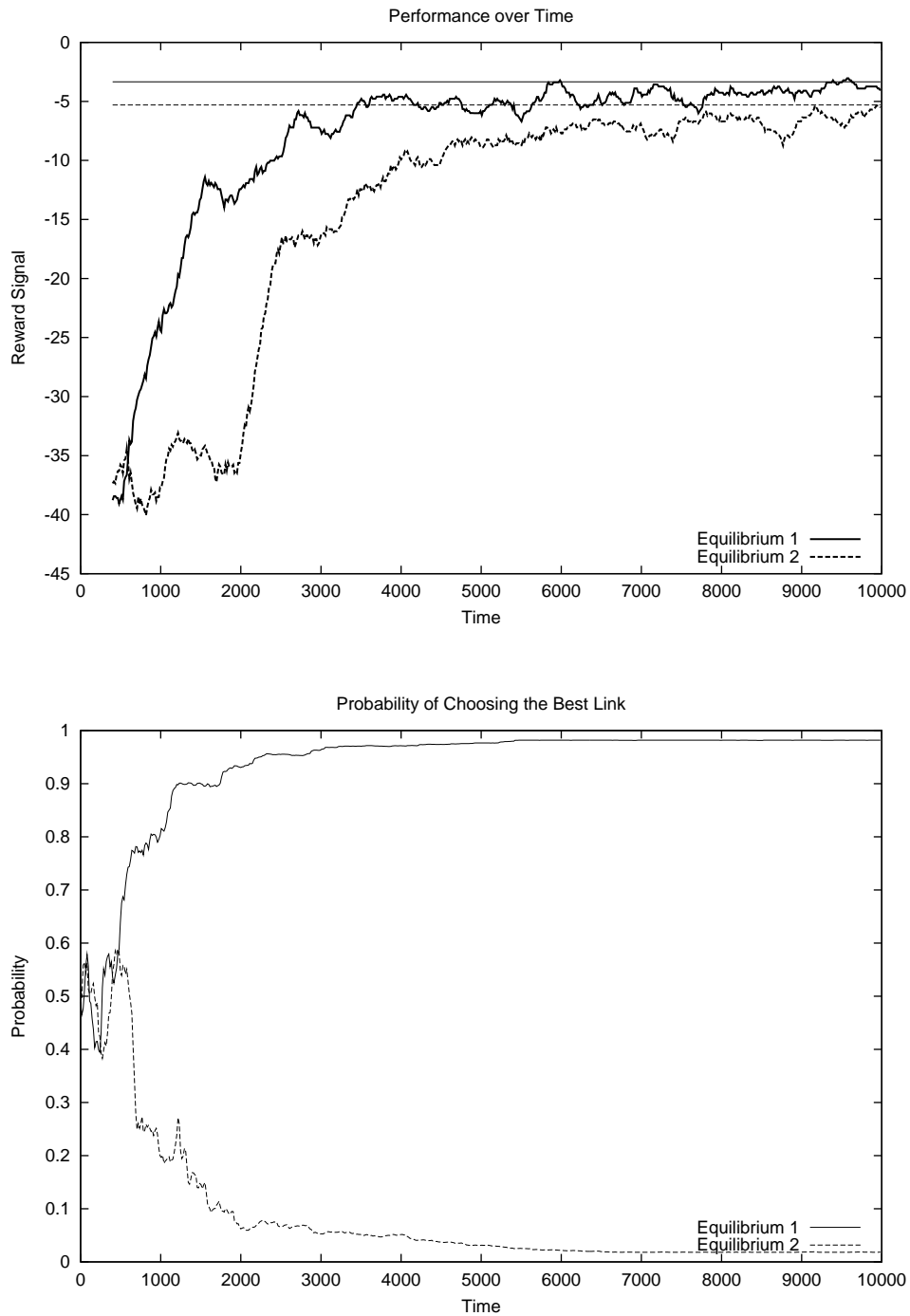


Figure 5.19: Reward signal r and probability $\mu_{A \succ B}^{A \rightarrow B}$ for two separate runs in the 3-node contention network.

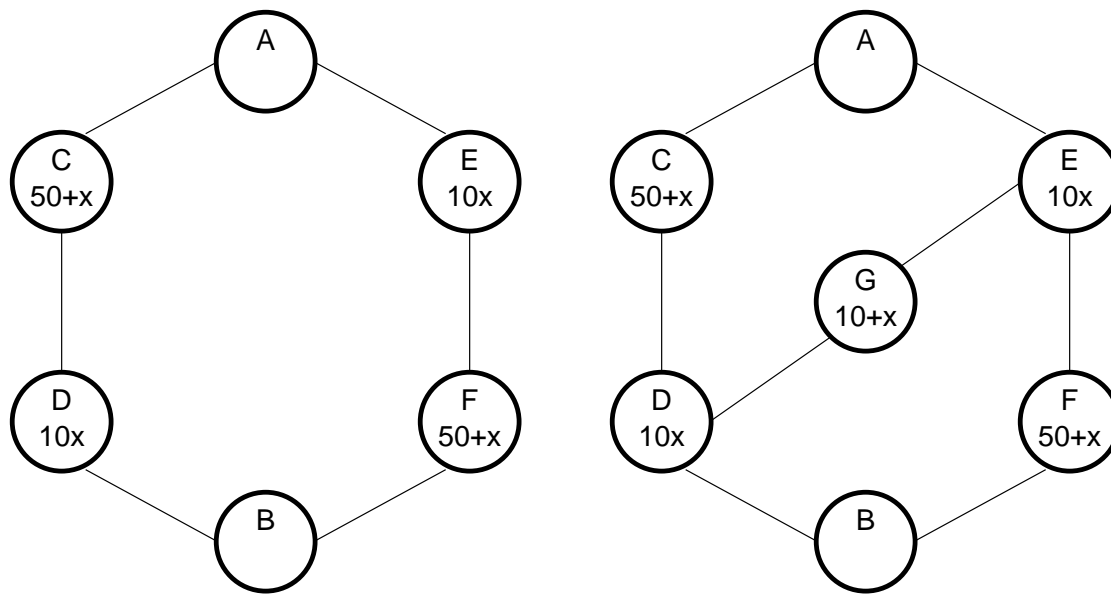


Figure 5.20: Braess' Two Networks

5.3.3 Braess' Network Paradox

Figure 5.20 depicts an example of Braess' paradox as given by Tumer and Wolpert [1999]. The network model in this case differs from previous experiments, in that costs are incurred at nodes, rather than at links. The formulae marking the intermediate nodes are the cost per packet of passing through that node, where x is the flow through that node. This model is motivated by traffic or commodity flows, such as road networks, or oil pipelines, where costs are dominated by queueing times caused by congestion. Cohen and Kelly [1990] give an example of Braess' paradox in the context of continuous time queueing networks, which is more appropriate for Internet routing. Nonetheless, the traffic flow model is sufficient to demonstrate the effectiveness of PGRL.

Denote the left-hand and right-hand networks as *Braess0* and *Braess1* respectively. In *Braess0*, if one packet per time step goes down the left hand path from *A* to *B*, then the cost per packet is $(50 + 1) + (10 \times 1)$, so the total cost is 61. If the packet rate doubles, the cost per packet is $(50 + 2) + (10 \times 2)$, and at two packets per time step, the total cost is $2 \times 72 = 144$.

If two packets are to be sent from *A* to *B*, then routing both down one track (either the left track or the right track) incurs a cost of 144. However, routing one down each track will cost twice 61, or 122. Similarly, at 6 packets per time step, the best policy is to route three down the left and three down the right, at a cost of 83×6 .

Now consider *Braess1*. Note that one node and two links have been added, and

any routing strategy for *Braess0* remains a valid strategy for *Braess1*. Consequently, at 6 packets per time step, it is possible to route at a cost of 83×6 .

However, if packets take the shortest cost route to *B*, then the diagonal path allows packets at *E* (the top right $10 \times x$ node) to avoid the costly node *F*. In fact, at 6 packets per time step from *A* to *B*, equilibrium is reached only when 2 packets go down each route: left, diagonal, and right. However, this leads to a cost for going down the left track of $(50 + 2) + (10 \times 4)$, or 92. Similarly, the cost for the right hand track is 92. Finally, the cost for the diagonal track is $(10 \times 4) + (10 + 2) + (10 \times 4)$, or 92.

Here is the paradox: by adding an extra route from *A* to *B*, the (individually-greedy) equilibrium cost goes up from 83×6 to 92×6 .

In this simulation, the link delays were set to 1. The actions taken by PGRL routing (with parameters $\beta = 0.99$, $\gamma = 10^{-5}$) is charted in figure 5.21. Both charts are probabilities for choosing links. The first chart is at *A* for taking the left link $A \asymp C$ rather than the right link $A \asymp E$. The second chart is at *E* for taking the downward link $E \asymp F$ rather than the diagonal link $E \asymp G$.

Over time, the system hovers around 50% for the first probability, and 100% for the second one — i.e. equal traffic flowing down the left and right tracks, with nothing going down the middle. Thus, PGRL routing has discovered the globally optimum policy and has avoided the paradox.

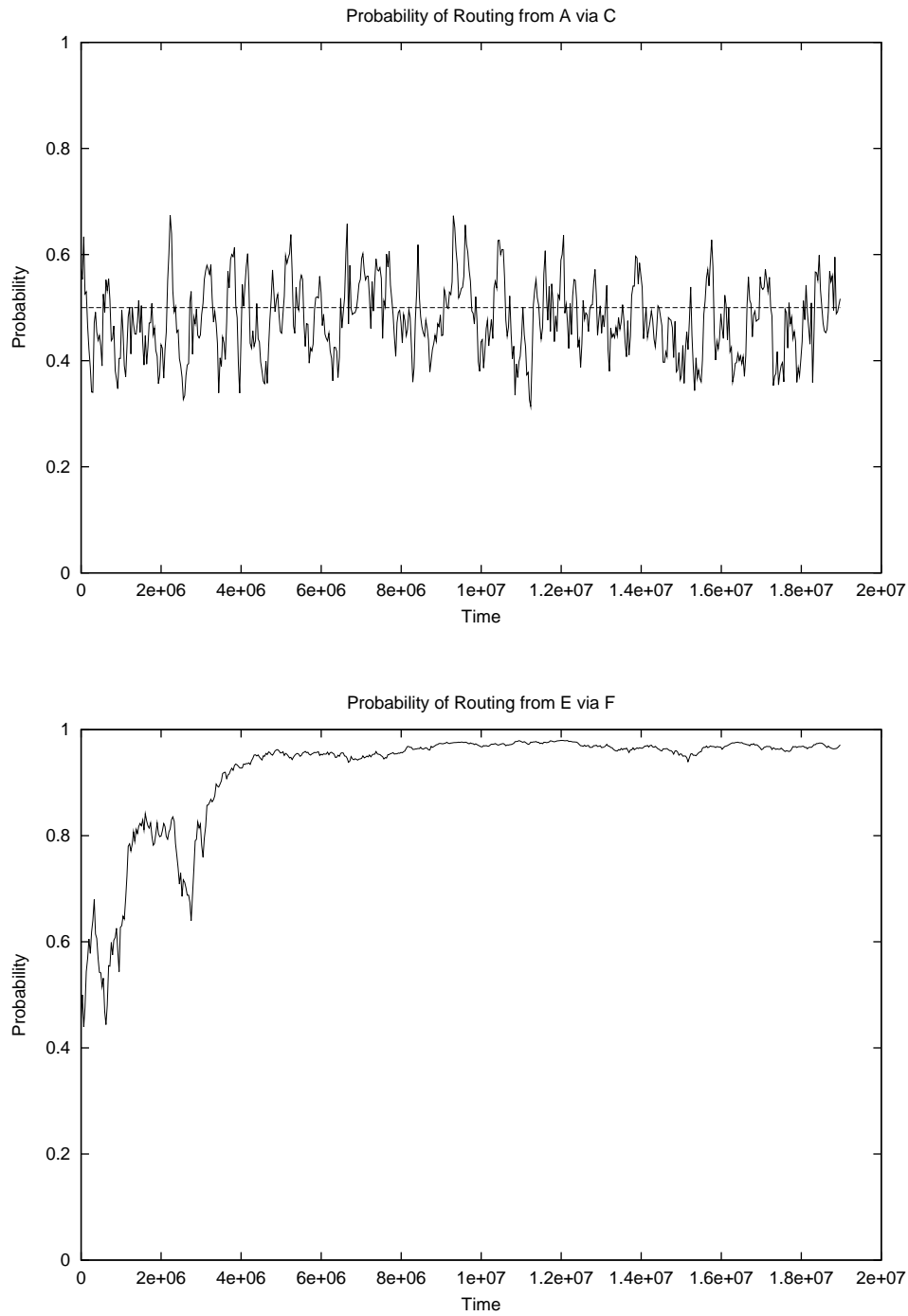


Figure 5.21: Probabilities $\mu_{A \succ C}^{A \curvearrowright B}$ and $\mu_{E \succ F}^{E \curvearrowright B}$ in Braess' second network.

5.4 Distributed Networks

5.4.1 Broadcast of Reward Signal Components

All experiments so far have involved the global reward signal magically sent to each agent at the same time. In a distributed system, it is impossible to communicate instantaneously. If a node receives a packet destined for it, a message noting its trip time (and thus its impact on the global reward signal) has to be delivered to all other agents.

The natural communication mechanism is the network itself - packets containing reward signal data can be sent to other nodes on the links connecting them. In fact, since all agents are to be informed of the reward message, that packet may be broadcast, rather than routed point-to-point. A broadcast to all nodes is implemented simply via flooding. In this case, the first time a node receives a (time-stamped) reward packet, it duplicates it on all other links (except the one it came on). Thus, the time lag between when a packet arrived at its destination X , and when a node Y receives the reward signal, is just the length of the shortest path from X to Y .

Figure 5.22 shows the performance of the linear three-node network (described above), with two reward distribution methods: instantaneous, and the more realistic broadcast. The system in the instantaneous case performs better in the short term, since the association between action and consequence is stronger. However, over time, both systems are still able to discover the long term implications of various actions, and both converge on the same optimum policy. The qualitative behavior of the learning system is unaffected by the modeling of distributed reward sources.

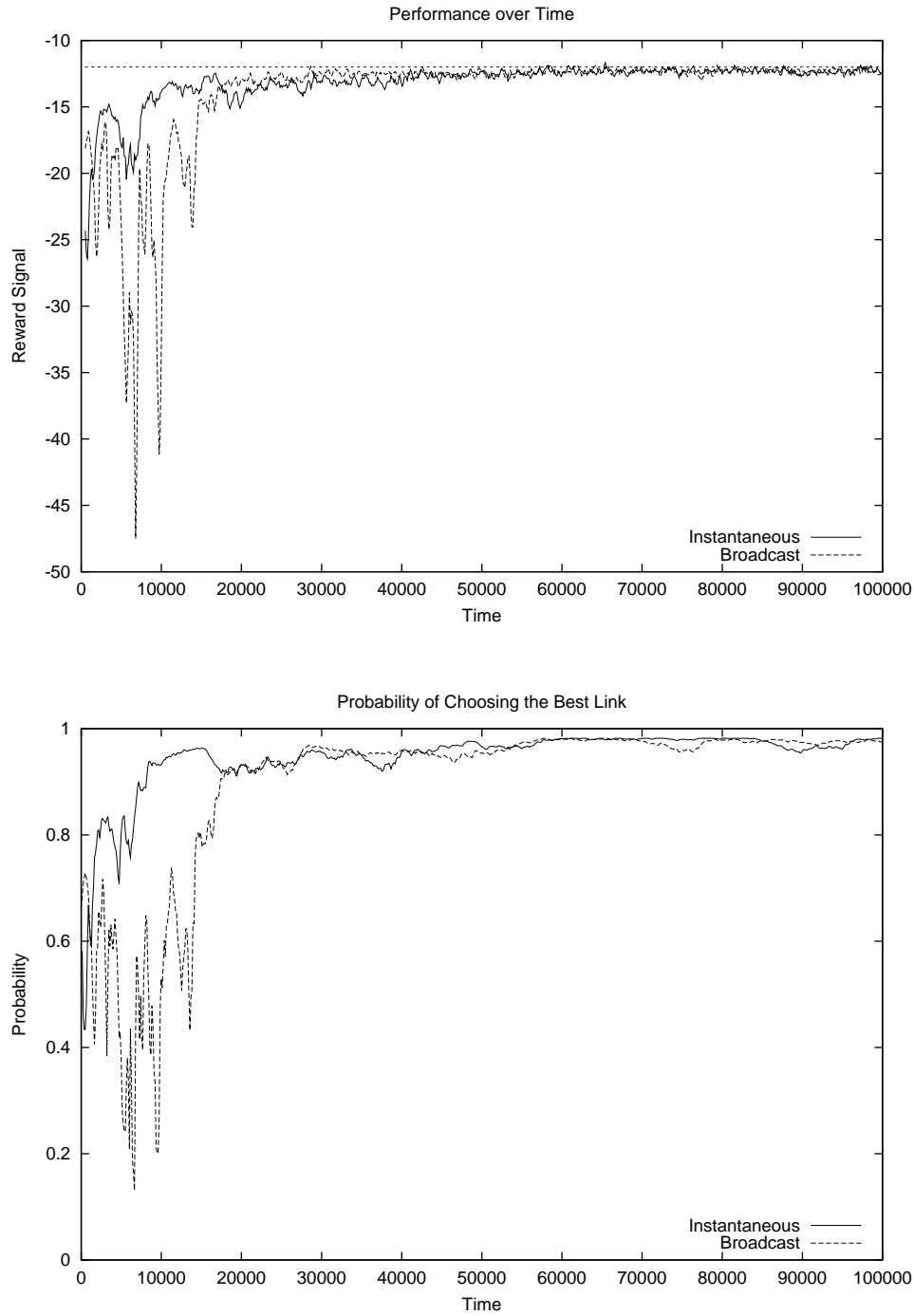


Figure 5.22: Reward signal r and probability $\mu_{B \succ A}^{B \rightarrow A}$ under different reward distribution schemes.

Discussion

There is a difference between knowing the path, and walking the path.

— *The Matrix*.

This chapter first addresses the arguments for and against the appropriateness of policy-gradient reinforcement learning for network routing, as raised in chapter 3. It then discusses the implications for PGRL and its applicability to various domains.

6.1 Was PGRL Appropriate for NR?

6.1.1 Robustness

In the simple networks, the Shortest Path algorithm finds the optimal policy. PGRL routing also converged to that optimum over time (§5.1, §5.2). When the network model was modified, so that the Shortest Path algorithm breaks down, then PGRL routing actually performed better (§5.3.1). Thus, PGRL methods were shown to be robust to modeling failures, and are applicable when the problem is difficult to model perfectly.

6.1.2 Guaranteed Performance Improvement

Almost every experiment in chapter 5 demonstrated that the performance of a PGRL routing system improves over time to an optimum. One exception is the first case in the 5-node network (§5.2.1), which was rectified by reducing the γ parameter. The other exception is the repeated case in §5.1.2, which still demonstrated that, *on average*, PGRL systems improve over time. Thus, PGRL methods were shown to guarantee performance improvement, on average, in a variety of problems.

6.1.3 Implicit Co-ordination

Braess' paradox demonstrated a situation in which agents pursuing individual incentives arrived at a globally sub-optimal equilibrium. The PGRL routing system avoided Braess' paradox (§5.3.3), and also solved the limited bandwidth problem (§5.3.1), where the choice to utilize the short link had the external effect of restricting link choices for other packets. Thus, PGRL methods were shown to incorporate all effects of an action, including externalities, and thus maximize the common good.

6.1.4 Distributable Computation

All the experiments in chapter 5 had no explicit communication between agents, except for a global reward signal. Each agent also made calculations based solely on variables local to that node, without the need for a shared memory between agents (except for the reward signal). Furthermore, the results in §5.4.1 demonstrated that PGRL was still effective when the global reward signal is localized and broadcast. Thus, PGRL methods were shown to be distributable.

6.1.5 Flexibility

The requirement that PGRL policies be probabilistic implies that they can represent a wider range of strategies than greedy algorithms, which only chooses the single “best” action. A greedy algorithm will fail when a mixed strategy is better, and the PGRL routing algorithm managed to find the right strategy in the 2-node contention network (§5.3.1) and in Braess' network (§5.3.3). Thus, PGRL methods were shown to be flexible in the class of solutions it could represent.

6.2 Was PGRL Inappropriate for NR?

6.2.1 Convergence Time

Each experiment in chapter 5 took a significant number of steps to converge. Even in the simple linear 3-node network (§5.1.1), convergence required around 10^5 time steps, compared to the network diameter of 6 time steps. Heuristics such as signal decomposition (§5.2.1) and cycle detection (§5.2.2) were shown to improve convergence time, and may be the only way to tackle larger scale problems in a reasonable time, as demonstrated by the 8-node network (§5.2.3).

The issue of convergence time means that the PGRL routing algorithm is not practical for Internet routing. Real world networks are dynamic — hosts go up, links go down, and components are upgraded. However, for other routing applications, such as airline scheduling, PGRL methods may be quite appropriate.

One of the contributing factors to long convergence times was the fact that the optimal solution (which the system was converging to) was at the extremes of the (θ -constrained) parameter space, with one link's probability being maximized, and all other links' probabilities minimized.

Furthermore, as the system approached an optimum, the reward signal improvement rate slowed down, as exhibited in almost every chart in chapter 5. An explanation is provided in figure 6.1. Consider the linear 3-node network again, and the agent $B \hookrightarrow A$. This agent has two parameters θ_1 and θ_2 . The update rules given in chapter 4, namely equation 4.5, imply that the updates are always equal and opposite, so that $\theta_1 + \theta_2 = 0$. The top chart in figure 6.1 shows the map from θ_1 to μ_1 , namely $\mu_1 = \frac{\exp(\theta_1)}{\exp(\theta_1) + \exp(-\theta_1)}$. As θ_1 approaches the boundary, then the sensitivity of μ_1 to θ_1 lessens, and a change in θ has an increasingly smaller effect on policy.

The bottom chart shows the dependence of η , the expected long-term average reward, on θ_1 , the probability of the $B \hookrightarrow A$ agent choosing the $B \asymp A$ link. This is calculated by determining the map from θ_1 to μ_1 , above, and then noting that the expected trip time from B to A is given by $\tau_A = \mu_1 \cdot 1 + (1 - \mu_1) \cdot (10 + \tau_A)$. PGRL methods attempt to modify θ_1 proportionally to the gradient of η with respect to θ_1 . The gradient near the optimal θ_1 (i.e. > 1.5), is a lot flatter than the gradient at the initial θ_1 (i.e. 0), and so performance improvement is slower as the system approaches the optimum.

One possible workaround would be to apply a monotonic transformation to the reward signal (e.g. an exponential) in order to straighten out the $\theta \mapsto \eta$ map, and so keep the convergence rate constant. Alternatively, different policy classes (rather than the normalized exponential) could be designed. For example, one simple class is $\mu_i = \theta_i$, with clipping to ensure that $0 < \mu_i < 1$, and normalization to ensure that $\sum_i \mu_i = 1$. A study of these ideas are left for future work.

In other application domains, the optimum solution may be in the center of policy-space, rather than at the boundary. In such a case, PGRL methods would presumably converge faster.

6.2.2 Multiple Local Optima

If a system has a number of local optima, then each one of those optima could be a convergence point for a PGRL system. This problem is exhibited in the triangular 3-node contention network (§5.3.2), in which the system sometimes converges to a sub-optimal policy. Nonetheless, if the system is repeatable (e.g. a simulation), then multiple runs can be made (possibly in parallel) for which the best convergence point can be taken.

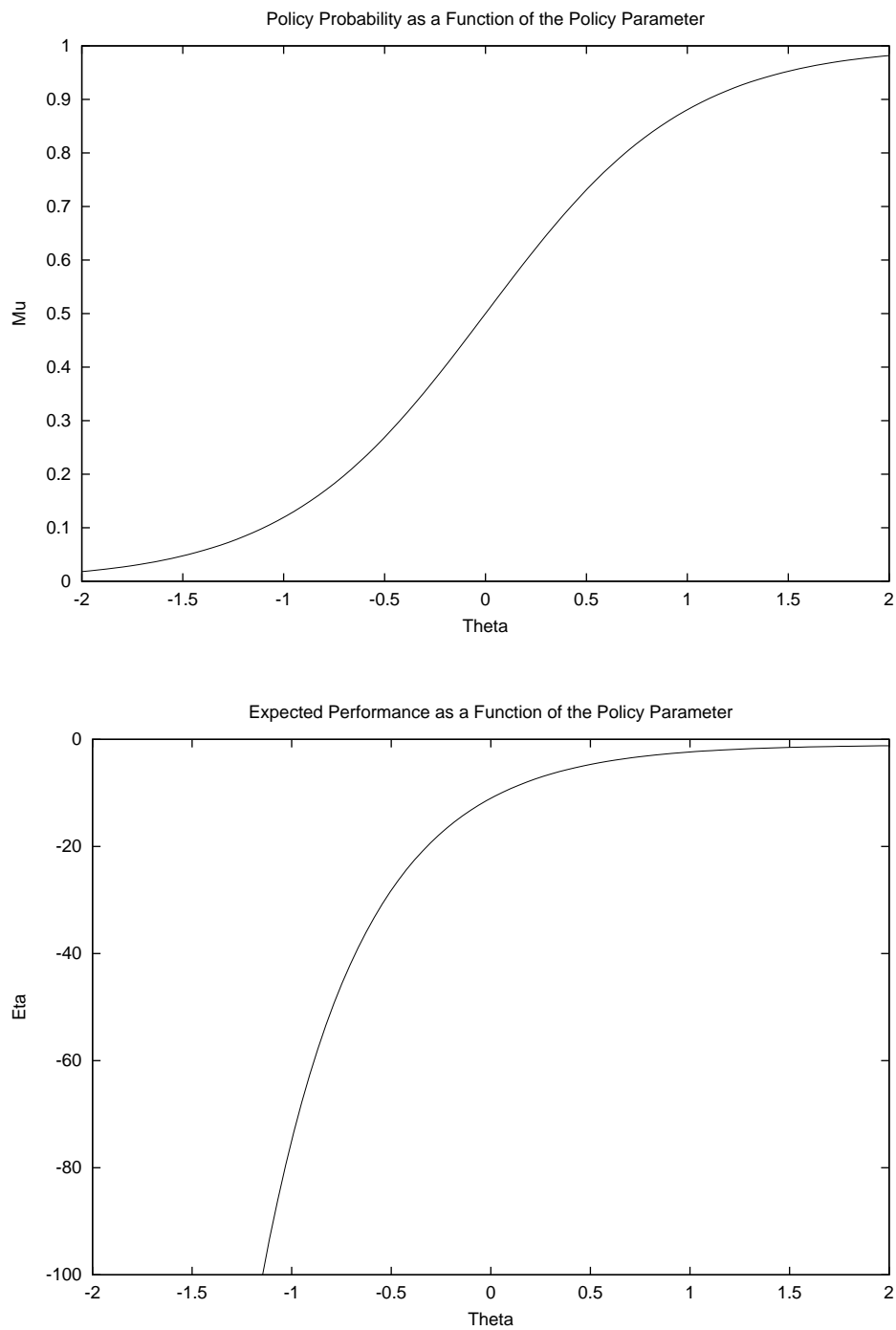


Figure 6.1: The maps $\theta \mapsto \mu$ and $\theta \mapsto \eta$ in the linear 3-node network.

6.2.3 Non-Determinism

PGRL methods are necessarily non-deterministic, since gradient methods require a smooth (i.e. differentiable) function from parameter space to actions, which need mixed strategies (i.e. each action having a non-zero probability of being chosen). If an action was chosen deterministically (e.g. always choose this link), then there would be no exploration of other actions, and so no feedback on their value.

As a consequence, in network routing, it is impossible to represent *the* best policy, since it usually involves choosing just the one link all the time. PGRL routing is able to approximate the best policy, in that the right link is almost always chosen. However, it is still possible that the wrong link is taken. In fact, it is even possible (although extremely unlikely) that the wrong link is chosen ten or a hundred times in a row. There is no worst-case performance guarantee for PGRL systems, as demonstrated in §5.1.2. In some network applications, such as multimedia streaming, worst-case trip time can be more important than average trip time. In these cases, PGRL routing is not appropriate.

However, if a non-deterministic policy is optimal (such as the 2-node contention network in §5.3.1 and Braess' network in §5.3.3), then PGRL can still be appropriate. Even in these cases, there can be better deterministic algorithms, which are not representable by PGRL. For example, in the 2-node contention network, the best policy is to deterministically route one packet on each link. In Braess' network, the best policy is to deterministically route three packets down each side, and avoid the diagonal route.

In other domains, a non-deterministic policy can be better than a deterministic one. For example, consider a competitive situation between a robot soccer player (striker), and a robot goal-keeper. The striker has the option to shoot for the left hand side of the goal, or the right hand side, and the goal-keeper can only block one side. If the striker always shoots left (or always shoots right), then the goal-keeper can devise a counter-strategy that is very effective, viz. always block left (or always block right). However, a mixed strategy (e.g. 60% left, 40% right by a right-footed striker), may be best from the striker's point of view.

6.2.4 Scalability

Scalability turned out to be an important issue with PGRL routing. The agents in the linear three-node network (§5.1.1) converged in around 10^5 time steps, whereas the five-node network (§5.2.1) required around 10^8 time steps, and the eight-node network (§5.2.3) was far from convergence by $3 \cdot 10^8$ time steps. There were at least two contributing factors to this problem.

First, a larger number of agents needed a smaller γ , or step size parameter, for convergence (§5.2.1). This was necessary to isolate the effect of a single agent's actions against the noise of the effects of all other agents' actions. A smaller step size implies a larger number of steps needed to reach the optimum policy (in this case, maximizing

θ for the best link, and minimizing θ for all other links). However, this problem can be tackled by reward signal decomposition (§5.2.1).

Second, as the network size increased, optimal paths required the co-operation of a larger number of agents. For the linear three-node network (§5.1.1), packets correctly routed by the agents at B arrived directly at their destination. However, for the linear six-node network (§5.2.2), for $B \hookrightarrow F$ to be rewarded for choosing the right link, all three agents $C \hookrightarrow F$, $D \hookrightarrow F$, and $E \hookrightarrow F$ also had to choose the right action, before the packet timed out. The greater the number of agents required to co-operate, the greater the chance of at least one agent choosing the wrong link. This increases the variance in the gradient estimate, and leads to longer convergence times.

6.2.5 Reward Signal Distribution

It was apparent that the extra delay in broadcasting reward components in a distributed system did not affect the qualitative behavior of the system (§5.4.1). The OLPOMDP algorithm can already handle a delay between an action and its consequence, and so the introduction of a further lag between an action's effect and the receipt of the reward signal does not break it.

6.3 Implications

Overall, the OLPOMDP routing algorithm (on page 31) is not the most appropriate technique for solving network routing problems. This is mainly due to the long convergence time, and poor scalability.

However, policy-gradient methods are still appropriate in a number of domains, especially when convergence time is not an issue. If a training period can be set aside before the system is deployed, a PGRL system can be quite effective. If the system is trained on a simulator, then multiple runs can be made, possibly in parallel, to overcome the sub-optimal convergence problem. Long convergence times also preclude rapid adaptation to changing conditions, and so PGRL would be best suited to static problems.

6.3.1 An Architecture for Reward Shaping

The Actor-Critic architecture is characterized by the separation of the agent's two functions: acting and learning. The Actor conducts the search through policy space, and the Critic provides critiques, or differences between expected and actual performance [Konda and Tsitsiklis 1999]. Figure 6.2 depicts the classical reinforcement learning model, with tight coupling between an agent's policy and learning algorithm. Figure 6.3 depicts an architecture which de-couples these two functions. This separation allows the Actor and Critic to have separate data structures for representing knowledge,

memory, and policy. Usually, the Actor is designed for efficiency, whereas the Critic is designed to incorporate a longer-term memory, e.g. to determine TD errors, with Temporal Difference methods [Sutton and Barto 1998].

This thesis demonstrated the effectiveness of incorporating expert knowledge (i.e. that cycles are always sub-optimal) in reinforcement learning. This motivates a new architecture, depicted in figure 6.4, where the Critic also assumes the role of Teacher, and can use domain expertise to shape the Actor's behavior. The shaping signal is combined with the underlying performance measure (e.g. by simple addition, or a weighted sum) to deliver a training signal to the Actor, or Student. Such an architecture can catalyze learning by explicitly penalizing behavior to be avoided, and awarding bonuses for behavior to be repeated. Once the Student has learned the taught behavior, it is presumably closer to the optimal solution of the underlying system, and the influence of the shaping signal can be lessened, or removed entirely. Teaching, or partial supervision, could also be used to learn intermediate behavior — e.g. learning to walk, before trying to run.

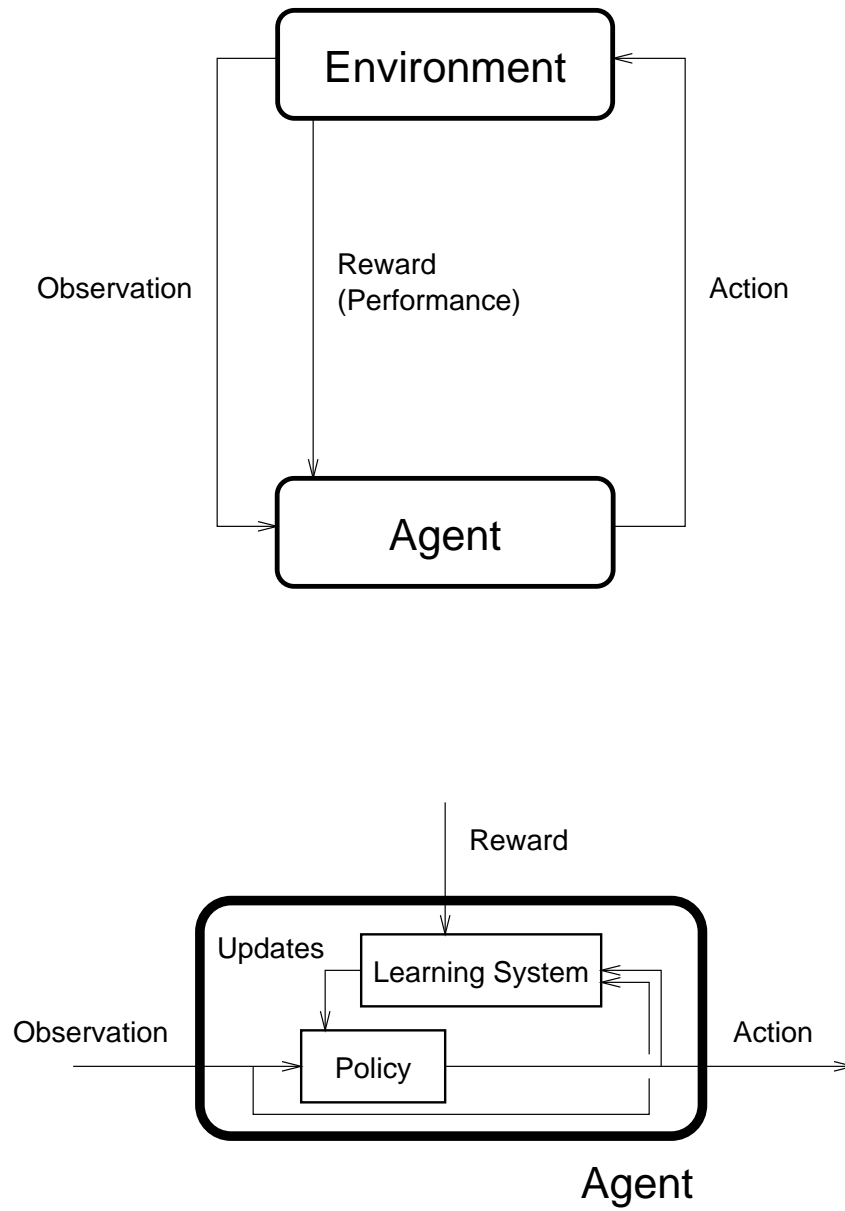


Figure 6.2: The classic reinforcement learning architecture.

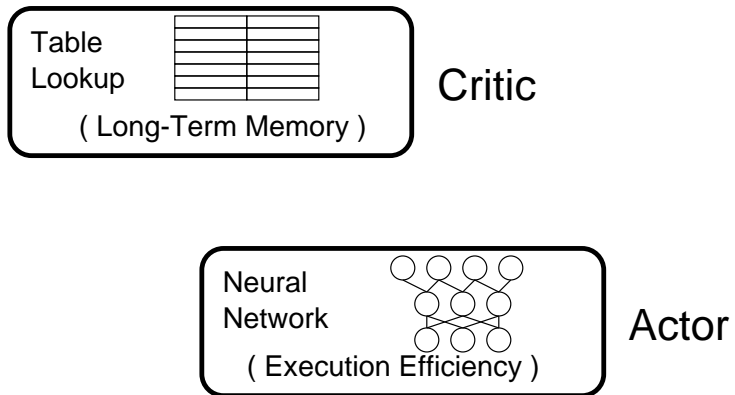
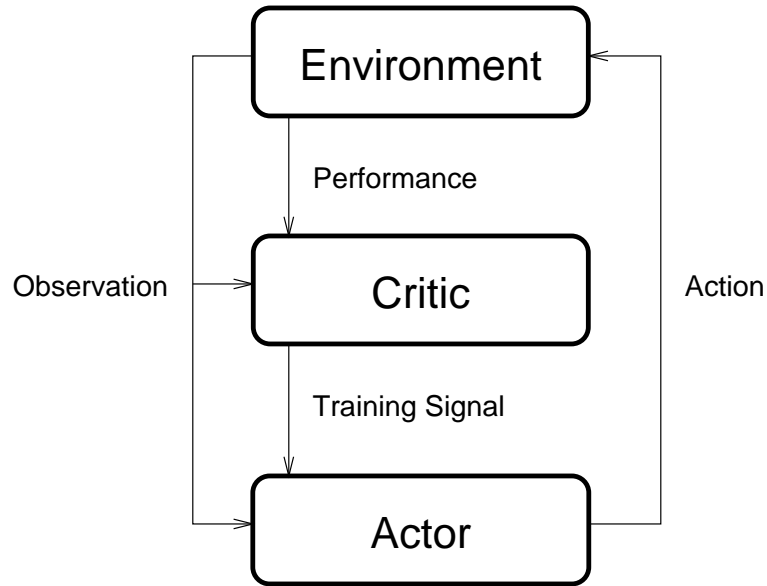


Figure 6.3: The actor-critic architecture.

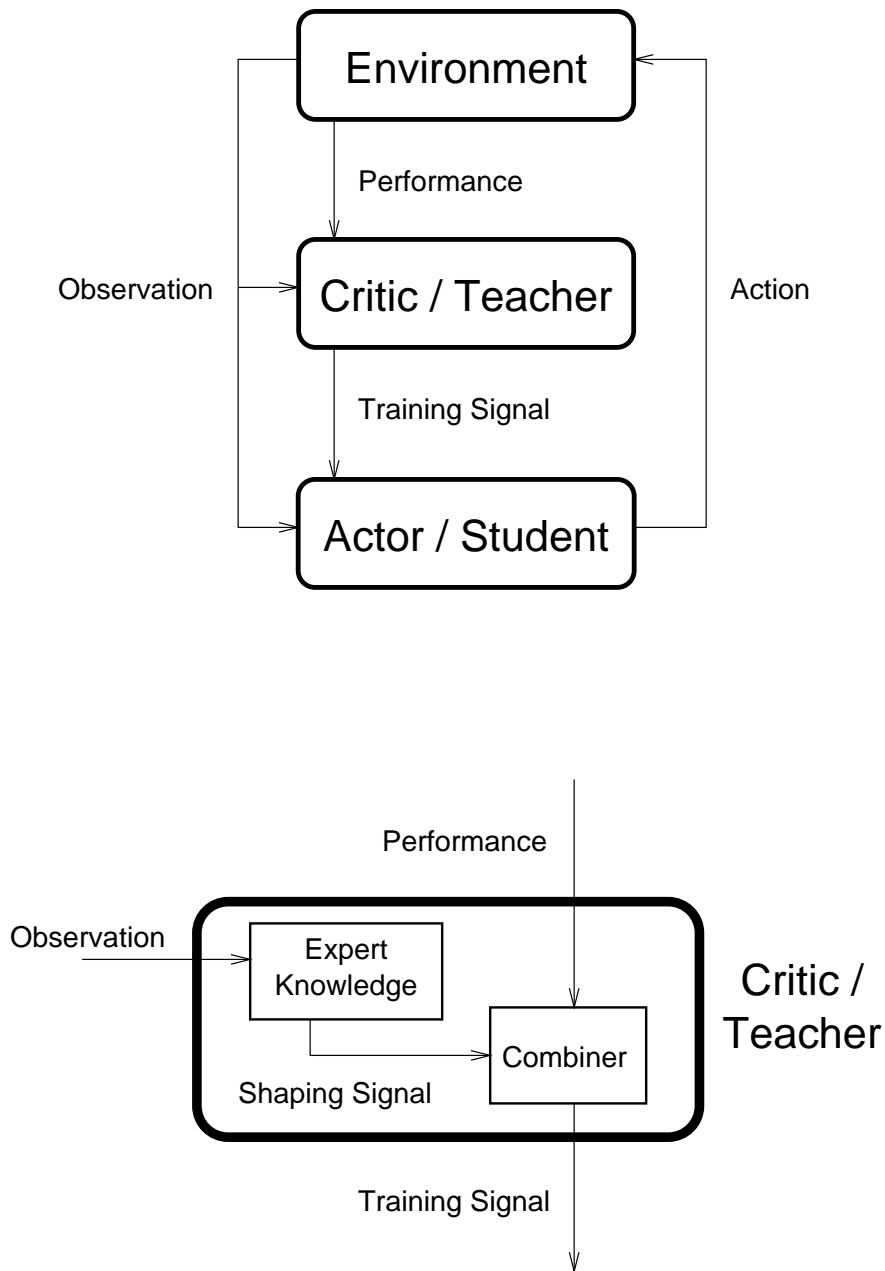


Figure 6.4: An architecture for reward shaping.

Conclusion

It is good to have an end to journey toward; but it is the journey that matters, in the end.

— Ursula K. Le Guin.

This thesis details the design and evaluation of an application of OLPOMDP — a policy-gradient reinforcement learning algorithm — to simulated network routing. In most situations, OLPOMDP has performed comparably to the benchmark Shortest Path algorithm, provided sufficient time was allowed for the algorithm to converge. OLPOMDP was also shown to successfully adapt when the network model changed, which the Shortest Path algorithm did not do. The policy-gradient framework is not specific to the routing domain, and thus is applicable to a wide range of problems. A small sample of alternative application domains is presented in §7.2.1.

7.1 Specific Contributions

1. I have shown that the OLPOMDP algorithm can consistently achieve an optimum solution in some problem domains, without the need for an analytical model.
2. I have shown that policy gradient reinforcement learning methods can be effectively distributed over a number of co-operating agents, without the need for explicit communication other than a (possibly distributed) reward signal.
3. I have shown that the addition of explicit training penalties (based on domain knowledge) to the reward signal can significantly improve the performance of a reinforcement learning algorithm, even without an explicit credit assignment algorithm. An extension of the Actor-Critic architecture was provided to incorporate domain knowledge.

7.2 Future Work

Clearly, this simplistic application of one particular PGRL algorithm is not a practical alternative to current routing methods, where the latter perform well. However, it shows the potential of the policy-gradient approach for solving difficult network routing problems. PGRL is an exciting and new research area, and a small sample of topics for future work is enumerated below.

First, policy-gradient methods allow a diverse range of policies, including a meta-policy that chooses between recommendations of other policies (e.g. Distance-Vector, Link-State). It may be possible to combine PGRL techniques to work with existing algorithms, capturing the strengths of both approaches. Furthermore, this thesis focused on the OLPOMDP algorithm, which is merely one way to perform gradient ascent. A number of alternative algorithms were presented in the same paper by Baxter and Bartlett [1999], and their performance remains untested in network routing, and in other domains.

Second, the performance of OLPOMDP was dependent on the proper choice of the β and γ parameters. These parameters had to be tuned by hand in order for the system to converge to an optimum in a reasonable length of time. There are some guidelines for choosing β , such as the mixing time criterion. However, there is no similar guidelines for γ . The development of a system complexity or noise measure could aid in automatically fixing a γ , and thus broaden the applicability of policy-gradient methods. It may even be possible to develop a self-tuning algorithm, perhaps based on the observed variance in $r\vec{z}$, to adapt its parameters to the system.

Third, a monotonic transformation of the reward signal (i.e. one that preserves order) may improve the convergence rate. Similarly, there remain many research questions on reward shaping in reinforcement learning, and comprehensive design guidelines are still lacking. Further work in both of these areas may help solve the convergence time problem, and will aid the applicability of policy-gradient reinforcement learning techniques.

Fourth, in this thesis, optimality was assessed by explicit analysis. However, in general situations, it may difficult to know whether a system has attained an optimum, or that it has merely slowed down its improvement rate in a noisy environment. The development of an explicit stopping criterion would be useful in further applications.

Fifth, there was no organizational structure, or communication between agents. The reinforcement of co-operative behavior required the co-incidence of those agents to each individually choose those actions. Imposing a hierarchy amongst agents (such as introducing co-ordinating agents who signal worker agents) may improve performance, or even lead to more expressive artificial intelligences. Similarly, explicit communication may hasten learning. For example if B and C were known to be topologically close (e.g. they had the same IP prefix), then the agents $A \leftrightarrow B$ and $A \leftrightarrow C$ might be able to share information. This remains a field of further study.

Finally, there are a variety of other application domains for policy-gradient methods. This is discussed below in more detail.

7.2.1 Other Application Domains

Reinforcement learning methods are applicable anytime there is an appropriate numerical measure of success. For a stock trading agent, success can be measured by profitability. For an information retrieval agent, success can be measured by precision, or average relevance of returned documents. For a data mining agent, success can be measured by classification accuracy. For a humanoid robot agent, success can be measured by walking speed. For a scheduling agent, success can be measured by waiting times. For a systems control agent, success can be measured by resources expended. For a game playing agent, success can be measured by proportion of wins.

Policy-gradient methods are naturally distributable, without any explicit communication between agents. All of the above example applications extend naturally to multiple agents co-ordinating their actions. Such methods could also be applied to competing agents, such as game playing, or stock trading. In competitive domains, deliberate communication is unlikely, yet PGRL agents can still (partially) observe their opponent's actions and respond appropriately. PGRL agents are also more flexible than greedy value-based reinforcement learning agents, since they can follow a mixed strategy, avoiding predictability.

Policy-gradient reinforcement learning is a general technique with wide applicability. We have only just scratched the surface.

Verification

The OLPOMDP routing algorithm is probabilistic, and so there is no “expected correct output” to compare to an implementation’s actual output. Furthermore, convergence (or near-convergence) is guaranteed only “in the long run”, which remains unquantified. Consequently, it is difficult to verify an implementation.

Nonetheless, it is possible to employ statistical techniques to quantify patterns in random processes, and therefore add confidence in the correctness of the implementation.

Our test network is very simple, and is depicted in figure A.1. There are three nodes, denoted A , B and C . Packets are generated at each node at the same rate, and each node has an equal chance of being the destination node of a packet. There are also two links: $B \asymp A$ has a length of 1, and $B \asymp C$ has a length of 5. In this network, we assume that links have unlimited capacity. Note that the routers at A and C always choose the one available link, and so we focus solely on the two routing agents at B . The agent for packets at B destined for A is denoted $B \hookrightarrow A$, and similarly for $B \hookrightarrow C$.

A.1 Theory

Each of these two agents maintains its own set of parameters — one for each link. For the agent $B \hookrightarrow A$, let $\theta_1^{B \hookrightarrow A}$ and $\theta_2^{B \hookrightarrow A}$ denote the parameters for the links $B \asymp A$ and $B \asymp C$ respectively. According to the equations in section 4.2, the probability of

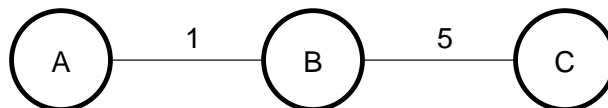


Figure A.1: The verification network.

choosing the link $B \succ A$ is

$$\frac{e^{\theta_1^{B \rightarrow A}}}{e^{\theta_1^{B \rightarrow A}} + e^{\theta_2^{B \rightarrow A}}}$$

and similarly for the link $B \succ C$.

Now suppose there is a packet at B destined for A , which is to be routed. Consider the expected trip time τ_A for that packet. We can rewrite that expectation as a weighted sum on the condition that a certain link are taken:

$$\tau_A = E[\text{trip_time}] = \sum_{l \in \mathcal{L}} E[\text{trip_time} | l] \cdot Pr(l)$$

Now, if the link $B \succ A$ is chosen, the trip time is exactly 1. If the link $B \succ C$ is chosen, it takes 5 time steps to reach C . On arrival, it is routed back on the link $B \succ C$, which takes another 5 time steps. Thus, it takes exactly 10 time steps for that packet to return to B , from which it takes, on average τ_A clock ticks to arrive at A .

Consequently, we have:

$$\tau_A = 1 \cdot \frac{e^{\theta_1^{B \rightarrow A}}}{e^{\theta_1^{B \rightarrow A}} + e^{\theta_2^{B \rightarrow A}}} + (10 + \tau_A) \cdot \frac{e^{\theta_2^{B \rightarrow A}}}{e^{\theta_1^{B \rightarrow A}} + e^{\theta_2^{B \rightarrow A}}}$$

so that

$$\left(1 - \frac{e^{\theta_2^{B \rightarrow A}}}{e^{\theta_1^{B \rightarrow A}} + e^{\theta_2^{B \rightarrow A}}}\right) \cdot \tau_A = \frac{1 \cdot e^{\theta_1^{B \rightarrow A}} + 10 \cdot e^{\theta_2^{B \rightarrow A}}}{e^{\theta_1^{B \rightarrow A}} + e^{\theta_2^{B \rightarrow A}}}$$

and

$$e^{\theta_1^{B \rightarrow A}} \cdot \tau_A = 1 \cdot e^{\theta_1^{B \rightarrow A}} + 10 \cdot e^{\theta_2^{B \rightarrow A}}$$

i.e.

$$\tau_A = 1 + 10 \cdot e^{\theta_2^{B \rightarrow A} - \theta_1^{B \rightarrow A}}$$

The agent has no impact on any routing decisions other than for those packets at B destined for A , and so the agent's impact on the global reward signal is merely proportional to τ_A , with the proportion being related to the amount of traffic which is at B and destined for A . In the long term, this is directly related to the amount of traffic generated for A , which is equal to one third of the total traffic generation rate.

Thus, $\nabla \eta$, the gradient (with respect to $\theta_{1,2}^{B \rightarrow A}$) of expected long-term average reward, is proportional to $\nabla \tau_A$. Now:

$$\frac{\partial \tau_A}{\partial \theta_1^{B \rightarrow A}} = -10 \cdot e^{\theta_2^{B \rightarrow A} - \theta_1^{B \rightarrow A}}$$

and

$$\frac{\partial \tau_A}{\partial \theta_2^{B \rightarrow A}} = +10 \cdot e^{\theta_2^{B \rightarrow A} - \theta_1^{B \rightarrow A}}$$

$\log_{10} T$	samples	mean	std. dev.
2	53995	-0.958	49.02
3	4394	2.772	34.86
4	2822	3.622	11.89
5	954	3.571	3.75
6	612	3.632	1.22

Table A.1: First component of gradient estimate for $B \leftrightarrow A$, with β held constant at 0.99.

Now consider the agent $B \leftrightarrow C$, let τ_C be the expected trip time for a packet at B to arrive at C . Then similarly,

$$\frac{\partial \tau_C}{\partial \theta_1^{B \leftrightarrow C}} = +2 \cdot e^{\theta_1^{B \leftrightarrow C} - \theta_2^{B \leftrightarrow C}}$$

and

$$\frac{\partial \tau_C}{\partial \theta_2^{B \leftrightarrow C}} = -2 \cdot e^{\theta_1^{B \leftrightarrow C} - \theta_2^{B \leftrightarrow C}}$$

If we initialize $\theta_1 = \theta_2 = 0$ for both agents, then the ratio of two gradients should be the negative of the ratio of the link costs.

A.2 Practice

Estimates of $\nabla_{\beta} \eta[1]$ for the agent at $B \leftrightarrow A$ were made over a number of independent runs. Baxter and Bartlett [1999] give two related algorithms: OLPOMDP for on-line gradient ascent, and GPOMDP which calculates a gradient estimate, parameterized by T , the number of time steps to gather data for. In their paper, they prove that $\lim_{T \rightarrow \infty} \text{GPOMDP} = \nabla_{\beta} \eta$.

Our implementation was tested against the theory by making a number of independent runs of GPOMDP in order to generate gradient estimates. The first parameter examined was T . Table A.1 suggests that, for sufficiently large T , the estimated mean is relatively stable. However, the variation in estimates is clearly dependent on T . Larger T leads to longer running times, and a choice of $T = 10^4$ or 10^5 appears a reasonable trade-off between speed and accuracy.

The next parameter of our algorithm that we examine is β , the notion of ‘memory’, or how far back in time previous actions are associated with the most recent reward signal. Table A.2 clearly shows the bias-variance trade-off in choosing β .

In gradient ascent algorithms, it is more important to adjust parameters in the right direction, rather than by the right magnitude. Moving in the right direction (i.e. uphill) means that each step (on average) improves the agent’s performance. If the step

β	$\frac{1}{1-\beta}$	samples	mean	std. dev.	ratio
0.9	10	951	-3.631	0.35	-10.374
0.96	25	979	-2.273	0.95	-2.397
0.99	100	954	3.571	3.75	0.952
0.996	250	970	5.994	9.52	0.630
0.999	1000	944	7.963	38.27	0.208
0.9996	2500	1317	8.716	92.38	0.094
0.9999	10000	1404	16.082	340.08	0.048

Table A.2: First component of gradient estimate for $B \leftrightarrow A$, with T held constant at 10^5 .

agent	samples	mean	std. dev.	ratio
$B \leftrightarrow A$	944	7.963	38.27	0.208
$B \leftrightarrow C$	1157	-1.546	37.86	-0.041

Table A.3: First component of gradient estimate per agent, $\beta = 0.999$, $T = 10^5$.

size is incorrect, it merely takes longer to converge to an optimum. Moving by the right amount means that each step is by a good length, but the agent might be stepping down-hill, and its performance will be getting worse.

Consequently, our measure of goodness for β is in fact the ratio of mean to standard deviation. The higher this is, the greater proportion of estimates will be positive (i.e. in the right direction). On this measure, the best choice in practice is $\beta = 0.99$. Note that at this value, the gradient estimate appears to be biased downwards, but this is the trade-off in order to achieve low variance.

However, for verification purposes, an unbiased estimate is preferable to a low variance estimate, in order to make a numerical comparison. For our statistical test, we choose $T = 10^5$ and $\beta = 0.999$. Our results are given in table A.3.

It appears reasonable to infer that the ratio of means is actually -5:1. Formally, we can make a statistical test of the null hypothesis: $H_0 : 1 \cdot \nabla_{\beta}^{B \leftrightarrow A} \eta[1] + 5 \cdot \nabla_{\beta}^{B \leftrightarrow C} \eta[1] = 0$ against the alternative hypothesis: $H_a : 1 \cdot \nabla_{\beta}^{B \leftrightarrow A} \eta[1] + 5 \cdot \nabla_{\beta}^{B \leftrightarrow C} \eta[1] \neq 0$

For simplicity, we can assume that the sample sizes are large and are both equal to 1000. The appropriate Z-value is therefore:

$$Z = \frac{1 \cdot \nabla_{\beta}^{B \leftrightarrow A} \eta[1] + 5 \cdot \nabla_{\beta}^{B \leftrightarrow C} \eta[1]}{\sqrt{\frac{1}{2} \cdot ((\sigma^{B \leftrightarrow A})^2 + (\sigma^{B \leftrightarrow C})^2) \left(\frac{1^2}{1000} + \frac{5^2}{1000} \right)}}$$

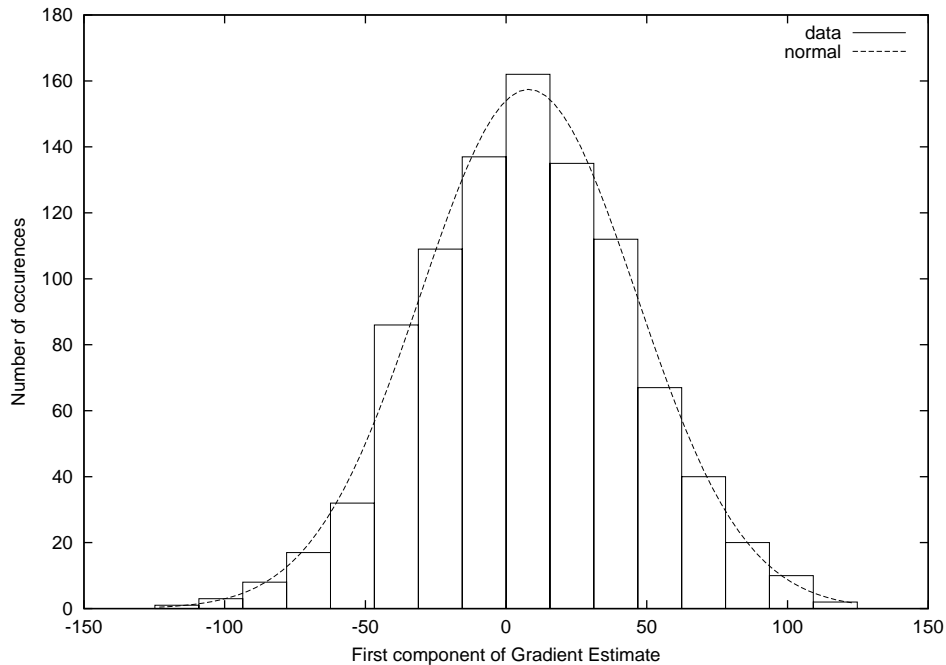


Figure A.2: Gradient estimate component compared to the normal distribution.

and so our data gives a z -score of:

$$\begin{aligned}
 z &= \frac{1.(7.963) + 5.(-1.546)}{\sqrt{1448 \cdot \frac{26}{1000}}} \\
 &= -0.038
 \end{aligned}$$

which, when compared to the normal distribution $N(0, 1)$, is not significant ($p = 0.96$). Thus, we retain the null hypothesis H_0 in favor of the alternative H_a .

This test relies on the assumption that the samples are normally distributed. A quick glance at a histogram (with an overlaid normal distribution) in figure A.2 suggests that this is a very reasonable assumption. This plot is for the agent $B \hookrightarrow A$ with $\beta = 0.999$ and $T = 10^5$. Plots for different parameter values are similar.

The Moving Average Smoother

Packets only register a reward in the time step they arrive at their destination. Although packets are generated at a regular rate, their arrival rate is irregular, being dependent on the routing choices made. Figure B.1 shows a typical reward stream, which is quite volatile. A moving average (of order N) of a time series r_t is the derived series

$$MA_t = \frac{1}{N} \cdot \sum_{i=0}^{N-1} r_{t-i}$$

Tracking the moving average removes the noise and clarifies the trend.

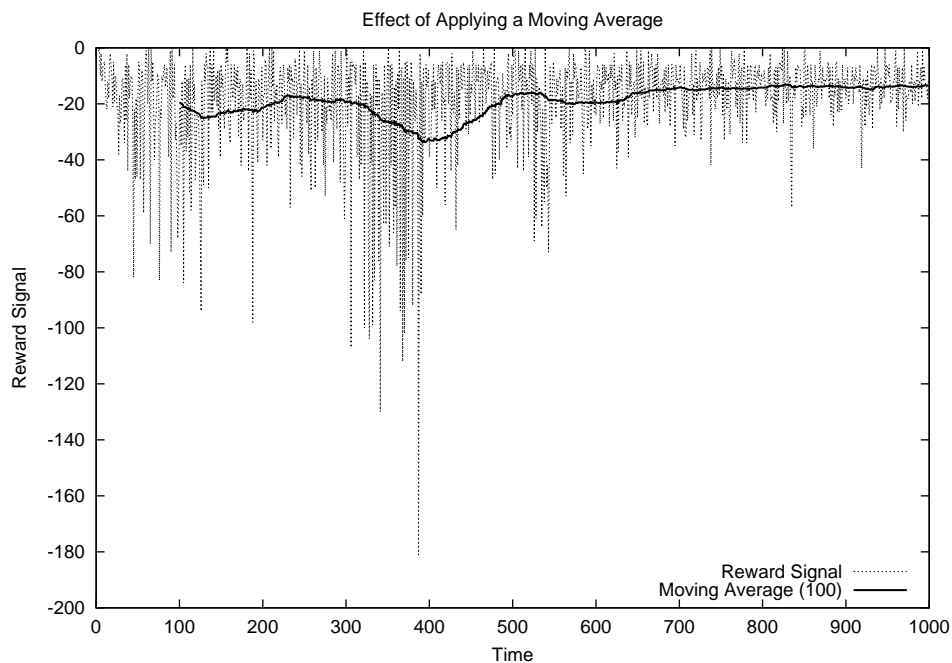


Figure B.1: Noise reduction by applying a moving average.

Existence of Multiple Optima

We present a simple model of network routing in which there exist local optima which are not global optima.

We model the network as a finite, connected weighted graph. The nodes of the graph represent routers, and the edges of the graphs represent links (with weights being the propagation time on that link). Denote the link joining X and Y by $X \asymp Y$. Bandwidth is limited, in that each link has an associated capacity. A link can only accept a number of packets per time step equal to its capacity. Extra packets are merely dropped (and a penalty term, *penalty*, is added to the reward signal).

The metric to be optimized is the expected reward signal, which consists of the negative of the long-term average of the total packet trip time, plus the long-term average penalties.

Consider the network depicted in figure C.1, with each link having capacity 1. Suppose that nodes B and C route perfectly, i.e. for a packet destined to X , they will place that packet on the direct link to X .

For the agent $A \leftrightarrow B$, denote by $\mu_1^{A \leftrightarrow B}$ and $\mu_2^{A \leftrightarrow B}$ the probabilities of choosing the link $A \asymp B$ and $A \asymp C$ respectively. The corresponding parameters for $A \leftrightarrow C$ are denoted similarly. Clearly, the optimum policy is to have $\mu_1^{A \leftrightarrow B}$ and $\mu_2^{A \leftrightarrow C}$ as close to

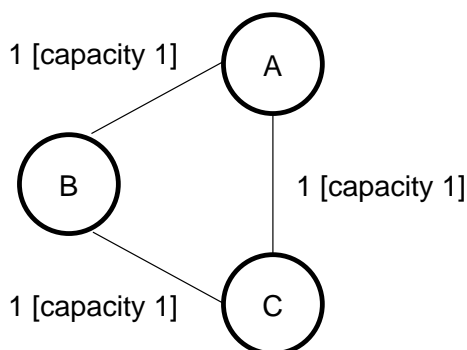


Figure C.1: Contention Network.

1 as possible (and consequently $\mu_2^{A \leftrightarrow B}$ and $\mu_1^{A \leftrightarrow C}$ as close to 0 as possible).

However, suppose that we are in a situation where $\mu_1^{A \leftrightarrow B} = 0.1$ and $\mu_2^{A \leftrightarrow B} = 0.9$. Furthermore, suppose that traffic is generated at A only, with exactly one packet destined for B and one packet destined for C generated at every time step. Since nodes B and C route perfectly, there is no incoming traffic to A that has to be re-routed, and so at every time step, the only traffic to be routed is those two packets generated at A .

Now, the probability of collision on the $A \asymp B$ link is $\mu_1^{A \leftrightarrow B} \cdot \mu_1^{A \leftrightarrow C}$. In that case, one packet is dropped, generating a reward of *penalty*. The other packet gets through, with a trip time of either 1 or 2, depending on whether it was destined to B or C .

Similarly, the probability of collision on the $A \asymp C$ link is $\mu_2^{A \leftrightarrow B} \cdot \mu_2^{A \leftrightarrow C}$, which would generate a reward value of *penalty* – 1.5, on average.

The probability of both packets being routed correctly is $\mu_1^{A \leftrightarrow B} \cdot \mu_2^{A \leftrightarrow C}$, for which the expected packet trip time is 1 + 1, and so the (anticipated) reward is –2. Similarly, the probability of both packets being routed incorrectly is $\mu_2^{A \leftrightarrow B} \cdot \mu_1^{A \leftrightarrow C}$, with an anticipated reward of –4.

So, the expected long-term average reward η , dependent on the policy parameters $\vec{\mu}^{A \leftrightarrow C}$ and $\vec{\mu}^{A \leftrightarrow B} = [0.1, 0.9]$, is

$$\eta(\vec{\mu}^{A \leftrightarrow C}) = \begin{aligned} & \mu_1^{A \leftrightarrow C} \cdot (0.9) \cdot (-4) + \\ & \mu_1^{A \leftrightarrow C} \cdot (0.1) \cdot (-1.5 + \textit{penalty}) + \\ & \mu_2^{A \leftrightarrow C} \cdot (0.9) \cdot (-1.5 + \textit{penalty}) + \\ & \mu_2^{A \leftrightarrow C} \cdot (0.1) \cdot (-2) \end{aligned}$$

Substituting $\mu_2^{A \leftrightarrow C} = 1 - \mu_1^{A \leftrightarrow C}$ gives:

$$\begin{aligned} \eta(\vec{\mu}^{A \leftrightarrow C}) &= \mu_1^{A \leftrightarrow C} \cdot (-3.6) + \\ & \mu_1^{A \leftrightarrow C} \cdot (-0.15 + 0.1 \cdot \textit{penalty}) + \\ & (1 - \mu_1^{A \leftrightarrow C}) \cdot (-1.35 + 0.9 \cdot \textit{penalty}) + \\ & (1 - \mu_1^{A \leftrightarrow C}) \cdot (-0.2) \\ &= \mu_1^{A \leftrightarrow C} \cdot (-2.2 - 0.8 \cdot \textit{penalty}) + (-1.55 + 0.9 \cdot \textit{penalty}) \end{aligned}$$

so that, provided *penalty* is sufficiently negative, the co-efficient of $\mu_1^{A \leftrightarrow C}$ is positive, and so the expected long term average reward is maximized with $\mu_1^{A \leftrightarrow C}$ as high as possible. Furthermore, the gradient direction is always towards high $\mu_1^{A \leftrightarrow C}$ (and thus low $\mu_2^{A \leftrightarrow C}$).

However, this means that the (unilaterally) best policy for $A \leftrightarrow C$ is to route them on the $A \asymp B$ link to avoid collisions, rather than fight for the shortest route.

Similarly, if $\mu_1^{A \leftrightarrow C}$ is high (and $\mu_2^{A \leftrightarrow C}$ is low), and the penalty term is sufficiently harsh, then the best policy for the agent $A \leftrightarrow B$ is to avoid collisions and prefer link 2. Thus, a situation can occur where the both agents avoid the best collective policy by pursuing their best individual policy.

Bibliography

- BAXTER, J. AND BARTLETT, P. L. 1999. Direct gradient-based reinforcement learning: I. gradient estimation algorithms. Technical report (July), Research School of Information Science and Engineering, Australian National University. (pp. 1, 20, 26, 33, 72, 77)
- BAXTER, J., TRIDGELL, A., AND WEAVER, L. 1998. KnightCap: A chess program that learns by combining TD(λ) with game-tree search. In *Proceedings of the Fifteenth International Conference on Machine Learning* (1998), pp. 28–36. (p. 15)
- BAXTER, J., WEAVER, L., AND BARTLETT, P. L. 1999. Direct gradient-based reinforcement learning: II. gradient ascent algorithms and experiments. Technical report (September), Research School of Information Science and Engineering, Australian National University. (p. 20)
- BELLMAN, R. E. 1957. *Dynamic Programming*. Princeton University Press, Princeton. (pp. 2, 15, 17)
- BELLMAN, R. E. 1958. On a routing problem. *Quarterly of Applied Mathematics* 16, 87–90. (pp. 10, 17)
- BOYAN, J. A. AND LITTMAN, M. L. 1993. Packet routing in dynamically changing networks: A reinforcement learning approach. *Advances in Neural Information Processing Systems* 6, 671–678. (p. 17)
- BRAESS, D. 1968. Uber ein Paradoxon aus der Verkehrsplanung [A paradox of traffic assignment problems]. *Unternehmensforschung* 12, 258–268. (p. 10)
- CARO, G. D. AND DORIGO, M. 1998. Ant colonies for adaptive routing in packet-switched communications networks. In *Proceedings of the Fifth International Conference on Parallel Problem Solving from Nature* (Amsterdam, Holland, 1998). Springer-Verlag. (p. 12)
- COASE, R. 1960. The problem of social costs. *The Journal of Law and Economics* 3, 1–44. (p. 14)
- COHEN, J. E. AND KELLY, F. P. 1990. A paradox of congestion in a queueing network. *Journal of Applied Probability* 27, 730–734. (pp. 10, 56)
- CORMEN, T. H., LEISERSON, C. E., AND RIVEST, R. L. 1990. *Introduction to Algorithms*. MIT Press, Cambridge, MA. (p. 10)

- DEJONG, G. AND SPONG, M. W. 1994. Swinging up the acrobot: an example of intelligent control. In *Proceedings of the American Control Conference* (1994), pp. 2158–2162. American Automatic Control Council, Evanston IL. (p. 20)
- DEO, N. AND PANG, C. 1984. Shortest path algorithms: Taxonomy and annotation. *Networks* 14, 275–323. (pp. 1, 10)
- DIJKSTRA, E. 1956. A note on two problems in connection with graphs. *Canadian Journal of Mathematics* 8, 419–433. (p. 10)
- DIXIT, A. AND SKEATH, S. 1999. *Games of Strategy*. W. W. Norton. (p. 12)
- FORD, L. R. AND FULKERSON, D. R. 1962. *Flows in Networks*. Princeton University Press, Princeton, NJ. (p. 10)
- FRIEDMAN, J. W. Ed. 1994. *Problems of co-ordination in economic activity*. Kluwer Academic Publishers, Boston. (p. 23)
- GAMMA, E., HELM, R., JOHNSON, R., AND VLISSIDES, J. 1995. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison Wesley Longman Inc. (p. 25)
- GIBNEY, M. A., JENNINGS, N. R., VRIEND, N. J., AND GRIFFITHS, J. M. 1999. Market-based call routing in telecommunications networks using adaptive pricing and real bidding. In *Proceedings of the Third International Workshop for Intelligent Agents for Telecommunications Applications*, Number 1699 in Lecture Notes in Artificial Intelligence (Stockholm, Sweden, 1999). (p. 12)
- HARDIN, G. 1968. The tragedy of the commons. *Science* 162, 1243–1248. (p. 14)
- HOLLAND, J. H. 1975. *Adaptation in Natural Artificial Systems*. University of Michigan Press, Ann Arbor. (p. 18)
- HOLLAND, J. H. 1985. Properties of the bucket brigade algorithm. In *Proceedings of the First International Conference on Genetic Algorithms and their Applications* (1985), pp. 1–7. (p. 14)
- KIRKPATRICK, S., GELATT, C. D., AND VECCHI, M. P. 1983. Optimization by simulated annealing. *Science* 220, 671–680. (p. 19)
- KONDA, V. AND TSITSIKLIS, J. 1999. Actor-critic algorithms. In *Proceedings of the 1999 Neural Information Processing Systems Conference* (1999). (p. 66)
- KUSHNER, H. J. AND YIN, G. 1997. *Stochastic Approximation Algorithms and Applications*. Springer, New York. (p. 28)
- LABOVITZ, C., MALAN, G. R., AND JAHANIAN, F. 1997. Internet routing instability. In *Proceedings of the ACM SIGCOMM '97 Conference* (Cannes, France, 1997). (p. 21)

-
- MARBACH, P. AND TSITSIKLIS, J. N. 1998. Simulation-based optimization of Markov reward processes. Technical report, MIT. (p. 20)
- MATARIC, M. J. 1994. Reward functions for accelerated learning. In *Proceedings of the Eleventh International Conference on Machine Learning*. (1994). (p. 17)
- MITCHELL, T. M. 1997. *Machine Learning*. McGraw-Hill. (pp. 2, 17, 19)
- NASH, J. F. 1950. Equilibrium points in n-person games. In *Proceedings of the National Academy of Sciences of the United States of America*, Volume 36 (1950), pp. 48–49. (p. 12)
- NG, A., HARADA, D., AND RUSSELL, S. 1999. Policy invariance under reward transformations: Theory and application to reward shaping. In *Proceedings of the Sixteenth International Conference on Machine Learning*. (1999). (p. 17)
- ORDA, A., ROM, R., AND SIDI, M. 1993. Minimum delay routing in multisatellite networks. *IEEE/ACM Transactions on Networking* 1, 2, 187–198. (p. 10)
- RANDLØV, J. AND ALSTRØM, P. 1998. Learning to drive a bicycle using reinforcement learning and shaping. In *Proceedings of the Fifteenth International Conference on Machine Learning*. (1998). (p. 17)
- SAMUEL, A. L. 1959. Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development* 3, 210–229. (p. 15)
- SCHOONDERWORED, R., HOLLAND, O., BRUTEN, J., AND ROTHKRANTZ, L. 1996. Ant-based load balancing in telecommunications networks. *Adaptive Behavior* 5, 2, 169–207. (p. 12)
- STERNBERG, R. J. 1998. *In Search of the Human Mind* (2nd ed.). Harcourt Brace. (p. 15)
- SUTTON, R. S. AND BARTO, A. G. 1998. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA. (pp. 2, 14, 15, 17, 28, 67)
- TANENBAUM, A. S. 1996. *Computer Networks* (3rd ed.). Prentice Hall International. (pp. 9, 10, 11, 17, 22)
- TESAURO, G. 1994. TD-Gammon, a self-teaching backgammon program, achieves master-level play. *Neural Computation* 6, 215–219. (p. 15)
- TUMER, K. AND WOLPERT, D. H. 1999. Avoiding Braess' paradox through collective intelligence. Technical report (December), NASA Ames Research Center. (pp. 11, 56)
- VARIAN, H. 1999. *Intermediate Microeconomics: A Modern Approach* (5th ed.). W. W. Norton. (p. 11)
- VON NEUMANN, J. AND MORGENSTERN, O. 1944. *Theory of Games and Economic Behavior*. Princeton University Press. (p. 12)

- WATKINS, C. 1992. Q-Learning. *Machine Learning* 8, 3, 229–256. (p. 15)
- WEAVER, L. AND BAXTER, J. 1999. Reinforcement learning from state and temporal differences. Technical report (May), Department of Computer Science, Australian National University. (p. 20)
- WILLIAMS, R. J. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning* 8, 229–256. (p. 20)
- YANG, C. Q. AND REDDY, A. V. S. 1995. A taxonomy for congestion control algorithms in packet switching networks. *IEEE Network Magazine* 9, 5. (p. 1)

Glossary

action: a conscious choice by an *agent* which can affect its *environment*.

agent: an intelligent *process* (*I*) that interacts with its *environment*.

bandwidth: the amount of traffic flow a link can accept.

bonus: a positive addition to the *reward signal* that is not directly related to performance, but is instead to encourage certain behaviors. See *penalty*.

broadcast: the delivery of a message from one router to all of its peers.

congestion: the overcrowding of a network resource (e.g. link, router buffer). This is an example of *contention*.

contention: a situation where multiple *agents* are competing for a limited resource.

converge: to achieve *equilibrium*.

credit assignment: the division and allocation of a *reward signal* to those *agents* (and their *actions*) which actually contributed to achieving the goal that triggered it.

deterministic: absolutely predictable, given sufficient data. See *non-deterministic*.

deterministic policy: a *policy* whose action is fully determined by the *observation*. Formally, it is a function from *observations* to *actions*. See *probabilistic policy*.

distributed: where resources are *local* (3).

eligibility trace: a record of *actions* likely to have contributed to the current *reward signal*. Its purpose is to deliver correct *credit assignment*.

environment: the system an *agent* senses, or *observes*. An *agent* can modify the state of its *environment* by its *actions*.

equilibrium: where a system is stable, on average, over time.

feedback: information on an *agent's* recent *actions* which allow it to assess the appropriateness of those *actions*.

flooding: a *routing* algorithm where incoming packets from one link are duplicated and forwarded on all other links, unless that packet has already been seen before. This is an effective way to *broadcast*.

global (1): pertaining to all *agents*, not just a single *agent*. See *local (1)*.

global (2): ranging over the whole of a *parameter space*, not just a neighborhood. See *local (2)*.

global optimum: a state achieving the best possible performance. See *global (2)*, *local optimum*.

gradient: the *gradient* of a surface at a point is the steepest uphill (positive) direction of that surface, from that point.

gradient ascent: adjustment of a *agent's parameters* in the *gradient* direction. If the adjustments are sufficiently small, this will *converge* to a *local optimum*.

local (1): pertaining to a single *agent*, not to all *agents*. See *global (1)*.

local (2): ranging only over a subset of a *parameter space* close to a certain point (e.g. a point that is a *local optimum*). See *global (2)*.

local (3): place-specific, in a *distributed* system. For example, a *routing agent* can be *local* to a node *A*, and *A's* memory is similarly *local*, but memory on other computers is not *local* to that *agent*.

local optimum: a state where a small change will always lead to a degradation in performance. Unless the *local optimum* is also a *global optimum*, a large change may actually improve performance. See *local (2)*.

machine learning: the study of algorithms which improve with experience. It is traditionally divided into the two separate fields of *supervised learning* and *unsupervised learning*.

Markov process: a *stochastic process* in which the probability of the system being in a certain state at time $(t + 1)$ is dependent only on the state at time t . The Markov property is equivalent to assuming that there is a finite length of time T such that any event more than T time units ago is no longer relevant in predicting future states.

Markov decision process: a *Markov process* in which the *transition probabilities* are affected by an *agent's* decisions (or *actions*).

model: a set of simplifying assumptions about a system which enable the application of analytical techniques.

multilateral: behavior change by one *agent* matched by corresponding changes in other *agents*. See *unilateral*.

Nash equilibrium: a situation involving multiple *agents* where no *agent* can *unilaterally* improve its welfare.

noisy: containing an unpredictable, or *stochastic* component.

non-deterministic: dependent on chance, to an extent. See *deterministic*.

non-deterministic policy: see *probabilistic policy*.

observation: the information an *agent* extracts from its *environment*. This can be *noisy*.

off-line learning: where *policy* updates are accumulated over a significant length of time, before being applied. See *on-line learning*.

OLPOMDP: a particular *policy-gradient* algorithm for solving *reinforcement learning* problems. It is an acronym for *On-line Learning* algorithm for *Partially Observable Markov Decision Processes*.

on-line learning: where *policy* updates are applied as soon as they are calculated. See *off-line learning*.

parameter: an adjustable factor for an algorithm, or *policy*. For example, with a neural network, the each linkage weight is a *parameter*.

parameter space: the range of all possible *parameters*.

partially observable: where *observation* is *noisy*, so the exact state of the *environment* is not known for certain. This may be for physical reasons (i.e. machine vision has limited resolution) or for efficiency reasons (i.e. compressing system state into a feature vector).

penalty: a negative addition to the *reward signal* that is not directly related to performance, but is instead to discourage certain behaviors. See *bonus*.

policy: how an *agent* reacts to its *observation* of the environment. This may be a *deterministic policy*, or a *probabilistic policy*.

policy-gradient methods: algorithms which aim to improve a *policy* by *gradient ascent* in the *policy's parameter space*.

probabilistic policy: a *policy* which may respond to an *observation* by one of a number of *actions*. Such *actions* have a probability of occurring, given the *observation*. Formally, a *probabilistic policy* is a function from *observations* to *probability distributions* over *actions*. Also known as a *non-deterministic policy*. See *deterministic policy*.

probability distribution: the chances of each of the set of possible outcomes occurring. For example, if two coins are flipped, the *probability distribution* is: 25% for two heads, 25% for two tails, and 50% for one of each.

process (1): the execution of a computer program.

process (2): statistically, a sequence of numbers, such as daily rainfall.

reinforcement learning: the study of *machine learning* algorithms which rely solely on a *reward signal* for *feedback*. Also known as *unsupervised learning*.

reward signal: a numerical performance measure. A higher *reward signal* indicates better performance.

routing: the problem of choosing the best path, or route, from one place to another.

stochastic: informally, influenced by chance or probability.

stochastic process: a sequence $\{X_0, X_1, X_2, \dots\}$ for which the *probability distribution* of X_t depends on previous history $\{X_{t-1}, X_{t-2}, \dots\}$. If it is only dependent on the most recent state X_{t-1} , then the *process (2)* is a *Markov process*.

supervised learning: the study of *machine learning* algorithms which rely on the presentation of training data (and the desired behavior to mimic) by a human with expert knowledge.

time-to-live: the maximum time a packet can survive before it is presumed to be in a routing loop, and dropped.

transition probability: for a *Markov process*, the probability of some immediate future state, given the current state (and the current *action*, for a *Markov decision process*).

unilateral: behavior change by one *agent* without corresponding changes in other *agents*. See *multilateral*.

unsupervised learning: see *reinforcement learning*.

utility: a numerical measure of welfare. Individual *utility* refers to a single *agent*'s welfare, *global (1) utility* refers to the collective welfare of all *agents*.

utility function: a formula for calculating *utility*.

Notation

Mathematics

\mathbf{R}	the set of real numbers
Σ	the summation operator
\mathbf{E}	the expectation operator
∇	the gradient operator
∂	the partial derivative operator
σ	the population standard deviation
$\lim_{T \rightarrow \infty}$	the limit as T approaches infinity
$x \sim d$	x follows the probability distribution d
$\vec{v}[i]$	the i^{th} component of vector \vec{v} , sometimes written as v_i for clarity

Reinforcement Learning

u	an action
μ_{u_i}	the probability of taking action u_i
$\vec{\theta}$	the policy parameters
$\eta(\vec{\theta})$	the expected long term average reward, as a function of the policy parameters
$\nabla \eta$	the gradient (with respect to $\vec{\theta}$) of η
\vec{z}	the accumulated eligibility trace
β	the eligibility trace discount rate
$\nabla_{\beta} \eta$	the estimate to $\nabla \eta$
r	the reward value
γ	the step size

Networks

$A \hookrightarrow B$	the routing agent at A for packets destined to B
$A \asymp B$	the link connecting nodes A and B
$A \asymp B \asymp C$	the path $A \asymp B$ followed by $B \asymp C$
\mathcal{L}	the set of links at a node
$\tau_{A,B}$	the expected trip time for a packet at A to get to B
τ_A	the expected trip time for a packet at A to get to its destination
TTL	the time-to-live for packets