UNIVERSITY OF CAMBRIDGE

An algebraic perspective on the convergence of vector-based routing protocols

Matthew L. Daggitt

Jesus College

This thesis is submitted on December 18th, 2018 for the degree of Doctor of Philosophy

An algebraic perspective on the convergence of vector-based routing protocols - by Matthew L. Daggitt

This thesis studies the properties of vector-based routing protocols whose underlying algebras are strictly increasing. Strict increasingness has previously been shown to be both a sufficient and a necessary condition for the convergence of path-vector protocols.

One of the key contributions of this thesis is to link vector-based routing to a much larger family of asynchronous iterative algorithms. This unlocks a significant body of existing theory, and allows asynchronous protocols to be proved correct by purely synchronous reasoning. As well as applying it to routing protocols, this thesis advances the asynchronous theory in two ways. Firstly it shows that the existing conditions required for convergence may be relaxed. Secondly it proposes the first model for "dynamic" asynchronous processes in which both the problem being solved and the set of participants change over time.

The thesis' attention then turns to models of routing problems, and presents a new algebraic structure that is simpler and more expressive than the state of the art. In particular this structure is capable of modelling routing problems that underlie both distance-vector and path-vector protocols. Consequently these two families of vector-based protocols may be unified for the first time. The new structure is also capable of modelling protocols that use path-dependent conditional policy.

Next the work above is used to construct a model of an abstract vector-based protocol. This is then used in the first proof of correctness for strictly increasing distance-vector protocols and a new proof of correctness for strictly increasing path-vector protocols. The latter is an improvement over previous results as it i) proves that convergence is deterministic ii) does not assume reliable communication between nodes and iii) applies to path-vector protocols with path-dependent conditional policy. The long standing question of the worst-case rate of convergence for a strictly increasing path-vector protocol is then answered by lowering the previous upper bound of $O(n!)$ to a new tight bound of $\Theta(n^2)$.

Finally all of the work has been formalised in the proof assistant Agda. Not only does this significantly increase users' confidence in the validity of the results, the resulting Agda library may also be used to verify the correctness of protocol implementations. To illustrate this, a formal proof of correctness is described for a path-vector protocol which contains many of the features of the Border Gateway Protocol including: local preferences, communities, an expressive conditional policy language and path inflation.

Declaration

This thesis is the result of my own work and includes nothing which is the outcome of work done in collaboration except as declared within. Several of the results have been published previously:

- *An Agda Formalization of Üresin & Dubois' Asynchronous Fixed-Point Theory*, Ran Zmigrod, Matthew L. Daggitt and Timothy G. Griffin, ITP 2018.
- *Asynchronous Convergence of Policy-Rich Distributed Bellman-Ford Routing Protocols*, Matthew L. Daggitt, Alexander J.T. Gurney and Timothy G. Griffin, SIGCOMM 2018.
- *Rate of convergence of increasing path-vector routing protocols*, Matthew L. Daggitt and Timothy G. Griffin, ICNP 2018.

This thesis is not substantially the same as any that I have submitted, or am concurrently submitting, for a degree or diploma or other qualification at the University of Cambridge or any other University or similar institution. I further state that no substantial part of my dissertation has already been submitted, or is being concurrently submitted, for any such degree, diploma or other qualification at the University of Cambridge or any other University or similar institution. This thesis does not exceed the prescribed limit of 60 000 words.

Matthew L. Daggitt
18th of December, 2018

Acknowledgements

Foremost, I would like to express my deep gratitude to my supervisor Dr Timothy Griffin for his support, enthusiasm and perseverance, both through the quiet times and the slightly more stressful times. It is difficult to imagine having had a better supervisor, and there's not enough coconut water in the world to demonstrate my appreciation.

Thank you to my partner Ruby for persuading me to do a PhD in the first place. As always you were absolutely right. I would also like to thank my friends in the lab Hugo, Philip, Ian, Dylan, Simon and Aurore for making my time in the Computer Lab so enjoyable. I am also grateful to my house mates Izabela, Kim and Becca, as well as Jesus Badminton Club, for ensuring I had a life outside of the lab as well.

Finally I would like to thank EPSRC for funding the research and to Jesus College for providing additional travel grants.

Contents

Chapter 1

Introduction

Large scale computer networks are ubiquitous in today's world and underpin many aspects of modern life. With the advent of the Internet of Things, the size of these networks looks set to increase by orders of magnitude once again, and therefore routing protocols that establish and maintain connectivity in a robust and scalable manner will continue to be a priority for network operators. However, routing as a technology is still not as mature as one might think. For example, even the Border Gateway Protocol (BGP), the protocol that controls the flow of information around the internet, contains major flaws that negatively impact connectivity [31, 32, 42, 49, 66].

One reason why such problems persist is that the networking community still lacks a mathematical theory of routing in which to reason about the behaviour of routing protocols. The field of *algebraic routing* is an attempt to develop such a theory, and this thesis will address various open questions within it. Before discussing these contributions, a brief introduction to both routing in general and the field of algebraic routing is now provided.

1.1 The basics of algebraic routing

A computer network consists of a collection of computers with communication links between them and hence can be modelled as a *graph*.

> **Definition 1.** A *graph* $G = (V, E)$ is a set of nodes V linked by a set of pairwise edges E.

The edges are assumed to be directed and each edge has an associated label which contains information about the link (latency, bandwidth, reliability etc.). Figure 1.1 shows a visual representation of such a network, similar to those used throughout this thesis.

The primary purpose of a computer network is to allow the transfer of information between its nodes. Clearly it is desirable that such communication is efficient as possible and this naturally leads to the formulation of the *best-path problem*.

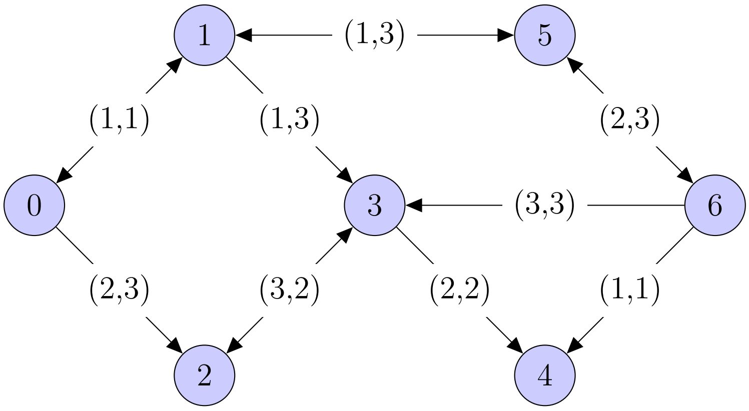


Figure 1.1: A graphical representation of a computer network. In this particular instance, the edges are labelled with the link's length (latency) and width (bandwidth).

> **Definition 2** (The best-path problem)**.** Given a network G and nodes i and j, what is the best path in G from i to j?

Although the statement of the problem is simple, it is intentionally vague on what *best* means. In practice which path is considered to be the best path depends on the aims of the network operators. For example an operator might consider the best path to be: i) the path with the lowest latency (known as the shortest-path problem), ii) the path with the highest bandwidth (known as the widest-path problem), iii) the path with the highest bandwidth that goes through node 3 or iv) the path that minimises some complex function of latency, bandwidth, reliability, loss rate, hop count, financial cost and unspecified political relationships between the different nodes in the network.

Clearly there are an infinite number of such best-path problems and each may be solved using a variety of different routing algorithms (e.g. link-state, distance-vector, path-vector). Despite the resulting combinatorial explosion of possible protocol implementations, historically the most common approach has to been to model and reason about the correctness of each protocol individually. In contrast a unified theory of routing should allow correctness results to be proved once about entire families of protocols, and then specialise these results to individual protocol implementations. The algebraic approach to routing achieves this by decomposing routing protocols as follows:

$$\text{Routing protocol} = \text{Algebra} + \text{Algorithm} \tag{1.1}$$

where the algebra specifies *which* problem is being solved (e.g. shortest-paths, widest-paths), and the algorithm specifies *how* it's being solved (e.g. link-state protocol, vector-based protocol). By considering an abstract algebra, it is then possible to study how different properties of the algebra affect the behaviour of a given algorithm.

For example, suppose that a node participating in a distance-vector routing computation currently has a best route x to a destination d. If the node then receives a new route y to d from one of its neighbours, it will first apply some policy f associated with that neighbour and produce a new candidate route $f(y)$. It will then compare x with $f(y)$ to determine which of the two routes is better. The comparison process will be represented as $\oplus$, and so the outcome of this particular comparison will be written as $x \oplus f(y)$. A very simple example is the shortest-paths problem where i) x and y are simply natural numbers representing distance, ii) $f(x) = l + x$ for some weight l, and iii) $x \oplus y = \min(x, y)$.

Classical routing theory [4, 9, 26] always assumes the following equation (or something equivalent) holds:

$$f(x \oplus y) = f(x) \oplus f(y) \tag{1.2}$$

This property is referred to as *distributivity.* The left-hand side of the equation can be interpreted as the result of the node's neighbour deciding which route, x or y, to send to the node, while the right-hand side is the result of the decision that would be made by the node itself were it to receive both routes. The equality between the two sides implies that a node's choices always agrees with the choices of its neighbours, and consequently that all nodes in the network agree with each other on what the best paths in the network are. The classical theory proves that whenever the routing problem is distributive then routing protocols will always arrive at a *globally optimal solution* where every node ends up using the best possible path available.

However distributivity does not hold in many modern routing protocols. A clear example of how distributivity violations might arise in routing can be seen in the use of *route maps* which are functions (scripts) that take routes as input and return routes as output. For example, if g and h are route maps, then another route map f can be defined as:

$$f(x) = \text{if } P(x) \text{ then } g(x) \text{ else } h(x),$$

where P is some predicate on routes (such as "does this route contain BGP community 17?"). To see how easily distributivity can be violated, suppose that:

$$\begin{array}{lcl} P(x) & = & \text{true}, \\ P(y) & = & \text{false}, \\ x \oplus y & = & x, \\ g(x) \oplus h(y) & = & h(y). \end{array}$$

Then the left-hand side of Eq 1.2 is:

$$f(x \oplus y) = f(x) = g(x),$$

while the right-hand side becomes:

$$f(x) \oplus f(y) = g(x) \oplus h(y)$$

For Eq 1.2 to hold we need $g(x) = h(y)$, which may not be the case. Indeed, if $g(x) = h(y)$ were always true, then there is no point in defining f. Perhaps the most common example of such conditional policies is *route filtering*, where $h(x)$ is equal to the invalid route.

However, even if distributivity does not hold then it is still possible for protocols to reach a routing solution. Sobrinho [62] showed that the following equation:

$$x = x \oplus f(x) \neq f(x)$$

holding is both a sufficient and necessary condition for a path-vector protocol to always converge. This property is known as *strict increasingness* and states that a route is always strictly preferred over any of its extensions, or equivalently, routes always get worse when extended. In such a case the protocol will converge to a *local optimum* where every node is assigned the best possible route given their neighbours' choices. An immediate consequence of Sobrinho's result is that any path-vector protocol can be proved correct simply by verifying that the underlying algebra is strictly increasing.

1.2 Contributions

The contributions of this thesis to the algebraic theory of routing can be divided into three areas: i) improvements to how routing problems are modelled, ii) improvements to how routing algorithms are modelled and iii) new results about the convergence properties of strictly increasing vector-based routing protocols. Along the way it also builds on prior work to develop a new, more general theory for asynchronous iterative algorithms, of which vector-based routing is just one example.

1.2.1 Improvements to the model of routing problems

In algebraic routing, all routing problems are represented as a instances of a particular algebraic structure. The set of routing problems that can be modelled therefore depends on the structure chosen and its assumed properties.

Contribution 1. This thesis proposes a new structure that is simpler than the previous state of the art. In particular the new structure models routing problems using only 5 primitives rather than the 7 primitives of Sobrinho algebras [62]. See Section 4.1 and [17, 18].

Contribution 2. The new structure is also more general as it can represent a wider selection of routing problems. More specifically, it can model path-vector operations such as path-based filtering, path inflation and tracking and removing looping paths. See Sections 4.3 & 7.2 and [17, 18].

1.2.2 Improvements to the model of routing algorithms

The algorithmic component of a protocol describes how the routing problem is solved. This includes how routes are discovered and shared between routers as well as the method of communication between routers.

Contribution 3. Reasoning about any distributed algorithm is complicated by the exponential explosion of possible message orderings between nodes. Furthermore links may delay or drop messages between nodes, or even re-order or duplicate them. Taking all of these factors into account greatly complicates correctness proofs of routing protocols. This thesis demonstrates how to decouple the underlying routing algorithm from the asynchronous environment in which it is running. In particular it uses the work of Üresin & Dubois [64] and Gurney [35] on general asynchronous iterations to further refine Equation 1.1 into:

$$\text{Routing protocol} = \text{Algebra} + \text{Synchronous algorithm} + \text{Asynchronous environment}$$

Their work is then used to derive conditions on the behaviour of the synchronous routing algorithm that are sufficient to guarantee its correctness even when run in an asynchronous environment. This significantly reduces the burden of proof, as only the synchronous algorithm needs to be directly reasoned about. See Chapter 3 and [18].

Contribution 4. The drawback of Üresin & Dubois's theory of asynchronous iterations is that it only models a *static* process in which the both the set of participants and the problem being solved remains constant over time. However these are unrealistic assumptions for many "always on" asynchronous iterative processes such as routing and consensus algorithms. This thesis therefore presents a new generalised theory of asynchronous iterations for *dynamic* asynchronous processes in which both the participants and the problem being solved may change over time. It then goes on to present conditions that are sufficient for such a dynamic process to always converge if given a sufficient period of stability. This new theory is in turn later used to construct a more accurate model of routing than found in previous work. See Section 3.3.

Contribution 5. Distance-vector and path-vector protocols have always previously been considered as different algorithms and therefore have required separate proofs of correctness. This is partly because it is difficult to model the path-vector operations directly within existing algebraic structures. As a consequence the algorithm has been required to explicitly specify the implementation details for tracking and removing paths, which in turn reduces the generality of the model. This thesis demonstrates for the first time that path-vector operations can be abstractly modelled in the algebra rather than the algorithm. This unifies the algorithms for path-vector and distance-vector protocols, allowing results on distance-vector protocols to be immediately applicable to path-vector protocols. Furthermore as the proposed algebraic model does not reveal implementation details, it can be applied to a far wider range of protocol implementations. See Sections 4.3 & 7.2 and [18].

1.2.3 New theoretical routing results

The work above is then combined to prove several new results about the convergence of strictly increasing vector-based protocols.

Contribution 6. The thesis presents the first proof that strictly increasing distance-vector protocols with finite path-weights always converge. This significantly enlarges the classes of distance-vector protocol implementations that are known to be correct. One inclusion of particular note is that it implies distance-vector protocols could be equipped with conditional policy languages. See Section 5.1 and [18].

Contribution 7. The thesis presents a new proof that strictly increasing path-vector protocols not only converge, but for the first time also converge deterministically. This means that no matter which asynchronous interleaving of messages occurs, the protocol will always reach the same final solution when run on the same network. This has significant practical implications for traffic engineering where non-deterministic convergence is known to cause routing problems that are extremely difficult to debug [28]. See Section 5.2 and [18].

Contribution 8. For the first time the above results have been proved without assuming in-order reliable delivery of messages. Although BGP uses TCP to ensure reliable in-order delivery of messages, these guarantees come at the cost of a significant increase in network traffic. This new result shows that the use of such a heavyweight mechanism is not required for correctness, and future protocols could use a more lightweight mechanism such as UDP. See Section 5.2 and [18].

Contribution 9. This thesis proves that the worst-case convergence time for strictly increasing path-vector protocols is $\Theta(n^2)$, where n is the number of nodes in the network. This is a significant improvement over previous results that only guaranteed it was at best $\Omega(n)$ and at worst $O(n!)$. More importantly it shows that even if the algebra is only strictly increasing rather than distributive, the associated routing protocol still converges in a feasible amount of time. This would not have been the case if the worst case had turned out be non-polynomial. See Chapter 6 and [17].

Contribution 10. It is widely known that reasoning about asynchronous processes is inherently tricky and a prominent source of bugs in both programs and proofs. All of the above work, including that of Üresin & Dubois and Gurney, has therefore been formalised in the Agda proof assistant. This allows the proofs to be checked by a computer, which in turn greatly increases confidence in the results. During formalisation it was found that two of Üresin & Dubois's propositions were fundamentally incorrect. Although these propositions were not directly on the proof path used by the thesis, it does serve a timely demonstration of the utility of such formalisation efforts. See Chapter 8 and [19, 68].

1.2.4 Physical artifacts

In addition to the theoretical contributions described above, the thesis has also resulted in some more tangible artefacts.

Contribution 11. The formalisation has been released as an Agda library that is capable of quickly and efficiently verifying the correctness of both hypothetical vector-based protocols and other dynamic iterative algorithms. The library provides the tools for users to implement the algebra that underlies their proposed protocol and formally verify that the resulting protocol will behave correctly in a matter of hours. Furthermore, it is designed such that the user need not understand the details of the proofs contained within. The library is available online [19] and the README file includes a guide to matching up the contents of the thesis with the associated Agda code. See Chapter 8 and [68].

Contribution 12. BGP cannot be proved correct using the above library, as the protocol implementation itself is incorrect. However using the library, this thesis does prove the correctness of an algebra containing many of the features of BGP including: local preferences, communities, conditional policies and path-inflation. This is not presented as a solution to the problems with BGP, but as a useful demonstration of how far the theory has come in that it is now capable of providing a formal proof of correctness for such a complex protocol. See Section 8.3 and [18].

1.3 Thesis outline

Chapter 2 provides an overview of routing, including a description of the Bellman-Ford algorithm and the mechanics of distance-vector and path-vector protocols. It then discusses some of the problems that existing protocols suffer from.

Next Chapter 3 provides an overview of theory of static asynchronous iterative algorithms and develops a new, more general theory of dynamic asynchronous iterative algorithms. Chapter 4 describes both past and present attempts at an algebraic theory of routing, and how distance-vector and path-vector protocols may be decomposed into an algebra and an algorithm. It then proposes and discusses several new simplifications and generalisations.

Chapter 5 uses the work from the previous chapters to prove new correctness results for both distance-vector and path-vector protocols and Chapter 6 proves a new tight worst-case convergence bound of $\Theta(n^2)$ for strictly increasing path-vector protocols.

Chapter 7 discusses how some of the inconsistencies between the theory and real-world protocols can be overcome. Chapter 8 describes the formalisation of the thesis's results in Agda, and as well as a formal proof of correctness for a new BGP-like algebra. Finally Chapter 9 concludes by providing a summary of the current state of knowledge and discusses future research directions.

Chapter 2

Routing algorithms

The problem of finding the shortest path between two points has had applications throughout history, from searching for food in primitive societies to the design of modern supply chains. Given this, it is perhaps surprising that the first deterministic procedures guaranteed to find the optimal path were only published in the late 1950s. Although there were several variants proposed, two main algorithms emerged: Dijkstra's algorithm [20] and the Bellman-Ford algorithm [7, 21]. The key ideas behind these two algorithms went on to form the basis of link-state and distance-vector routing protocols respectively. Schrijver [61] provides an overview of the early years of the theory.

This chapter first discusses destination-based next-hop routing and some of the general failure modes. It then describes the Bellman-Ford algorithm in the context of the shortest paths problem and explains how the algorithm forms a basis for distance-vector and path-vector routing protocols. This is followed by a high-level overview of Routing Information Protocol (RIP), a distance-vector protocol, and the Border Gateway Protocol (BGP), a path-vector protocol, along with a discussion of their properties and problems.

2.1 Next-hop forwarding

The dominant routing paradigm today is *destination-based next-hop forwarding.* Every node in the network has a *routing table* in which address ranges are mapped to neighbours. When a packet arrives at a node, the node looks up the neighbour associated with packet's destination in its routing table and then forwards the packet accordingly. This section discusses some of the ways in which next-hop routing can fail, independently of the method of populating the routing table.

Forwarding loops As suggested by the name, these occur when the next-hop choices of several nodes result in a forwarding path that contains a loop. Consider Figure 2.1 which shows the next-hop choices of nodes that are routing towards node 0. A message sent by node 3 will be forwarded to node 2, who in turn will forward it to node 1. Unfortunately node 1 will then forward it back to node 2, and the message will continue to circulate between nodes 1 and 2 until it times out. Node 3 will therefore be unable to communicate with node 0 until the routing loop is resolved.

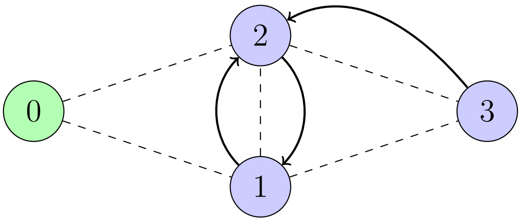


Figure 2.1: An example of a routing loop. Arrows represent the chosen next-hop towards node 0.

Routing loops come in two flavours: *transient* and *persistent*. Transient loops are temporary and can be formed during a protocol's processing of changes to the network topology, and will result in brief losses of connectivity. Persistent loops occur when such choices are permanent and result in a sustained loss of connectivity. They are therefore more serious and their presence indicates a flaw in the design of the protocol.

Routing oscillations Similar problems occur when the routing protocol itself enters a loop, and the next-hop decisions of the network continue to oscillate between several different routing solutions indefinitely. Figure 2.2 shows such an oscillation. In it there are two possible routing solutions. Either node 1 uses node 2's route to node 0 or node 2 uses node 1's route. Depending on the timing of messages between the two nodes, the routing protocol may continue to switch between the two solutions indefinitely.

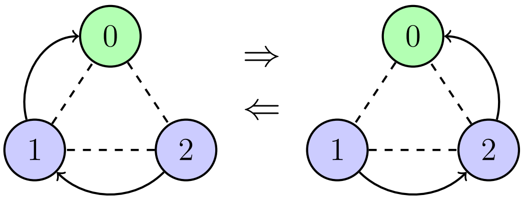


Figure 2.2: A routing oscillation

Routing oscillations may complicate the path a packet takes through the network and hence introduce significant latency and may even cause it to time out. Additionally traffic between a pair of nodes may be unpredictably split between multiple paths that may have very different properties (latency, bandwidth etc.). This in turn may cause a further increase in latency when reconstructing the sequence of packets at the destination.

Multiple stable states A third problem is *multiple stable states*. Even when the protocol starts in the same initial state, it may be the case that the final solution reached depends on the timings of the communication between nodes. An example is shown in Figure 2.3. Unlike in a routing oscillation, the two final states are stable and so the protocol cannot move between them. However which of the two states the network ends up in is determined by the exact sequence and timing of messages between the nodes.

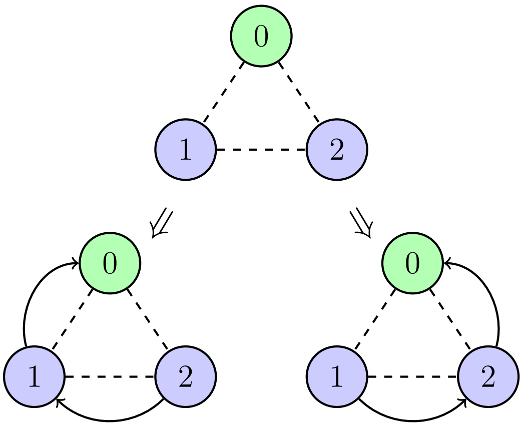


Figure 2.3: Multiple stable states

It may not be immediately obvious why this is a problem. The key insight is that routing is not a passive process in many networks. For example network operators will want to funnel traffic over cheaper, primary, high-capacity links and avoid heavy traffic over more expensive links that are only installed for backup purposes. If the final next-hop choices are non-deterministic then it becomes much harder for network operators to guarantee ending up in the desired solution. In the context of inter-domain routing, this problem is even worse because there is no one with the requisite global knowledge of the network to debug the problem.

A properly designed protocol should therefore avoid the formation of permanent routing loops and routing oscillations, as well as guaranteeing that the final next-hop choices are deterministic.

2.2 Vector-based protocols

Vector-based protocols are one of the two main families of routing protocols that are used to populate the routing tables for next-hop forwarding. It is based on the Bellman-Ford algorithm, and comes in two flavours: *distance-vector* protocols and *path-vector* protocols.

Algorithm 1: The Bellman-Ford algorithm to calculate the shortest path to a destination v_0. The network topology is represented by the $n \times n$ adjacency matrix, $\mathbf{A}$, where $\mathbf{A}_{ij}$ is the length of the link from i to j. Note that this is implicitly hard-state since each node u remembers $d[u]$, the best route it has ever seen (see Section 2.2.2.1).

```
function BellmanFord ($A, v_0$):
// Initialisation
for each node $v$ do
    $d[v] = \infty$
$d[v_0] = 0$
// Main calculation
for $i$ from 1 to $n$ do
    for each edge $(u, v)$ do
        $d[u] = \min(d[u], d[v] + A_{uv})$
```

2.2.1 Bellman-Ford algorithm for shortest paths

Given a source node v_0 and an adjacency matrix $\mathbf{A}$ where $\mathbf{A}_{ij}$ is a natural number that represents the length of the link from node i to node j, the Bellman-Ford algorithm computes the shortest distance from each node to v_0. If there is no link from i to j then this can be represented by setting the associated weight $\mathbf{A}_{ij}$ to ∞. Algorithm 1 shows a pseudocode implementation.

Like Dijkstra's algorithm, the algorithm is based on the idea of *relaxing* edges. An edge (u, v) is relaxed by checking if node u can obtain a shorter route than its current shortest route by extending the route used by v. Unlike Dijkstra's algorithm which uses global knowledge of the network to pick an efficient sequence of relaxations, Bellman-Ford applies the same sequence of relaxations regardless of the network topology. The sequence chosen performs n iterations, where each iteration relaxes every edge in the network. The time complexity for the algorithm is $O(|V||E|)$. Unsurprisingly, given that it does not optimise the ordering of the relaxations, it is less efficient than Dijkstra's algorithm which can achieve $O(|V| \log |V|)$.

The Bellman-Ford algorithm can be shown to be correct by proving the following statement "after k iterations $d[v]$ will contain the weight of the best path to node 0 with at most k hops" by induction over k. Consequently after n iterations $d[v]$ contains the weight of the best path with less than or equal to n hops, and in the absence of negative weight cycles in the graph, this must therefore be the best path in the network between v and v_0.

2.2.2 Distance-vector protocols

The Bellman-Ford algorithm only computes the best paths to a single source. Although more efficient solutions exist, the algorithm can be run n times to obtain the best paths between all pairs of nodes. Such a computation forms the basis of the family of distance-vector routing protocols. Each node's routing table stores one element from each of the $d[v]$ vectors. When a node transmits its routing table to its neighbours, the neighbour can then perform the associated relaxation steps.

Unlike the Bellman-Ford algorithm, nodes exchange their routing tables with each other asynchronously and so, even in identical networks, the order in which relaxations occur will vary between executions. An example execution is shown in Figure 2.4 for the computation of the shortest-paths towards node 0. Note that the asynchronous nature of the protocol allows states to be reached that would never occur in the synchronous Bellman-Ford computation. For example between steps 6 and 7 of the execution, node 2 temporarily uses the *longest* possible path in the network.

In general the number of possible orderings of messages grows exponentially in the size of the network. Proving the correctness of vector-based protocols therefore requires reasoning about all possible message orderings and hence is far harder than than proving the correctness of the Bellman-Ford algorithm. Further complexity is added by the fact that some protocols allow route advertisements to be lost, reordered or even duplicated (see Section 2.2.2.2).

2.2.2.1 Soft state vs hard state

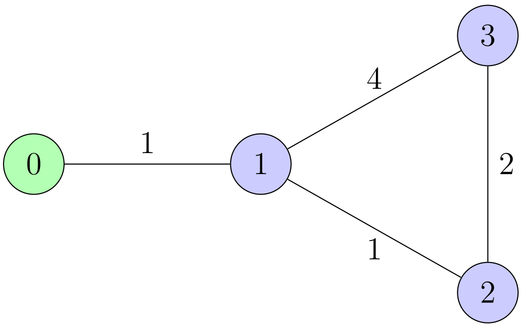
The content and timing of routing update messages varies from protocol to protocol. Broadly there are two approaches: *soft state* and *hard state*.

In soft state protocols the entire routing table is retransmitted at regular intervals, and so it is unnecessary for nodes to permanently store a neighbour's routing table. Such protocols are therefore better at recovering from node failures, as all relevant information is regularly retransmitted. However they require more network resources as they transmit larger messages more frequently.

In hard state protocols each node permanently stores the latest versions of their neighbours' routing tables. Therefore a node only needs to send an update message when its own routing table changes, and even then it need only transmit the relevant changes rather than the entire table. This reduces network traffic but makes recovering from failures more difficult as the state needs to be regenerated.

2.2.2.2 Reliable vs unreliable communication

Another set of trade-offs is the mechanism for inter-node communication. Routing protocols assume that connectivity across individual links is provided by a suitable transport-layer protocol. However different transport-layer protocols provide different communication guarantees.



Node			
0	1	2	3
0	∞	∞	∞
0	1	∞	∞
0	1	∞	5
0	1	7	5
0	1	2	5
0	1	2	4

1. Initially node 0 has a distance of 0, and every other node has no route (represented by a distance of ∞). Node 0 then sends an update message to node 1, advertising that it has a route to itself of distance 0.

2. Node 1 receives the advertisement from node 0, performs the relaxation step and installs a distance of 1 in its routing table entry for node 0. Node 1 will then send out update messages to nodes 0, 2 and 3.

3. Node 0 receives the advertisement of 1 from node 1. Node 0 will perform the relaxation but rejects the route as 2 is longer than its current distance of 0.

4. Node 3 receives the advertisement of 1 from node 1, and accepts it and installs a distance of 5 in its routing table. Node 3 sends update messages to nodes 1 and 2.

5. Node 1 receives the advertisement of 5 from node 3, but rejects it as 5 is longer than its current route of 1.

6. Node 2 receives the advertisement of 5 from node 3, and accepts it as 7 is better than its current value of ∞. Node 2 then sends update messages to nodes 1 and 3.

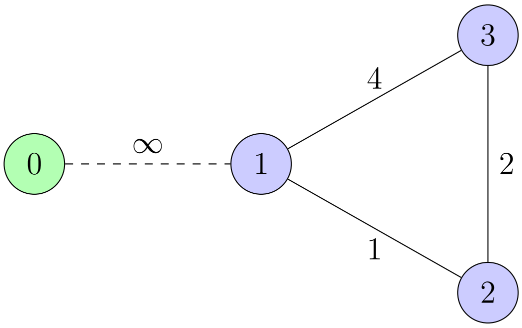
7. Node 2 receives the advertisement of 1 from node 1, and accepts it as 2 is better than its current value of 7. Node 2 then sends updates messages to nodes 1 and 3.

8. Node 3 receives the advertisement of 7 from node 2, but rejects it as 9 is longer than its current route of 7.

9. Node 3 receives the advertisement of 2 from node 2, and accepts it as 4 is better than its current value of 7.

10. The remaining advertisements still in flight are received in some order and all rejected.

Figure 2.4: An example network and one possible execution order of a distance-vector protocol. The table shows the routing table entries for each node for destination node 0.



Node			
0	1	2	3
0	1	2	4
0	3	2	4
0	3	6	4
0	3	6	8
0	7	6	8
0	7	8	8
...	...	...	...

Figure 2.5: Count-to-convergence for the shortest-paths problem. Starting from the state reached at the end of Figure 2.4, the link between node 0 and node 1 has failed. The routing table entries for the other nodes continue to count up to infinity.

Physical layer links are inherently unreliable. For instance messages may be dropped if the link bandwidth is exceeded. Furthermore pairs of messages may arrive at their destination in a different order than they were sent and the same message may even arrive twice. Lightweight transport-layer protocols such as the User Datagram Protocol (UDP) [54] make no effort to conceal this and instead provide best-effort communication. Hence routing protocols built on top of UDP-like transport-layer protocols must necessarily be robust enough to cope with such communication anomalies.

In contrast heavy-weight transport-layer protocols such as the Transmission Control Protocol (TCP) [55] provide better guarantees. By using more complex mechanisms such as packet sequence numbers and acknowledgement messages, TCP ensures that the destination node can reconstruct the exact sequence of packets as they were sent by the source node. These guarantees come at a higher cost in both terms of both network traffic and latency.

One important consideration is the interaction between communication guarantees and the state-model used by a protocol. A soft state protocol can handle missed messages as the entire state is regularly retransmitted, and therefore can use either reliable or unreliable delivery. In contrast, a hard-state protocol requires reliable communication as each update is only sent once and hence lost or re-ordered updates cannot be recovered from.

2.2.2.3 Count-to-convergence

Naively designed distance-vector protocols suffer from the *count-to-infinity* problem which generates both routing loops and routing oscillations. In the shortest paths problem, count-to-infinity occurs when links in the network either fail or their length increases.

Figure 2.5 demonstrates an example of count-to-infinity. Initially the network is in the final state reached in Figure 2.4. The link between node 0 and node 1 then fails. Nodes 1, 2 and 3 therefore have no path to node 0. One might expect the protocol to realise this. However in a basic distance-vector protocol this does not occur as nodes 1, 2 and 3 continue to advertise the route to each other even though it no longer exists. Node 1 adopts node 2's route to 0, node 2 adopts node 3's route, node 3 adopts node 2's route and so on. Every time the length of the "supposed" paths will increase. If the protocol enforces a maximum path length then the protocol will eventually reach this upper limit and then converge. In this case the problem is known as *count-to-convergence*. If not then the protocol will continue to count to infinity without ever realising there is no longer a route to node 0.

2.2.3 Path-vector protocols

A path-vector protocol is a particular type of distance-vector protocol designed to avoid counting problems. Counting problems occur in distance-vector protocols because they allow the indefinite propagation of *junk routes* that no longer exist in the network. Detecting such routes immediately is impossible without global knowledge of the network topology. Instead the insight is that if a node receives a route that it itself advertised, then that route must be junk and so should be ignored.

Path-vector protocols therefore transmit the path along which the route was generated as well as the route's value. A node will automatically reject any route that already contains itself in its path. This mechanism eventually halts the continued spread of junk routes throughout the network. However transmitting the path and the value is significantly more expensive than transmitting the value alone, and therefore path-vector protocols may not be suitable for all situations.

2.3 Example protocols

This section presents a brief overview of two vector-based protocols. It is important to stress that the aim of this thesis is not to reason about any particular protocol, but instead to reason about the fundamental mathematical properties of the vector-based approach to routing. Therefore this section omits many of the more intricate implementation details particular to each protocol. Instead it only describes their operation and their associated problems at a high-level, in order to motivate the fundamental questions posed and answered later on in this thesis.

2.3.1 RIP

The Routing Information Protocol (RIP) [47] is one of the oldest examples of distance-vector routing protocols. Its aim is to compute the path between two nodes with the smallest number of nodes visited (referred to as the *hop-count*). This routing problem is therefore an instance of the shortest-paths problem where every link has a length of 1. RIP is a soft-state protocol and by default routers will locally broadcast their routing table to their neighbours every 30 seconds. Accordingly it uses UDP for inter-node communication and so there is no guarantee that routing advertisements are delivered reliably.

When the topology of the network changes, RIP may suffer from count-to-convergence. RIP uses a combination of different techniques to combat this problem. It employs *split-horizons* and *poisoned chalice* optimisations to reduce (but not eliminate) occurrences of count-to-infinity. It then imposes a limit of 16 hop counts on all paths through the network. This essentially turns the count-to-infinity problem into the count-to-convergence problem. This small maximum hop-count means that after link failure it requires at most 8 update periods to re-converge.

2.3.1.1 Problems

Although it only requires 8 update periods, each period is 30 seconds long. This means that in the worst case it will still require approximately 4 minutes to realise a node is no longer reachable. Additionally the artificial imposition of the maximum hop-count of 16 means that the protocol can no longer be deployed in networks with diameter greater than 16. Hence usually RIP is only deployed as a quick, light-weight solution for routing in small, local networks.

2.3.2 BGP

The wider internet has a much richer structure than a typical network and in reality is better viewed as a network of networks. As shown in Figure 2.6, the backbone of the internet is made up of individual networks called Autonomous Systems (ASs). ASs are usually privately owned by service providers and therefore have significant autonomy in the routing protocols that they run locally. However to communicate with each other ASs use the Border Gateway Protocol (BGP) [57]. BGP is a hard-state protocol which uses TCP to guarantee reliable communication between routers, and is the main path-vector protocol in use today.

However routing in such in an environment bears little resemblance to finding the shortest path. Transmitting large volumes of traffic is expensive and an AS will not be willing to carry the traffic from another AS for free and so routing decisions are often affected by additional economic concerns. In order to facilitate such fine grained routing

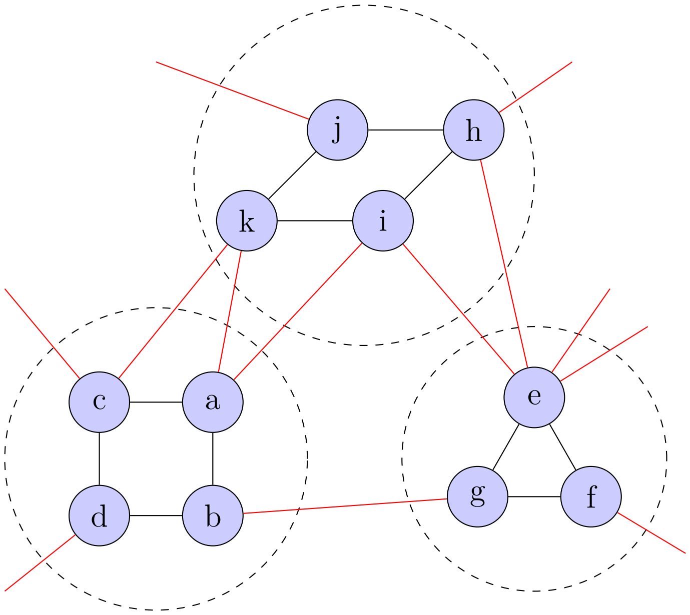


Figure 2.6: The internet is split up into Autonomous Systems (ASs), each of which are independently owned networks.

decisions BGP has a policy language designed to allow network operators to write complex methods of filtering and re-ordering a router's preferences.

Correspondingly the associated procedure to choose between two alternative routes is more complex than that of RIP and takes over 7 pages to describe [57]. Consequently only a *very* high-level view of the process is described here.

1. First the route with the highest *local preference* is chosen. A route's local preference is simply a number that may be set at will by routers to signify how desirable they find the route.

2. If tied then the route that goes through the smallest number of ASs is chosen.

3. If still tied then the route that was advertised from the AS with the smallest ID number is chosen.

4. Finally if both routes originate from the same AS, the route with the lowest *multi-exit discriminator* (MED) attribute is chosen. The MED attributes provide ASs with the ability to influence how traffic from other ASs enter their network to reach certain destinations.

2.3.2.1 Problems

Whereas RIP could be modelled within a sound theoretical framework for best-path problems at the time of its development, BGP firmly lay outside of existing routing theory, and to a certain extent still does. Like many complex systems engineered by very capable people, it does a great job at solving a very complicated problem most of the time. However over the years it has become increasingly clear that it suffers from both of the problems of non-convergence and non-deterministic convergence described in Section 2.1.

Starting in the late 1990s, various work showed that it was possible for BGP to oscillate forever. This was either caused by conflicting AS policies [32, 42, 49, 66] or by the implementation of the MED attribute [31]. Furthermore Griffin and Huston [28] showed that it was possible for a network running BGP to non-deterministically end up "stuck" in a different stable state than that intended by the network operators. They named these occurrences "BGP wedgies". BGP wedgies are particularly pernicious as they arise from the interaction of internal policies of multiple different ASs. Without a global view of the network, it is very difficult for any one AS to diagnose their cause. Furthermore having identified such problems, it still requires coordination between the competing ASs in order to move the system from the problematic stable state to the desired stable state.

The question of how to guarantee that BGP always converges and ensure that it does so in a deterministic manner has been an active subject of research over the last twenty years. In the process of reasoning about BGP, this has triggered a revival in the much broader field of algebraic path problems. Such research is applicable to *all* routing protocols, not just BGP, and is capable of providing deeper insights into the very nature of routing computations. A discussion of the algebraic routing literature can be found in Section 4.5. This chapter will restrict itself to discussing the BGP-specific results.

2.3.2.2 Gao-Rexford conditions

The observation by Huston [38] that many typical inter-domain policies are based on the commercial relationships between ASs has motivated much of the work on BGP. In particular AS relationships often obey a *customer-peer-provider* model. In such a model pairs of ASs are related in two ways: i) they may be in a *customer-provider* relationship where the customer pays the provider for access to the wider internet or ii) they may be in a *peer-to-peer* relationship, in which both ASs allow the other's customer traffic to travel across their network.

Gao and Rexford [23] showed that if these commercial relationships are universally followed then it is sufficient to ensure the convergence of BGP. More precisely they state that:

Theorem 1. If the customer-provider relation forms a directed acyclic graph then BGP will always converge to a stable state, even after the removal of arbitrary nodes and links, as long as all ASs' policies obey the following rules:

1. A customer AS may only export its own routes and the routes of its customers to its

provider. It may not export routes learnt from its peers or its provider.

2. A provider AS may export any of its routes to its customers.
3. A peer AS may only export its own routes and the routes of its customers to its peers. It may not export routes learnt from other peers or its provider.
4. An AS must always choose a route from a customer over a route from a peer or a provider.

Proof. Theorem 5.1 in [23]. □

In other words if the conditions above hold then the oscillations in BGP are eliminated. Importantly this still holds true after node and link failure. However there are several drawbacks to the Gao-Rexford conditions:

Unnecessarily restrictive It is not necessarily the case that customer-provider and peer-to-peer are the only economic relationships in the wider internet. For example, whilst the constraint of always choosing a customer route over a peer or a provider route aligns with economic incentives (as sending traffic through a customer is effectively free), it seems unlikely that there is no conceivable circumstance in which an AS might prefer a customer or a peer route. However, even if these are the only relationships in the internet today, then this may not remain the case in networks in the future.

Subsequent work in algebraic routing (described in Chapter 4) has uncovered a more fundamental reason why the Gao-Rexford conditions are sufficient for the convergence, by showing that they are a special case of a much broader set of constraints on topology and policies that are sufficient and necessary for a path-vector protocol to converge over a particular network. One immediate consequence of this is that the Gao-Rexford conditions therefore exclude many valid routing policies.

Global coordination Furthermore, even assuming that the conditions are liberal enough to be globally adopted by every AS in the internet, it is still a non-trivial task to verify that the conditions hold. For example the acyclicity of the customer-provider graph requires global knowledge of the economic relationships between the ASs. In practice, ASs are often reluctant to reveal this commercially sensitive information. Gao & Rexford suggest a global internet registry that would store and analyse the commercial relationships in order to prevent the formation of loops. Unfortunately, global coordination in the internet is well known to be notoriously difficult to achieve, even when there are clear economic incentives for all parties involved, e.g. the rollout of IPv6 [39].

Re-verification One subtlety is that the theorem imposes conditions on the topology of the network, not just the AS policies. Consequently every time a new link or node is added to the network, the conditions require reverification. In a network as large as the internet this constant reverification might be a significant burden.

Non-deterministic convergence Lastly Theorem 1 only guarantees convergence to some stable state rather than one specific stable state. Hence the result does not guarantee that the network is free of BGP wedgies.

2.4 Motivation for algebraic routing

The short-comings of both BGP and Theorem 1 give rise to several of the motivating questions for algebraic routing. In particular for a general vector-based protocol:

1. What is the weakest set of constraints on policies that guarantees the protocol converges?
2. Can these constraints be made independent of the network topology so that global coordination is not required?
3. How do different sets of constraints affect the rate of convergence of the protocol?

The following chapters will discuss prior work on these questions and present results that provide new pieces to the puzzle.

Chapter 3

Asynchronous iterative algorithms

One of the key contributions of this thesis is the insight that vector-based routing protocols are just one instance of a more general class of asynchronous iterative algorithms. This unlocks a large, existing body of literature about conditions that are sufficient and necessary for such algorithms to deterministically converge to a fixed point. Crucially these conditions guarantee the convergence of the asynchronous iteration only using properties of the synchronous iteration, hence removing the need to reason directly about the asynchronous iteration.

Initially the chapter summarises the related literature and highlights some useful results. Subsequently it describes two new contributions to the theory of asynchronous iterations. First it demonstrates that the conditions required by existing convergence results may be relaxed. In particular it shows that there is no need to prove that the underlying state space of the iteration can be partitioned into a series of boxes nested inside each other.

The second contribution is a generalisation to the model of the iteration itself. The existing literature only considers a single *static* iteration in which all nodes continue to participate. However in routing (and in other applications such as consensus algorithms) both the participants in the computation and the very computation itself will change over time. Although at first glance it appears possible to incorporate this into the existing static model, this chapter will explain why it is not possible to do so in practice.

To address this short-coming, the chapter develops a new, more general model for *dynamic* asynchronous iterations in which both the set of participants and the underlying iteration may change over time. Two new convergence results, based on generalisations of existing results from the literature, are then proved for the dynamic model. This more general theory of dynamic iterations will form the basis of the model of vector-based protocols presented in Chapter 4 and the proofs of convergence will be used in Chapter 5 to prove new, stronger correctness results for vector-based protocols.

Appendix A contains some common definitions that may be useful to the reader. All of the results in this chapter have been formalised in Agda.

3.1 Iterative algorithms

3.1.1 Synchronous iterations

One approach to computing a fixed-point of a function is to iterate by starting in some initial state and then repeatedly applying the function to create progressively better estimates of the fixed point. Assume there exists a set S and a function $F : S \to S$.

> **Definition 3.** The *synchronous state function*, $\sigma : \mathbb{N} \to S \to S$, is defined as follows:
>
> $$\sigma^t(x) = \begin{cases} x & \text{if } t = 0 \\ F(\sigma^{t-1}(x)) & \text{otherwise} \end{cases}$$
>
> where $\sigma^t(x)$ is the state of the iteration at time t having started in the initial state x[1].

An iterative algorithm attempts to find a fixed point for F by starting at some initial state x and computing the following sequence (also known as the orbit of x [41]):

$$\sigma^0(x),\ \sigma^1(x),\ \sigma^2(x),\ \sigma^3(x),\ ...,\ \sigma^k(x),\ \sigma^{k+1}(x),\ ...$$

If at any point $\sigma^k(x) = \sigma^{k+1}(x)$ then $\sigma^k(x)$ is the desired fixed point for F. Whether or not a fixed point is ever reached depends on the properties of F and x. There is a significant body of literature on such properties [12, 58].

One particular case is now considered. Suppose there exists a distance function $d : S \times S \to \mathbb{N}$ [2]. The function d is referred to as a *distance function* rather than a *metric* as later in this chapter it may be assigned properties that are either weaker or stronger than those of a standard metric. For the moment it is assumed that it need only obey the equality axioms (i.e. $d(x, y) = 0 \Leftrightarrow x = y$) and is therefore a quasi-semi-metric (see Appendix A for definitions).

> **Definition 4** (Definition 4.12.1 in [37])**.** A function $F : S \to S$ is *strictly contracting* with respect to a distance function d over a set $A \subseteq S$ if:
>
> $$\forall x, y \in A : x \neq y \Rightarrow d(F(x), F(y)) < d(x, y)$$

[1] In the dynamical systems literature [41] $\sigma^t(x)$ is more commonly written as just $x(t)$ or $F^t(x)$. However the former notation makes it difficult to differentiate between the synchronous and asynchronous state functions and the latter runs into problems after other superscripts are added to F in Section 3.3.

[2] It is more common for the image of d to be $\mathbb{R}^+$ rather than $\mathbb{N}$. However later lemmas in this chapter require that $<$ is a well-founded relation over the image. Furthermore this thesis only considers sets S which are discrete/equipped with non-dense orderings. In such a case it is unnecessary for the distance between two elements to be a member of a densely ordered set such as $\mathbb{R}^+$.

The function F being strictly contracting with respect to d is sufficient to guarantee it has a unique fixed point (see Lemmas 1 & 2). However, in practice the function being strictly contracting is an unnecessarily strong condition, and there are occasions in this thesis where it is impossible to construct a distance function for which F, the function of interest, is strictly contracting. Accordingly several weaker notions of contraction that still guarantee the existence of a unique fixed point are now considered.

> **Definition 5** (Definition 4.12.1 in [37]). A function $F : S \rightarrow S$ is *strictly contracting on orbits* with respect to a distance function d over a set $A \subseteq S$ if:
>
> $$\forall x \in A : x \neq F(x) \Rightarrow d(F(x), F^2(x)) < d(x, F(x))$$

Lemma 1 (Simplification of Theorem 1 in [56]). If F is strictly contracting on orbits with respect to d over a non-empty set $A \subseteq S$ and A is closed over F then F has a fixed point.

Proof. Let x be an element in A. While $F^k(x) \neq F^{k+1}(x)$, the strictly contracting on orbits property can be used to construct the following chain:

$$d(x, F(x)) > d(F(x), F^2(x)) > d(F^2(x), F^3(x)) > ...$$

This is a decreasing chain in $\mathbb{N}$ and so must have finite length. Therefore there must exist a time t such that $F^t(x) = F^{t+1}(x)$ and so $F^t(x)$ is the required fixed point. □

> **Definition 6** (Definition 4.12.1 in [37]). A function $F : S \rightarrow S$ is *strictly contracting on an element* e with respect to a distance function d over a set $A \subseteq S$ if:
>
> $$\forall x \in A : x \neq e \Rightarrow d(e, F(x)) < d(e, x)$$

Lemma 2 (Simplification of Theorem 1 in [56]). If F is strictly contracting on a fixed point x^* with respect to a set $A \subseteq S$ and A is closed over F then x^* is the unique fixed point in A.

Proof. Suppose there exists another fixed point y^* such that $x^* \neq y^*$ then the following is immediately obtainable via the strictly contracting property:

$$\begin{aligned} d(x^*, y^*) &= d(x^*, F(y^*)) \\ &< d(x^*, y^*) \end{aligned}$$

This is clearly a contradiction and so $x^* = y^*$. □

Therefore to show that σ will always converge to a unique fixed point it is sufficient to show that there exists a distance function for which F is strictly contracting on orbits and strictly contracting on the resulting fixed point. Note that if a distance function is strictly contracting then it is immediate from the definitions that it is both strictly contracting on orbits and strictly contracting on every element.

3.1.2 Asynchronous iterations

An iterative algorithm is *parallelisable* if both the state space S and the function F are decomposable into n parts:

$$S = S_1 \times S_2 \times ... \times S_n$$
$$F(x) = (F_1(x), F_2(x), ..., F_n(x))$$

where each function $F_i : S \to S_i$ takes a state and calculates the i^{th} component of the new state. This decomposition can be used to parallelise the iteration, either over several cores on a single computer or over multiple computers. The theory is agnostic as to this choice, and so going forwards each computational unit will simply be referred to as a "node". Also note that the model does not necessarily require every node to compute the same type of data, as the set S_i need not be the same as S_j.

In the parallelised iteration every node has its own view of the global state and the i^{th} node applies F_i periodically to generate an updated value for its own state. It then broadcasts this new value to the other nodes for use in their calculations. If the nodes collectively synchronise their updates and message timings then the end result will be the same computation as σ, the original synchronous iteration (albeit a more efficient version). However such synchronisation is expensive and may not always be practical. In such situations the computation is carried out *asynchronously*, where no attempt is made to synchronise a node's update and message timings with those of the other nodes participating in the iteration.

The properties of an asynchronous iteration may be different from that of its synchronous counterpart. For example, in Section 2.2.2 it was shown that an asynchronous routing computation may reach states that were previously unreachable by the associated synchronous iteration. It is therefore possible for the synchronous version of an iteration to always converge and yet the asynchronous version to diverge.

3.2 Static asynchronous iterations

3.2.1 Model

A mathematical model for asynchronous iterations was standardised by work in the 1970s and 80s [5, 8, 63], although notation continues to vary. The notation and terminology found in this thesis broadly follows that used by Üresin & Dubois [64]. It will be noted explicitly where it does diverge from that of Üresin & Dubois, usually either to improve clarity or to avoid clashes with other notation.

Assume that the set of times T is a discrete linear order. Each point in time marks the occurrence of events of interest: for example a node computing an update or a message arriving at a node. The set of times can be represented by $\mathbb{N}$ but for notational clarity T will be used. Additionally let $V = \{1, 2, ..., n\}$ be the set of nodes that are participating in the computation.

> **Definition 7** (Definition 1 in [64]). A *static schedule* consists of a pair of functions:
>
> - $\alpha : T \to 2^V$ is the *activation function*, where $\alpha(t)$ is the set of nodes which activate at time t.
> - $\beta : T \times V \times V \to T$ is the *data flow function*, where $\beta(t, i, j)$ is the time at which the latest message node i has received from node j at time t was sent by node j.
>
> such that:
>
> (SS1) $\forall i, j, t : \; \beta(t + 1, i, j) \leq t$
>
> (SS2) $\forall i, t : \exists k : \; i \in \alpha(t + k)$
>
> (SS3) $\forall i, j, t : \exists t' : \forall k : \; \beta(t' + k, i, j) \neq t$

The function α describes when nodes update their values, and the function β tracks how the resulting information moves between nodes. Assumption (SS1) enforces causality by stating that information may only flow forward in time. Assumption (SS2) guarantees that every node continues to activate indefinitely and assumption (SS3) states that every pair of nodes continues to communicate indefinitely. Note that although assumption (SS3) guarantees that new messages will eventually arrive, it does not forbid the data flow function β from delaying, losing, reordering or even duplicating messages. Schedules are therefore capable of describing both reliable and unreliable communication between nodes (see Section 2.2.2.2).

Note that in their original paper, Üresin & Dubois use a model in which the nodes communicate via shared memory, and so their definition of β takes only a single node i.

However, in distributed processes (e.g. internet routing) nodes communicate in a pairwise fashion. In the definition above β has therefore been augmented to take two nodes, a source and destination. Üresin & Dubois's original definition can be recovered by providing a function β that is constant in its third argument.

> **Definition 8** (Definition 1 in [64])**.** Given a schedule (α, β) the *asynchronous state function*, $\delta : T \to S \to S$, is defined as follows:
>
> $$\delta^t_i(x) = \begin{cases} x_i & \text{if } t = 0 \\ \delta^{t-1}_i(x) & \text{else if } i \notin \alpha(t) \\ F_i(\delta^{\beta(t,i,1)}_1(x), \delta^{\beta(t,i,2)}_2(x), ..., \delta^{\beta(t,i,n)}_n(x)) & \text{otherwise} \end{cases}$$
>
> where $\delta^t_i(x)$ is the state of node i at time t when the iteration starts from state x.

At time 0 the process is in the initial state x. At subsequent times t if node i is not in the set of active nodes then its state remains unchanged. Otherwise if node i is in the active set of nodes it applies its update function F_i to its *current view* of the global state. For example $\delta^{\beta(t,i,1)}_1(x)$ is the state of node 1 at the time of departure of the most recent message node i has received from node 1.

The synchronous state function, σ, can be recovered by using the synchronous schedule where $\alpha(t) = V$ and $\beta(t, i, j) = t - 1$, (i.e. at each time step every node activates and all messages only take a single time step to propagate).

Several generalisations to this basic model have been proposed previously. For example, Miellou et al. [50] show how the model may be extended to the case where the application of F_i is not atomic, and messages may be dispatched to other nodes during the middle of an update. Although applicable to some asynchronous processes, this generalisation is not relevant to routing where routers only dispatch update messages after having fully processed the received route advertisement.

3.2.2 Results

Before discussing prior work on when δ converges, it is first necessary to establish what is now meant by convergence in an asynchronous environment.

> **Definition 9.** The asynchronous iteration *converges non-deterministically over a set* $X \subseteq S$ if for all schedules and starting states $x \in X$ there exists a fixed point x^* for F and a time t such that $\delta^{t+k}(x) = x^*$ for all k.

> **Definition 10.** The asynchronous iteration *converges deterministically over a set* $X \subseteq S$ if there exists a fixed point x^* for F such that for all schedules and starting states $x \in X$ there exists a convergence time t such that $\delta^{t+k}(x) = x^*$ for all k.

Non-deterministic convergence only implies that the iteration will find one fixed point out of many, and that the exact fixed point found may depend on the schedule and the starting state. In contrast, deterministic convergence guarantees that the fixed point found will be the same no matter the starting state and the schedule. Many of the more recent correctness proofs in routing (see Section 2.3.2.2 & Section 4.5) only prove non-deterministic convergence. However the literature on asynchronous iterations, and therefore the remainder of this chapter, only considers the stronger and more desirable notion of deterministic convergence.

The survey paper by Frommer and Szyld [22] provides an overview of the results in the literature for this and other related models. Much of the work has been motivated by iterative algorithms in numerical analysis and consequently many of the proofs of convergence assume that the set S is equipped with a dense ordering. Unfortunately in fields such as routing, consensus algorithms and others, the set S is discrete, and so many of the more common results are inapplicable. However in the late 1980s Üresin and Dubois [64] came up with one of the first conditions for the convergence of discrete asynchronous iterations.

> **Definition 11** (Definition 2 in [64])**.** A function F is an *asynchronously contracting operator* (ACO) if there exists a sequence of sets $B(k) = B(k)_1 \times B(k)_2 \times ... \times B(k)_n$ for $k \in \mathbb{N}$ such that:
>
> (SA1) $\forall k : B(k+1) \subseteq B(k)$.
>
> (SA2) $\forall k, x : x \in B(k) \Rightarrow F(x) \in B(k+1)$.
>
> (SA3) $\exists k^*, x^* : \forall k : k^* \leq k \Rightarrow B(k) = \{x^*\}$.

Theorem 2 (Theorem 1 in [64])**.** If F is an ACO then δ converges deterministically over the set $B(0)$.

Proof. See [64]. □

The ACO conditions require that the state space S can be divided into a series of nested boxes $B(k)$ where every application of F moves the state into the next box, and eventually a box $B(k^*)$ is reached that only contains a single element. See Figure 3.1 for a visualisation. The reason why these conditions guarantee asynchronous convergence, rather than merely synchronous convergence, is that each box must be decomposable over each of the n nodes.

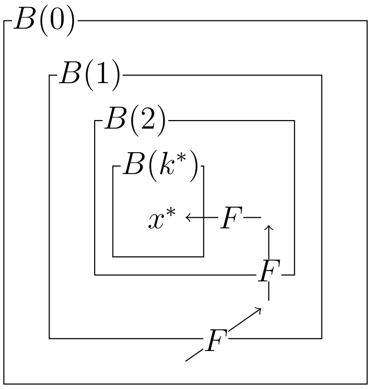


Figure 3.1: If F is an ACO then the space S can be divided up into a series of nested boxes B. Note that this figure is a simplification, as each set $B(k)$ is decomposable into $B(k)_1 \times ... \times B(k)_n$ and so in reality the diagram should be n dimensional.

Therefore the operator is always contracting even if every node hasn't performed the same number of updates locally. Note that Theorem 2 only guarantees δ will converge from states in the initial set $B(0)$. Hence $B(0)$ can be thought of as a basin of attraction [51]. When applied to routing in Chapter 5, it is desirable to prove that the protocol will converge from any state and hence the boxes will constructed such that $B(0) = S$.

Additionally there are two slight differences between Definition 11 and how it is stated in [64]. Firstly the sets have been renamed from D to B to avoid conflicting with later notation. Secondly for (SA1), Üresin & Dubois originally assumed $\forall k : B(k+1) \subset B(k)$. This relaxation of this condition to $\forall k : B(k+1) \subseteq B(k)$ is widely known and is used near universally in later literature (e.g. [22]).

Some additional intuition about these conditions may be gained by considering the example operators in Figure 3.2. There are two nodes participating in the iteration, and the cells represent possible states. The arrow associated with each cell represents the result of applying F to that state. The operator is an ACO if it is possible to construct a sequence of contracting boxes such that:

1. Each box is a quadrilateral (otherwise the box is not decomposable).

2. Each box is contained inside the previous box (SA1).

3. Each newly excluded cell has an arrow into the current box (SA2).

4. The final box only contains the fixed point (SA3).

The advantage of the ACO conditions is that they are independent of both the asynchronous state function and the schedule. Therefore proving that δ converges only requires reasoning about the function F. Additionally, the ACO conditions are minimal at least some of the time, as Theorem 2 in [64] proves that F being an ACO is also a necessary condition for convergence when S is finite.

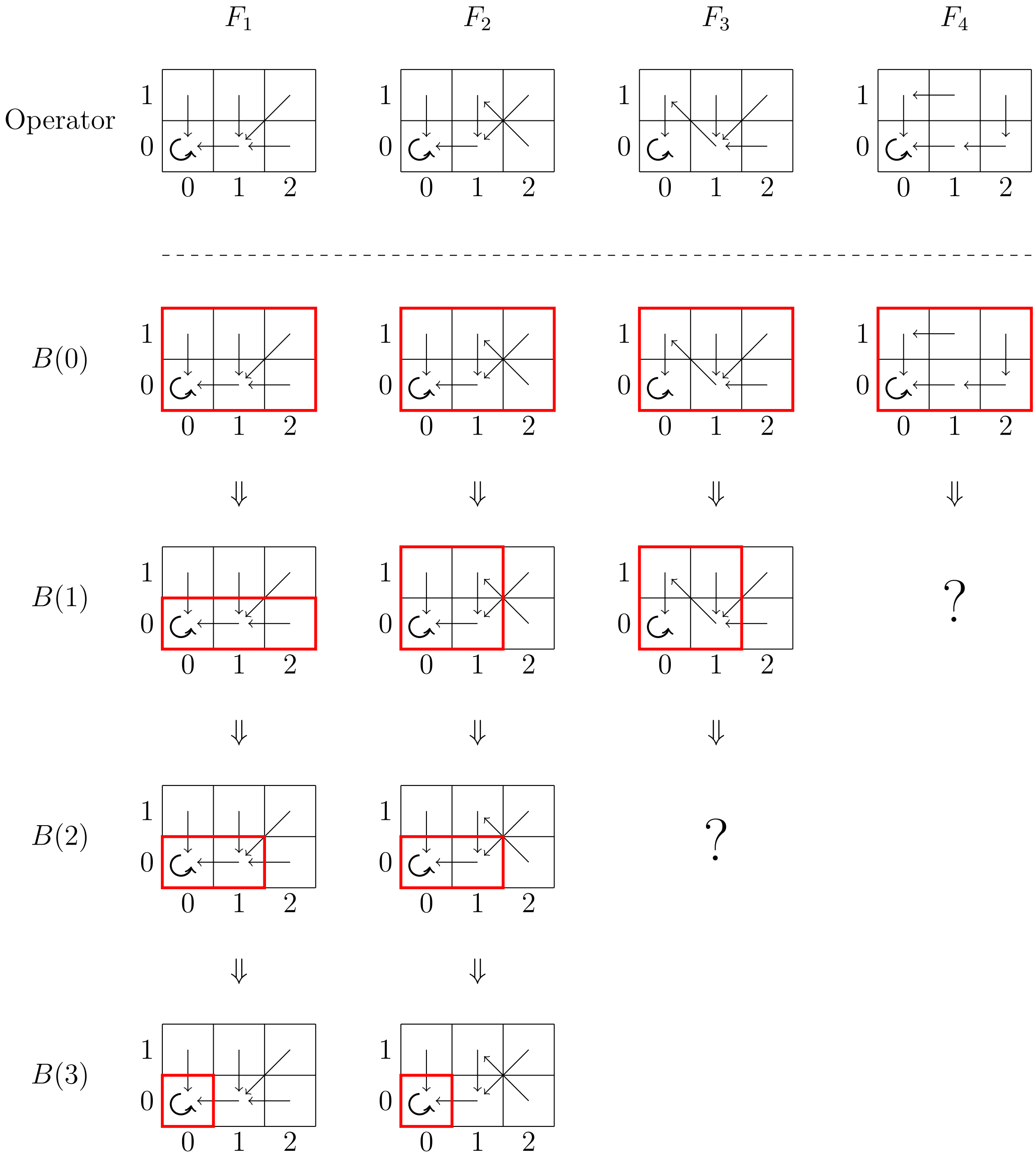


Figure 3.2: Examples of 2-node operators. The cells represent possible states and the arrow associated with each cell represents the result of applying the operator to that state. The operators F_1 and F_2 are ACOs as a sequence of nested boxes can be constructed obeying all the required assumptions. F_3 and F_4 are not ACOs as it is impossible to construct boxes $B(2)$ and $B(1)$ respectively that obey (SA2).

However in practice the set of boxes B can be difficult and non-intuitive to construct. Üresin & Dubois recognised this and provided several other conditions that are sufficient to construct an ACO[3]. An alternative set of conditions were recently described by Gurney [35].

> **Definition 12.** A function F is an *asynchronously metrically contracting operator* (AMCO) if for every node i there exists a distance function d_i such that:
>
> (SU1) $\forall i : d_i$ is an ultrametric
>
> (SU2) $\forall i : d_i$ is bounded
>
> (SU3) F is strictly contracting on orbits w.r.t. D over S
>
> (SU4) F is strictly contracting on x^* w.r.t. D over S for any fixed point x^*
>
> (SU5) S is non-empty
>
> where $D(x, y) = \max_i d_i(x_i, y_i)$,

The definition of an ultrametric may be found in Appendix A. The AMCO conditions require the construction of a notion of distance between states (SU1) such that there exists a maximum distance (SU2) and that successive iterations become both closer together (SU3) and closer to any fixed point (SU4). Note that in practice a unique fixed point must exist by Lemmas 1 & 2. The assumption that S is non-empty (SU5) ensures that the iterative argument in Lemma 1 has a starting point.

Gurney [35] proves that the AMCO conditions are equivalent to the ACO conditions by constructing reductions in both directions. Consequently the following convergence theorem holds.

Theorem 3 (Theorem 5 in [35])**.** If F is an AMCO then δ converges deterministically over the set S.

Proof. See [35]. □

3.2.3 Unstructured spaces

Both the ACO and the AMCO conditions require that the space have regular structure. For the ACO conditions this is encoded by the boxes being nested (SA1), and for the AMCO conditions that S is an ultrametric space (SU1). However this thesis is the first to note that the full power of these assumptions is not needed and so the conditions may

[3] Several of these conditions have since been shown to be wrong by Zmigrod, Daggitt & Griffin [68], who demonstrated counter-examples that satisfied the conditions but did not converge deterministically.

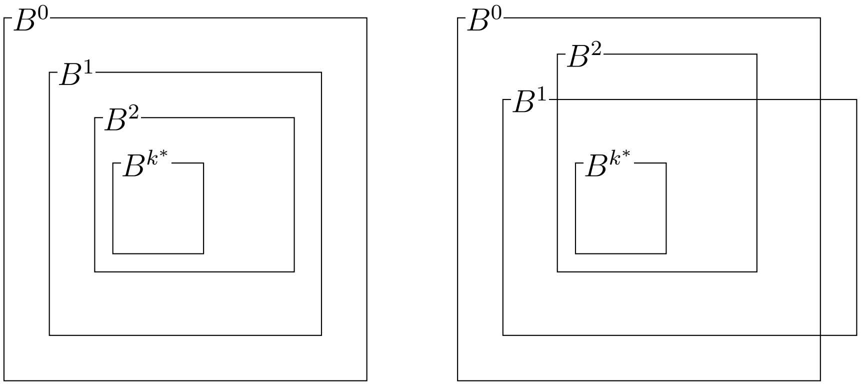


Figure 3.3: Highlighting the change to the definition of the ACO conditions. Whereas (SA1) required each set to be contained within the previous set (left), the new assumption (SA1)' does not require this (right).

be relaxed so that the space is unstructured. In particular assumptions (SA1) and (SU1) may be relaxed as follows:

$$\begin{array}{llcll} (SA1) & \forall k, x : B(k+1) \subseteq B(k) & \quad & (SU1) & \forall i : d_i \text{ is an ultrametric} \\ \Downarrow & & & \Downarrow & \\ (SA1)' & \forall x : x \in B(0) \Rightarrow F(x) \in B(0) & & (SU1)' & \forall i : d_i \text{ is a quasi-semi-metric} \end{array}$$

The definition of a quasi-semi-metric may be found in Appendix A. Figure 3.3 attempts to provide a geometrical interpretation of what this relaxation means in the context of the ACO conditions. The adjusted proofs of convergence for the static model will not be shown here, and interested readers may consult the Agda code [19]. However the proof that these relaxations are valid for the more general model of dynamic asynchronous iterations may be found in the second half of this chapter.

Weakening these assumptions has the immediate benefit of lessening the burden of proof on the user of these theorems. However a natural question to ask is that having removed the requirement of structure on the space, is it now possible to prove the convergence of new asynchronous processes for which it was not previously possible? Theorem 2 of [64] suggests that there are no finite examples, as it shows that when the set S is finite then the operator being an ACO is also a necessary condition for convergence. In the infinite case it still remains an open question. It should be noted there are plenty of interesting asynchronous iterations with infinite state spaces e.g. path-vector routing protocols.

A point of interest is that the formalisation of these proofs in Agda played an important role in the discovery of these relaxations. In a pen-and-paper proof it is very difficult to verify that certain assumptions have not been used implicitly. In contrast a proof assistant such as Agda can instantly list every use of an assumption. This is discussed further in Chapter 8.

It should also be noted that d_i, the distance functions in the AMCO conditions, are now a very strange set of functions indeed. They need obey neither the symmetry law nor any version of the triangle equality, and so are only quasi-semi-metrics. The quantity that such functions measure perhaps does not deserve the epithet “distance”, however, for lack of a better alternative, this thesis will continue to use the term.

3.2.4 Drawbacks of the static model

Even after incorporating the generalisations discussed in the previous section, there are two main drawbacks to the standard model of asynchronous iterations. The first is that the definition of a schedule assumes that the set of nodes is fixed (SS2) and that the links between nodes continue to function indefinitely (SS3). While these liveness assumptions may be reasonable when modelling a process being run on a multi-core computer, it is unrealistic in many decentralised “always on” processes such as routing. For example the global BGP system has been “on” since the early 1990’s. Since then countless links and nodes have failed/been removed and at the same time the system has grown from a few dozen routers to millions, all whilst the protocol continues to run.

It is possible to argue that a temporary (though not permanent) node failure can be represented in the existing model by simply excluding a node from the set of active nodes over the relevant period of time. However this is still unsatisfactory as many types of node failure will result in the node’s state being erased (e.g. after replacing a faulty server in a data centre). In reality after such an event the node is forced to revert back to the initial state. This “rebooting” of a node cannot be described by the existing model.

Another issue is that the model assumes that the operator F remains the same over time. This may not be the case if, for example, the function F depends on some process external to the computation (e.g. link latency in routing) or if it depends on the set of participants (e.g. resource allocation/consensus algorithms). This thesis therefore uses the term *static* to refer to the asynchronous iterations discussed so far and *dynamic* to refer to this new class of asynchronous iterations in which the set of participating nodes and functions may change over time.

In many works (including the author’s own work [18]) it is informally argued (or even implicitly assumed) that the correctness of dynamic iterations is an immediate consequence of the correctness of static iterations. The reasoning runs that a dynamic iteration is really a sequence of static iterations, where each new static iteration starts from the final state of the previous static iteration in the sequence. However this reasoning is incorrect as it does not take into account the messages from the previous iteration that will arrive during the new iteration. For example if node 1 goes down, it may still have messages in flight that node 2 will receive at some point in the future. However the static model does not allow messages to arrive from outside the set of nodes V.

The second half of this chapter therefore proposes a new, more general model that can describe both static and dynamic processes.

3.3 Dynamic asynchronous iterations

3.3.1 Model

Let V be the set of all the nodes that participate at some point during the iteration. It is still assumed that V is finite, despite the aim to model a process in which an arbitrary number of nodes may leave and join over time. The justification for this is that the only cases in which V is infinite are if either an infinite number of nodes were participating at the same time or an infinite amount of time has passed since the iteration began. Neither case is of interest in reality.

Similar to before, there exists a product state space $S = S_1 \times S_2 \times .. \times S_n$ where $n = |V|$. In order to capture the new dynamic nature of the process the concept of an *epoch* is introduced. An epoch is a contiguous period of time in which both the function being iterated and the set of participating nodes remain constant. The set of epochs is denoted as E but as with time can be assumed to be an alias for $\mathbb{N}$.

Instead of a single function F, it is now assumed that F is a family of indexed functions where F^{ep} is the function being computed in epoch e by participants $p \subseteq V$. Furthermore it is assumed there exists a special non-participating state $\perp \in S$.

A schedule must therefore not only track the activation of nodes and the flow of data between them but also the current epoch and the participants. Given these requirements it is natural to redefine a schedule as follows:

> **Definition 13.** A *dynamic schedule* is a tuple of functions $(\alpha, \beta, \eta, \pi)$ where:
>
> - $\alpha : T \to 2^V$ is the *activation function*, where $\alpha(t)$ is the set of nodes which activate at time t.
> - $\beta : T \times V \times V \to T$ is the *data flow function*, where $\beta(t, i, j)$ is the time at which the information used by node i at time t was sent by node j.
> - $\eta : T \to E$ is the *epoch function*, where $\eta(t)$ is the epoch at time t.
> - $\pi : E \to 2^V$ is the *participants function*, where $\pi(e)$ is the set of nodes participating in the computation during epoch e.
>
> such that:
>
> (DS1) $\forall i, j, t : \ \beta(t, i, j) < t$ – information only travels forward in time

(DS2) $\forall t_1, t_2 : t_1 \leq t_2 \Rightarrow \eta(t_1) \leq \eta(t_2)$ – the epoch number increases monotonically

Note that the two problematic assumptions of node liveness (SS2) and link liveness (SS3) have been dropped as they no longer make sense in the current form. For example, how can it be guaranteed that a node will continue to activate indefinitely if the possibility exists that it permanently ceases to participate? In their place is the assumption (DS2) that epochs are monotonically increasing. Although not technically required, the assumption is convenient as if there are two points in time in the same epoch then consequently every point between them must also be in the same epoch. This assumption does not reduce the expressive power of the model, as for any non-monotonic η it is possible to find a suitable relabelling of epochs that recovers monotonicity. Another possible assumption that might be made is that a node can only activate if it is participating in the iteration (i.e. $\forall t : \alpha(t) \subseteq \pi(\eta(t))$). However, although the assumption is reasonable, the dynamic asynchronous state function δ will be defined in such a way that it will not be required.

Some additional notation is also defined for $\rho(t)$, the set of nodes participating at time t, and F^t, the function being used at time t:

$$\begin{aligned} \rho(t) &\triangleq \pi(\eta(t)) \\ F^t &\triangleq F^{\eta(t)\rho(t)} \end{aligned}$$

It is now possible to redefine the asynchronous state function.

Definition 14. Given a schedule $(\alpha, \beta, \eta, \pi)$ the *dynamic asynchronous state function* is defined as follows:

$$\delta_i^t(x) = \begin{cases} \bot_i & \text{if } i \notin \rho(t) \\ x_i & \text{else if } t = 0 \text{ or } i \notin \rho(t-1) \\ \delta_i^{t-1}(x) & \text{else if } i \notin \alpha(t) \\ F_i^t(\delta_1^{\beta(t,i,1)}(x), \delta_2^{\beta(t,i,2)}(x), ..., \delta_n^{\beta(t,i,n)}(x)) & \text{otherwise} \end{cases}$$

where $\delta_i^t(x)$ is the state of node i at time t starting from state x.

If a node is not currently participating then it adopts its non-participating state. If it is participating at time t but was not participating at the time $t-1$ then it must have just (re)joined the computation and it therefore adopts its initial state. If the node is a continuing participant and is inactive at time t then its state remains unchanged. Otherwise, if it is active at time t, it updates its state in accordance with the data received from the other nodes in the computation.

Note that at time t nodes can use data from any node in V rather than just $\rho(t)$, the current set of participants. Hence nodes that are currently participating may end up processing messages from nodes that are no longer participating in the current epoch.

Also note that this new model is a strict generalisation of Üresin & Dubois's static model as the previous definition of δ is immediately recovered by setting $\eta(t) = 0$ and $\rho(t) = V$.

3.3.2 What does "correct" mean?

Unlike static iterations, the definition of correctness for dynamic iterations is not immediately obvious. Guaranteeing that such a process will always converge to a fixed point is impossible as both the underlying computation and the participants may continue to change indefinitely. Furthermore the epoch durations may be short enough that no fixed point is ever reached, even temporarily. The natural intuitive notion in such circumstances is to say that an iteration is *convergent* if whenever an epoch lasts "long enough" then δ will converge to a fixed point for the remainder of that epoch.

Therefore the next question is what does "long enough" mean? Unlike in a static schedule, the definition of a dynamic schedule does not constrain the activation or data flow functions. Consequently there is no guarantee that a node will activate even once during an epoch. One solution to this problem might be to enforce a restriction along the lines of "there exists an m such that every participating node must activate at least every m time steps". However this condition would exclude an infinite set of schedules for which convergence still occurs.

Instead this thesis turns its attention to some concepts that are used (implicitly rather than explicitly) by Üresin & Dubois in their proof of static convergence.

> **Definition 15.** A period of time $[t_1, t_2]$ is an *activation period* for node i if $\eta(t_1) = \eta(t_2)$ and there exists a time $t \in [t_1, t_2]$ such that $i \in \alpha(t)$.

Therefore an activation period is simply a contiguous period of time during an epoch in which the node is guaranteed to activate at least once.

> **Definition 16.** A period of time $[t_1, t_2]$ is an *expiry period* for node i if $\eta(t_1) = \eta(t_2)$ and for all nodes j and times $t \geq t_2$ then $t_1 \leq \beta(t, i, j)$.

After an expiry period the node is guaranteed to use only data generated after the start of the expiry period. In other words, all messages in flight to node i at time t_1 have either arrived or been lost by time t_2.

> **Definition 17.** A period of time $[t_1, t_2]$ is a *pseudocycle* if $\eta(t_1) = \eta(t_2)$ and for all nodes $i \in \rho(t_1)$ there exists a time $t \in [t_1, t_2]$ such that $[t_1, t]$ is an expiry period for node i and $[t, t_2]$ is an activation period for node i.

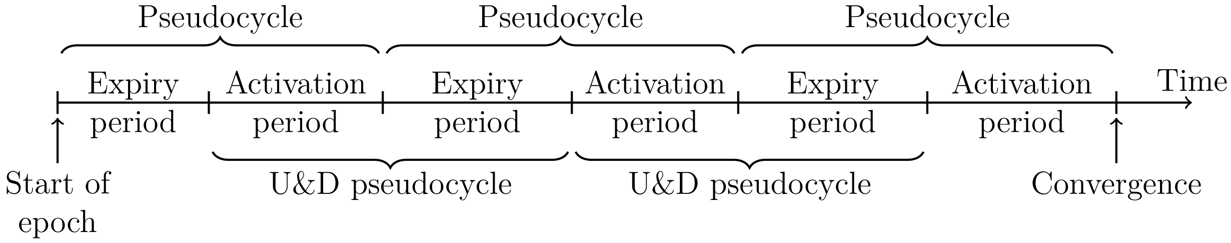


Figure 3.4: A sequence of pseudocycles comprised of alternating activation and expiry periods.

> **Definition 18.** A period of time $[t_1, t_k]$ *contains k pseudocycles* if there exists times t_i for $i \in [2, k-1]$ such that the time periods $[t_1, t_2]$, $[t_2, t_3]$, ..., $[t_{k-1}, t_k]$ are all pseudocycles.

The term “pseudocycle” refers to the fact that during such a period of time the asynchronous iteration will make at least as much progress as that of a single step of the synchronous iteration. This notion will be made formal later on by Lemma 9 in Section 3.3.4.

Note that the definition of a pseudocycle above differs from that of Üresin & Dubois as they define a pseudocycle the other way around, i.e. an activation period and then an expiry period. The reason for this change is that convergence occurs at the end of an activation period, as will be implied by Lemmas 5 & 9. Furthermore, unlike in the static model the dynamic model will require an extra expiry period at the front of the sequence to flush stale information that originated in previous epochs. Consequently, as illustrated in Figure 3.4, Üresin & Dubois's definition of pseudocycle is misaligned with both the start of the epoch and the time of convergence. Realigning the pseudocycles therefore simplifies both the definition and the proof of convergence.

Using these new definitions, correctness can now be defined as follows:

> **Definition 19.** A dynamic asynchronous iteration is *deterministically convergent* over a set $X \subseteq S$ if:
>
> 1. for every epoch e and set of participants p there exists a fixed point x^*_{ep} and number of iterations k^*_{ep}.
>
> 2. for every starting state $x \in X$, schedule and time t_1 then if the time period $[t_1, t_2]$ contains $k^*_{\eta(t_1)\rho(t_1)}$ pseudocycles then for every time t_3 such that $t_2 \geq t_3$ and $\eta(t_1) = \eta(t_3)$ then $\delta^{t_3}(x) = x^*_{\eta(t_1)\rho(t_1)}$.

The definition requires that there exists a fixed point x^*_{ep} for every epoch e and set of participants p. However it does not guarantee that the iteration will ever reach any of these fixed points, even temporarily. Instead it only guarantees that the fixed point will be reached if the current epoch contains at least k^*_{ep} pseudocycles.

Why is the criterion of the time period being a pseudocycle the correct one?

1. Consider the consequences of an epoch *not* containing a pseudocycle. It implies either a) one of the participating nodes never activated or b) one of the participating nodes was still using information from the previous epoch at the end of the current epoch. In either case, guaranteeing that δ will converge to the current fixed point is impossible.

2. In the degenerate case of a static computation this is a strictly weaker condition than the node and link liveness assumptions for static schedules. Node and link liveness imply that there are infinite number of pseudocycles (Proposition 1 in [64]), whereas this new notion of convergence requires only a finite number of pseudocycles. Therefore one could argue that in the static theory, the default assumptions of node and link liveness should be removed from the definition of the static schedule, and instead be relegated to sufficient conditions.

3. Any intuitive liveness constraints on the activation and data flow functions result in a "sufficiently long" time period being a pseudocycle. Some examples of such liveness constraints are:

    - There exists an m such that every participating node is guaranteed to activate at least once every m time steps, and there exists a q such that every pair of participating nodes succeeds in exchanging a message at least every q time steps.
    - There exists a final epoch in which every node and link continue to activate indefinitely.

3.3.3 Correctness conditions

Equipped with a suitable definition of correctness, this section now considers how to generalise the ACO and ACMO conditions. However before doing so, some additional definitions are needed. As would be expected, information from non-participating nodes that is still "in-flight" may interfere with the convergence of δ. Therefore some notion of a state only containing information for the current set of participants is needed.

> **Definition 20.** A state x is *accordant* with respect to a set of participants p if every inactive node is assigned the inactive state, i.e. $\forall i \notin p : x_i = \bot_i$. The set of states that are accordant with p is denoted as A_p.

It is also important to be careful about the notion of equality in the upcoming definitions. Due to the changing set of participants there are now two possible definitions for equality over states. Equality over the entire state:

$$x = y \triangleq \forall i : x_i = y_i$$

and equality with respect to p, the current subset of active nodes:

$$x =_p y \triangleq \forall i \in p : x_i = y_i$$

Clearly if states x and y are accordant then $x =_p y$ implies $x = y$.

The concept of a dynamic ACO is now defined, and while it might be tempting to simply require that every F^{ep} be a static ACO, there are a couple of additional constraints necessary.

> **Definition 21.** The set of functions F are a *dynamic ACO* if for every epoch e and set of participants p there exists a sequence of sets $B^{ep}(k) = B^{ep}(k)_1 \times B^{ep}(k)_2 \times ... \times B^{ep}(k)_n$ for $k \in \mathbb{N}$ such that:
>
> (DA1) $\forall e, p, x : x \in B^{ep}(0) \Rightarrow F^{ep}(x) \in B^{ep}(0)$
>
> (DA2) $\forall e, p, k, x : x \in A_p \wedge x \in B^{ep}(k) \Rightarrow F^{ep}(x) \in B^{ep}(k+1)$
>
> (DA3) $\forall e, p : \exists k^*_{ep}, x^*_{ep} : \forall k : k^*_{ep} \leq k \Rightarrow B^{ep}(k) = \{x^*_{ep}\}$
>
> (DA4) $\forall e, f, p, q, i : B^{ep}(0)_i = B^{fq}(0)_i$
>
> (DA5) $\forall e, p, k, i : i \notin p \Rightarrow \bot_i \in B^{ep}(k)_i$

Assumptions (DA1), (DA2) & (DA3) are generalised versions of (SA1)', (SA2) & (SA3) respectively. The only difference is that (DA2) has been weakened so that applying F only advances a box when the state is accordant with the current set of participants. This means that progress need not be made in the case when stale messages are still being received from nodes that are no longer participating. The new assumption (DA4) requires that all the initial boxes are equal and, when combined with (DA1), this ensures that the iteration is always in the initial box of the current epoch. Finally (DA5) enforces that the boxes respect the non-participating state.

The corresponding definition for the AMCO conditions is:

> **Definition 22.** The set of functions F are a *dynamic AMCO* if for every epoch e and set of participants $p \subseteq V$ there exists a distance function d^{ep}_i such that:
>
> (DU1) $\forall e, p, i : d^{ep}_i$ is quasi-semi-metric

> (DU2) $\forall e, p, i : d_i^{ep}$ is bounded
>
> (DU3) $\forall e, p$: F^{ep} is strictly contracting on orbits w.r.t. D^{ep} over A_p
>
> (DU4) $\forall e, p$: F^{ep} is strictly contracting on x^* w.r.t. D^{ep} over A_p for any fixed point x^*
>
> (DU5) $\forall e, p, x : F^{ep}(x) \in A_p$.
>
> where $D^{ep}(x, y) = \max_{i \in p} d_i^{ep}(x, y)$.

Again assumptions $(DU1) - (DU4)$ are generalisations of $(SU1)' - (SU4)$. The crucial difference is that everything is restricted to the set of participants: i) F^{ep} need only be strictly contracting over accordant states A_p, ii) the notion of equality used in (DU3) and (DU4) is $=_p$ rather than $=$, and iii) the distance function D is defined as the maximum over all participating states. Note that assumption (SU5) that S is non-empty is not needed as the dynamic model assumes the existence of the non-participating state $\bot \in S$. Instead the new assumption (DU5) ensures that the operator F respects the current set of participants. This assumption was not stated explicitly in the dynamic ACO conditions but can be derived from assumptions (DA2) and (DA5).

3.3.4 Dynamic ACO implies convergent

This section now proves that if F is a dynamic ACO then δ is convergent over $B^{0\{\}}(0)$. Unless stated otherwise, it assumes the existence of some arbitrary schedule $(\alpha, \beta, \eta, \pi)$ and starting state $x \in B^{0\{\}}(0)$. As with F^t, a shorthand $B^t \triangleq B^{\eta(t)\rho(t)}$ is used so that the boxes may be indexed by time rather than by epoch and participants. Initially some auxiliary definitions are introduced in order to improve the readability of the proof.

> **Definition 23.** *The state of node i is in box k at time t* if:
>
> $$\delta_i^t(x) \in B^t(k)_i$$

This simply formalises what is meant by the statement that node i's state is in box k.

> **Definition 24.** *The messages to node i are in box k at time t* if:
>
> $$\forall s : (s > t) \land (\eta(s) = \eta(t)) \Rightarrow \forall j : \delta_j^{\beta(s,i,j)}(x) \in B^t(k)_j$$

This ensures that any message arriving at node i after time t and before the end of the current epoch is guaranteed to be in box k. An alternative way of viewing this condition is that node i's *local* view of the iteration's state is (and will always be) in box k. This is the first of the two preconditions necessary for (DA2) that will be needed by Lemma 7 to prove that node i's state permanently advances a box.

> **Definition 25.** *The messages to node i are well formed at time t* if:
>
> $$\forall s : (s > t) \wedge (\eta(s) = \eta(t)) \Rightarrow \forall j : j \notin \rho(s) \Rightarrow \delta_j^{\beta(s,i,j)} = \bot_j$$

This ensures that any message arriving at node i after time t from a non-participating node j will always contain the non-participating state $\bot_j$. This is equivalent to stating that node i's local view of the state is accordant with the current of participants. This is the second of the two preconditions necessary for (DA2) that will be needed by Lemma 7 to prove that node i's state permanently advances a box.

> **Definition 26.** *The computation at node i is in box k at time t* if:
>
> 1. the state of node i is in box k at time t.
> 2. the messages to node i are in box $k - 1$ at time t.
> 3. the messages to node i are well formed at time t.

This definition collects together the pre-conditions required to prove that the state of node i will always be in box k for the remainder of the epoch (see Lemma 5). Finally this definition is lifted to the whole computation as follows:

> **Definition 27.** *The computation is in box k at time t* if for all nodes $i \in \rho(t)$ then the computation at node i is in box k at time t.

Note that the above definition does not place any requirements on non-participating nodes. This is because by the definition of δ any non-participating node i is always in the non-participating state $\bot_i$, which, by assumption (DA5), is in every one of the boxes, including the final one.

The proof is now split into four parts. The first set of closure lemmas prove that the state and the messages remain in box 0 even across epoch boundaries. The second set of stability lemmas describe how after the computation reaches box k it remains in box k for the remainder of the epoch. The third set of progress lemmas demonstrate how during a pseudocycle the entire computation advances at least one box. Finally these three sets of lemmas are combined to prove convergence.

3.3.4.1 Closure lemmas

Assumption (DA4) states that the initial boxes are all equal no matter which epoch or set of participants they are parametrised by. In order to later apply the other ACO assumptions, it is first necessary to establish that this initial box is closed over δ, i.e. that the iteration never escapes the initial box. The following lemmas therefore prove that both the state and the messages are always in the initial box of the current epoch.

Lemma 3. For every time t and node i then the state of node i is in box 0 at time t.

Proof. Consider an arbitrary time t and node i. The proof that $\delta_i^t(x) \in B^t(0)_i$ proceeds by induction over the definition of δ.
Case 1: $i \notin \rho(t)$
Then $\delta_i^t(x) = \bot_i$ and $\bot_i \in B^t(0)_i$ by assumption (DA5).
Case 2: $i \in \rho(t)$ and ($t = 0$ or $i \notin \rho(t-1)$)
Then $\delta_i^t(x) = x_i$ and $x_i \in B^0(0)_i$ by the initial assumption. Hence $\delta_i^t(x)$ is also in box $B^t(0)_i$ by assumption (DA4) which states that $B^0(0)_i = B^t(0)_i$.
Case 3: $i \in \rho(t)$ and $i \in \rho(t-1)$ and $i \notin \alpha(t_2)$
Then $\delta_i^t(x) = \delta_i^{t-1}(x)$, and $\delta_i^{t-1}(x) \in B^{t-1}(0)_i$ by the inductive hypothesis applied to time $t-1$. Hence $\delta_i^t(x) \in B^t(0)_i$ by assumption (DA4) which states that $B^{t-1}(0)_i = B^t(0)_i$.
Case 4: $i \in \rho(t)$ and $i \in \rho(t-1)$ and $i \in \alpha(t)$
Then $\delta_i^t(x) = F_i^t(\delta_1^{\beta(t,i,1)}(x), ..., \delta_n^{\beta(t,i,n)}(x))$. For each j then $\delta_j^{\beta(t,i,j)}(x) \in B^{\beta(t,i,j)}(0)_j$ by the inductive hypothesis applied to time $\beta(t,i,j)$. Furthermore $B^{\beta(t,i,j)}(0)_j = B^t(0)_j$ by assumption (DA4). Hence $F_i^t(...) \in B^t(0)_i$ by assumption (DA1) which states that $B^t(0)$ is closed under F^t. □

Lemma 4. For every time t and node i then the messages to node i are in box 0 at time t.

Proof. Consider an arbitrary time t and node i. Then for all times $s \geq t$ and nodes j it must be shown that $\delta_j^{\beta(t,i,j)}(x) \in B^t(0)_j$. This immediately follows as $\delta_j^{\beta(t,i,j)}(x) \in B^{\beta(t,i,j)}(0)_j$ by Lemma 3 and $B^{\beta(t,i,j)}(0)_j = B^t(0)_j$ by assumption (DA4). □

3.3.4.2 Stability lemmas

Guaranteeing progress towards the fixed point in asynchronous iterations is complicated by the fact that old and out-of-date messages from earlier in the epoch may arrive and undo recent progress. The next series of lemmas examine what conditions are necessary to guarantee that once the state and messages are in box k then they will always be in box k.

Note that from here on, the proof will only reason about the behaviour of δ within a single epoch. Therefore going forwards every moment of time referenced by the proof is assumed to belong to the same epoch e, unless explicitly stated otherwise. The dependency of F and B on the time t are therefore dropped for notational clarity.

Lemma 5. If the computation at node i is in box k at time t then the state of node i is in box k for every time $s \geq t$.

Proof. Assume that the computation at node i is box k at time t for an arbitrary node i and time t. It is then necessary to show that $\delta_i^s(x) \in B_i(k)$ for any $s \geq t$. The proof proceeds by induction over time s and the definition of δ. If $s = t$ then the state of node i is in box k at time t by the definition of the computation at node i being in box k at time t. Otherwise if $s > t$ then consider the following cases:
Case 1: $i \notin \rho(s)$
Then $\delta_i^s(x) = \bot_i$ and $\bot_i \in B(k)_i$ by assumption (DA5).
Case 2: $i \in \rho(s)$ and $i \notin \rho(s-1)$
As $s-1$ is in the same epoch as s then $\rho(s-1) = \rho(s)$, contradicting the case assumptions.
Case 3: $i \in \rho(s)$ and $i \in \rho(s-1)$ and $i \notin \alpha(s)$
Then $\delta_i^s(x) = \delta_i^{s-1}(x)$ and $\delta_i^{s-1}(x) \in B(k)_i$ by the inductive hypothesis at time $s-1$.
Case 4: $i \in \rho(s)$ and $i \in \rho(s-1)$ and $i \in \alpha(s)$
Then $\delta_i^s(x) = F_i(\delta_1^{\beta(s,i,1)}(x), ..., \delta_n^{\beta(s,i,n)}(x))$. The arguments to $F_i(...)$, (i.e. the latest messages to arrive at node i from each node j) are all well formed and in $B(k-1)_j$, thanks to the assumption that the computation at node i is in box k at time t. Hence $F_i(...) \in B(k)_i$ by assumption (DA2). □

Lemma 6. If messages to node i are in box k at time t then the messages to node i are in box k for all times $s \geq t$.

Proof. This is trivial by the definition of the messages to node i being in box k at time t. □

3.3.4.3 Progress lemmas

Having established that i) the iteration is always in the initial box no matter the epoch and ii) once the computation at node i has reached box k, it remains in box k, it is next necessary to prove when the computation advances a box during an epoch. These conditions are intimately tied to the notion of a pseudocycle.

Lemma 7. If the messages to node i are well-formed and are in box k at time t and $[t, s]$ is an activation period then the state of node i is in box $k+1$ at time s.

Proof. The proof that $\delta_i^s(x) \in B(k+1)_i$ proceeds by induction over the definition of δ and time s. As activation periods are of non-zero length then $s > t$ and so consider the following cases:
Case 1: $i \notin \rho(s)$
Then $\delta_i^s(x) = \bot_i$ and $\bot_i \in B(k+1)_i$ by assumption (DA5).
Case 2: $i \in \rho(s)$ and $i \notin \rho(s-1)$
As $s-1$ is in the same epoch as s then $\rho(s-1) = \rho(s)$, contradicting the case assumptions.
Case 3: $i \in \rho(s)$ and $i \in \rho(s-1)$ and $i \notin \alpha(s)$
Then $\delta_i^s(x) = \delta_i^{s-1}(x)$. If $s = t+1$ then the initial assumptions are contradicted as i has

not activated during the period $[t, s]$. Therefore $s > t + 1$ and hence $\delta_i^{s-1}(x) \in B(k+1)_i$ by applying the inductive hypothesis at time $s - 1$.

Case 4: $i \in \rho(s)$ and $i \in \rho(s-1)$ and $i \in \alpha(s)$

Then $\delta_i^s(x) = F_i(\delta_1^{\beta(s,i,1)}(x), ..., \delta_n^{\beta(s,i,n)}(x))$. By the assumptions that all the messages to node i were well formed and in box k at time t then node i's local view of the state at time s is accordant and in box k. Hence $F_i(...) \in B(k+1)_i$ by assumption (DA2). □

Lemma 8. If the computation is in box k at time t and $[t, s]$ is an expiry period for node i then the messages to node i are in box k at time s.

Proof. Assume that the computation is in box k at time t and consider two arbitrary nodes i and j. It is necessary to show that for all times $r \geq s$ then $\delta_j^{\beta(r,i,j)}(x) \in B(k)_j$. As $[t, s]$ is an expiry period then $t \leq \beta(r, i, j)$ and therefore $\beta(r, i, j) \in [t, r]$. If j is not participating then $\delta_j^{\beta(r,i,j)}(x) = \bot_j$ and $\bot_j \in B(k)_j$ by assumption (DA5). Otherwise if j is participating then the required result follows by Lemma 5 applied to times t and r and the fact that the computation at node j is in box k at time t. □

Lemmas 7 & 8 have shown that during activation and expiry periods the state and the messages are guaranteed to advance at least one box respectively. The next lemma combines them to prove that during a pseudocycle the whole computation progresses at least one box, i.e. during a pseudocycle the asynchronous iteration makes at least as much progress as a single step of the synchronous iteration.

Lemma 9. If the computation is in box k at time t and the period $[t, s]$ is a pseudocycle then the computation is in box $k + 1$ at time s.

Proof. As $[t, s]$ is a pseudocycle then for each participating node $i \in \rho(t)$ there exists a time m such that $[t, m]$ is an expiry period and $[m, s]$ is an activation period.

- As the messages to node i are well-formed at time t then they are also well-formed at times m and s.

- As $[t, m]$ is an expiry period and the computation is in box k at time t, then the messages to node i are in box k at time m by Lemma 8, and also therefore at time s by Lemma 6.

- As $[m, s]$ is an activation period and the messages to node i are well-formed and in box k at time m (by the previous two points) then the state of node i in box $k + 1$ at time s by Lemma 7.

Consequently all three requirements for the computation at node i being in box $k + 1$ at time s are fulfilled. □

3.3.4.4 Convergence

Now that Lemma 9 has established that during a pseudocycle the whole computation advances one box, one might think this can be repeatedly applied to prove convergence immediately. Unfortunately however, the base case is still missing, as although Lemmas 3 & 4 proved that the state and the messages are always in box 0, the computation as a whole is *not* necessarily in box 0. This is because the messages are not necessarily well-founded, i.e. there may still be messages in flight from non-participating nodes. The following lemma therefore establishes a base case by proving that after one pseudocycle the computation is in box 1 (the trivial case when $k^* = 0$ will be dealt with separately in Theorem 4).

Lemma 10. If $[t, s]$ is a pseudocycle then the computation is in box 1 at time s.

Proof. As $[t, s]$ is a pseudocycle then for each participating node $i \in \rho(t)$ there exists a time m such that $[t, m]$ is an expiry period and $[m, s]$ is an activation period.

- As $[t, m]$ is an expiry period then all messages arriving at node i from node j after time m must be from the current epoch and hence the messages to node i are well-formed at times m and s.

- The messages to node i are in box 0 at times m and s by Lemma 4.

- As $[m, s]$ is an activation period and the messages to node i are well-formed and in box k at time m (by the previous two points) then the state of node i in box 1 at time s by Lemma 7.

Consequently all three requirements for the computation at node i being in box 1 at time s are fulfilled. □

Finally the main theorem may now be proved.

Theorem 4. If F is a dynamic ACO then δ is deterministically convergent over $B^{0\{\}}(0)$.

Proof. To prove that δ is convergent it is first necessary to construct a fixed point x^*_{ep} and iteration number k^*_{ep} for every epoch e and set of participants p. Let these be the x^*_{ep} and k^*_{ep} respectively as specified by assumption (DA3).

Next consider an arbitrary schedule, starting state $x \in B^{0\{\}}(0)$ and starting time t_1 in epoch $e = \eta(t_1)$ with participants $p = \rho(t_1)$. It is now necessary to show that if $[t_1, t_2]$ contains k^*_{ep} pseudocycles then δ will have converged to x^*_{ep} by time t_2. If $k^*_{ep} = 0$ then it is only necessary to show that the state is always in the initial box which holds trivially by Lemma 3. Otherwise if $k^*_{ep} \neq 0$ then after the first pseudocycle the computation is in box 1 by Lemma 10. Consequently after the remaining $k^*_{ep} - 1$ pseudocycles, the computation is in box k^*_{ep} at time t_2 by repeated application of Lemma 9. Hence for any subsequent time t_3 in epoch e, then $\delta^{t_3}(x) \in B^{ep}(k^*_{ep})$ by Lemma 5 and, as x^*_{ep} is the only state in $B^{ep}(k^*_{ep})$ by assumption (DA3), then $\delta^{t_3}(x) = x^*_{ep}$. □

3.3.5 Dynamic AMCO implies convergent

Although the dynamic ACO conditions are sufficient to guarantee convergence, they can be a tricky to construct in practice as discussed previously in Section 3.2.2. As will be seen in Chapter 5, the dynamic AMCO conditions are often easier to work with. This section proves that the dynamic AMCO conditions also guarantee convergence by constructing a reduction from the dynamic AMCO conditions to the dynamic ACO conditions. The main thrust of the reduction is relatively simple. As F^{ep} is strictly contracting on orbits & its fixed points, it possesses a unique fixed point x^*_{ep}. As all distances are bounded above by $d^{ep}_{\max}$, the box $B^{ep}(k)_i$ can then be defined as the set of the states which are at a distance of no more than $d^{ep}_{\max} - k$ from $(x^*_{ep})_i$, the i^{th} component of the fixed point.

Theorem 5. If function F is a dynamic AMCO then F is a dynamic ACO.

Proof. Consider an arbitrary epoch e and set of participants p. For notational clarity the superscripts e and p will be dropped from F, d_i, d_{max}, D and B unless where explicitly mentioned otherwise.

Using $\bot$ as the accordant starting state, and as F is strictly contracting on orbits (DU3) and on fixed points (DU4), then by Lemmas 1 & 2 there exists a unique fixed point x^* such that $F(x^*) =_p x^*$. This fixed point is necessarily accordant with p by (DU5) and its method of construction, and hence the equality can be strengthened to $F(x^*) = x^*$.

The i^{th} component for the k^{th} box is then be defined as follows:

$$B(0)_i \triangleq S_i$$

$$B(k+1)_i \triangleq \begin{cases} \{\bot_i\} & \text{if } i \notin p \\ \{x_i \mid d_i(x_i, x^*_i) \leq d_{max} - k\} & \text{if } i \in p \end{cases}$$

It is now necessary to verify that the boxes B fulfil the required conditions:

1. (DA1) – $\forall x : x \in B(0) \Rightarrow F(x) \in B(0)$

   Immediate from the definition of $B(0)$.

2. (DA2) – $\forall k, x : x \in A_p \wedge x \in B(k) \Rightarrow F(x) \in B(k+1)$

   Consider an accordant state $x \in B(k)$ and an arbitrary node i. If $i \notin p$ then $x_i = \bot_i$ by $x_i \in B(k)_i$, and hence x_i is also in $B(k+1)_i$. Otherwise if $i \in p$ it remains to show that $F(x)_i \in B(k+1)_i$ or equivalently that $d_i(x^*_i, F(x)_i) \leq d_{max} - (k+1)$. To see why this inequality holds consider whether or not $x =_p x^*$.

If $x =_p x^*$ then the inequality follows directly:

$$\begin{aligned} d_i(x^*_i, F(x)_i) &= d_i(x^*_i, F(x^*)_i) && \text{(as } x =_p x^*\text{)} \\ &= d_i(x^*_i, x^*_i) && \text{(as } F(x^*) = x^*\text{)} \\ &= 0 && \text{(by (DU1))} \\ &\leq d_{max} - (k+1) \end{aligned}$$

Otherwise if $x \neq_p x^*$ then $d_i(x^*_i, F(x)_i) < d_{max} - k$ as:

$$\begin{aligned} d_i(x^*_i, F(x)_i) &\leq D(x^*, F(x)) && \text{(by definition of } D\text{)} \\ &< D(x^*, x) && \text{(by (DU4) \& (DU5))} \\ &\leq d_{max} - k && \text{(as } x \in B(k)\text{)} \end{aligned}$$

which implies that $d_i(x^*_i, F(x)_i) \leq d_{max} - (k+1)$.

Hence in either case $F(x)_i \in B(k+1)_i$.

3. (DA3) – $\exists k^*, x^* : \forall k : k^* \leq k \Rightarrow B(k) = \{x^*\}$

   Let $k^* = d_{max}$ and x^* be the fixed point found above and consider a $k \geq d_{max}$. If $k = 0$ then $d_{max} = 0$ and so the result holds trivially as all points are equal by (SU2) & (DU1). If $k > 0$ then suppose there exists a state $x \in B(k)$. It must therefore be shown that $x = x^*$. If $i \notin p$ then $x_i \in B(k)_i$ implies $x_i = \bot_i$ and $\bot_i = x^*_i$ as x^* is accordant. Otherwise if $i \in p$ then $x_i \in B(k)_i$ implies $d(x_i, x^*_i) \leq d_{max} - k$ and as $k \geq d_{max}$ then $d(x_i, x^*_i) = 0$ and hence $x_i = x^*_i$ by (DU1).

4. (DA4) – $\forall e, f, p, q, i : B^{ep}(0)_i = B^{fq}(0)_i$

   Immediate from the definition of $B(0)$.

5. (DA5) – $\forall k, i : i \notin p \Rightarrow \bot_i \in B(k)_i$

   If $k = 0$ or $k \neq 0$ then either way immediate from the definition of B.

Hence the conditions are satisfied and F is a dynamic ACO. □

Theorem 6. If F satisfies the dynamic AMCO conditions then δ is convergent.

Proof. As F is a dynamic AMCO then F is a dynamic ACO by Theorem 4. Hence δ is convergent by Theorem 5. □

3.4 Conclusions

The purpose of this chapter has been to construct a mathematical theory for dynamic asynchronous iterations that will form the basis of a new model for vector-based protocols described in Chapter 4. Along the way it has made several contributions to the existing theory.

Firstly it has relaxed the assumption that all asynchronous iterations that converge must occur in spaces with nested structure. Secondly it has described a new more general model for dynamic asynchronous iterations in which both the computation and the set of participants may change over time. It has then generalised the existing ACO and AMCO conditions for the static model and shown that the generalisations are sufficient to guarantee the correctness of the dynamic model.

It is still an open question whether the dynamic conditions are now truly as general as possible and therefore necessary as well as sufficient for the convergence of an asynchronous iteration. Furthermore it would be interesting to find a real-world convergent asynchronous iteration whose space does not have a nested structure.

Chapter 4

An algebraic model for vector-based protocols

The philosophy of the algebraic approach to routing is that routing problems can and should be specified separately from the algorithm used to solve them. Consequently classes of routing problems with similar properties can be identified, and the behaviour of routing algorithms may be reasoned about with respect to a class of routing problems rather than a single, specific instance.

The first half of this chapter discusses the two algebraic structures that have historically been used to model routing problems. Next it proposes a new structure that is both simpler and more expressive than the previous state-of-the-art. In particular this new structure is capable of modelling path-vector operations as part of the routing problem rather than the routing algorithm, thereby unifying distance-vector and path-vector protocols. The new structure can also model complex path-dependent conditional policy, such as route filtering in BGP.

The second half of the chapter uses this new algebraic structure to construct a model for a vector-based routing protocol over an abstract routing problem. Initially it defines a single synchronous iteration, and subsequently it applies the work in Chapter 3 to construct a fully asynchronous model. Finally the chapter provides an overview of the existing results and open questions in the field of algebraic routing.

Once again Appendix A contains some basic definitions that may be of use to the reader.

4.1 Routing problems as algebras

Before trying to formalise the notion of an abstract routing problem, it is first necessary to identify the core components that all routing problems have in common. For example every routing problem has a method of choosing between two alternative path weights: in the shortest-path problem the path with the smaller length is chosen, in the widest-path problem the path with the greater bandwidth. The corresponding choice procedures for RIP and BGP are described in Section 2.3.2. In total there are five such routing "primitives":

1. A set of weights assigned to paths.
2. A method for choosing between two alternative path weights.
3. A method for calculating the new weight when extending a path along a new edge.
4. A path weight representing the absence of a valid route.
5. A path weight representing the trivial route from a node to itself.

The following sections now discuss three models for routing problems: semiring algebras which held sway up until the early 2000s, a more general algebraic structure first proposed by Sobrinho in 2005 and finally a new algebra which is both simpler and more expressive than that of Sobrinho.

4.1.1 Semiring algebras

Shortly after the publication of the first deterministic algorithms for solving the shortest-paths problem, it was noticed that the algorithms only required minor modifications in order to solve many other interesting best path problems. Upon further investigation, it became clear that many common routing problems such as shortest-paths, widest-paths, most-reliable-paths etc. all formed instances of algebraic structures called *semirings* [25]. The first proposed model for routing problems was therefore a (commutative) semiring [13, 26] $(S, \oplus, \otimes, \overline{\infty}, \overline{0})$[1] where:

- S is the set of path weights.
- $\oplus : S \times S \rightarrow S$ is the choice operator, which given two path weights returns the preferred path weight.
- $\otimes : S \times S \rightarrow S$ is the extension operator, which given two path weights returns the weight of the concatenation of the two paths.
- $\overline{\infty}$ is the weight of the invalid path, i.e. a value representing no route.
- $\overline{0}$ is the weight of the trivial path, i.e. the route from a node to itself.

[1] Typically the semiring literature uses 0 and 1 instead of $\overline{\infty}$ and $\overline{0}$ respectively. However there was push-back from the networking community against the choice of 0 to represent the worst possible route, and 1 to represent the best possible route. The overline is used to differentiate them from elements of $\mathbb{N}^{\infty}$.

which obey the following axioms:

1. $\oplus$ is associative and commutative

   i.e. both the order in which choices are made and the order of the path weights in the choice are irrelevant. For example, for any path weights x, y and z choosing between x and y and then choosing between the result and z is equivalent to choosing between x and z and then choosing between the result and y.

2. $\otimes$ is associative

   i.e. the order in which the weight of a path is calculated is irrelevant. For example consider a path $ABCD$: starting at A and then extending the path to B, then to C and then to D is equivalent to starting at B, extending the path forwards to C, and then extending it backwards to A and then extending it forwards again to D.

3. $\overline{\infty}$ is an identity for $\oplus$ and an annihilator for $\otimes$

   i.e. any other route is preferred over the invalid route, and extending the invalid route always results in the invalid route.

4. $\overline{0}$ is an annihilator for $\oplus$

   i.e. the trivial route is always preferred over any other route.

5. $\otimes$ distributes over $\oplus$

   This assumption is more nuanced than the others. Essentially it says that all nodes in the network agree on which are the best paths through the network. This assumption will be discussed in greater depth in Section 4.5.

It is important to note that not all semirings have a natural interpretation as a routing problem. Indeed routing is just one of many fields in which semirings have been used. However amongst the semirings that do have a routing interpretation, many also have the additional property that $\oplus$ is selective:

$$\forall x, y \in S : x \oplus y \in \{x, y\}$$

This guarantees that when choosing between two path weights, one of the original path weights will be returned, and hence that $\oplus$ really does represent a choice. Routing problems with non-selective $\oplus$ operators will be discussed in Section 4.1.4.

As discussed in Appendix A, any $\oplus$ that is associative, commutative and selective induces/is induced by a total ordering $\leq$ over routes where $x \leq y \Leftrightarrow x \oplus y = x$, i.e. route x is less than or equal to route y if x is chosen over y. Correspondingly, assumptions (R2) & (R3) imply that $\overline{0}$ and $\overline{\infty}$ are the minimum and maximum elements of the order respectively.

As will be seen in the subsequent sections, the ordering $\leq$ is often used interchangeably with $\oplus$.

Despite being a good model for many basic routing problems, semirings are not general enough to capture all routing problems of interest. In particular:

$\otimes$ is not expressive enough While the $\otimes$ operator is capable of expressing simpler extension operations, it cannot model more complex operations. This is because the model labels links with path weights and conditional policies such as "reject any path with length greater than 4, otherwise add 1 to the length" have no natural representation as path-weights. Fundamentally the issue is that edges should be labelled with methods for transforming path weights rather than path weights themselves.

$\otimes$ is associative Closely linked to the previous point is that requiring the extension operator to be associative is both unnecessary and counter-intuitive. In practice paths are traversed from start to finish, and therefore there is no physical interpretation for calculating the weight of a path using an out-of-order traversal.

$\otimes$ distributes over $\oplus$ Finally the requirement that $\otimes$ distributes over $\oplus$ does not capture a fundamental aspect of routing. For example the shortest-widest path problem (where routes are first compared using their bandwidth and then by their length) is not distributive. Instead distributivity is simply one of several possible relationships that may exist between the choice and the extension operators. This is discussed further in Section 4.5.

4.1.2 Sobrinho algebras

Despite the problems outlined above, semirings remained the dominant model for routing and other related path problems for the next 30 years [3, 10, 44, 46, 48, 60]. During this time networks underwent a seismic shift in both scale and complexity. It was only after the wide-scale deployment of BGP that it was realised that some modern routing protocols could not be described within the existing semiring framework. In the case of BGP, amongst other issues, its underlying algebra is not distributive and its complex conditional policies cannot be adequately represented by the $\otimes$ operator. To address these problems, Sobrinho [62] proposed a new set of algebraic routing primitives $(S, L, \Sigma, \leq, \triangleright, \overline{\infty}, \Sigma_{\overline{0}}, g)$[2] where:

- S is the set of path weights.

[2] Notation has been changed to make the parallels with other models clearer.

- L is the set of edge labels.

  This set, distinct from the set of path weights, represents the set of policies that may be associated with edges in the network.

- Σ is the set of signatures.

  Sobrinho does not provide a justification for the inclusion of this set, nor a physical interpretation of what part of a protocol it represents (see the discussion on "Unnecessary complexity" at the end of this section). The best interpretation available is that these can be thought of as an intermediate language for describing path weights.

- $\leq$ is an ordering over S.

  As discussed in the previous section and in Appendix A, this is simply an alternative formulation of the choice operator $\oplus$ in the semiring model.

- $\triangleright : L \times \Sigma \rightarrow \Sigma$ is the extension operation.

  Given an edge labelled with l and a path with a signature x then $l \triangleright x$ is the signature of the path extended by the edge. This replaces the $\otimes$ operator in the semiring model.

- $\overline{\infty} \in \Sigma$ is the signature of the invalid route.

- $\Sigma_{\overline{0}} \subseteq \Sigma$ is the set of trivial signatures, one for each node in the network.

  Note that this component is not explicitly included by Sobrinho in the definition of the algebra. However their existence is later assumed (referred to as $s(u)$ in the paper for some node u).

- $g : \Sigma \rightarrow S$ is a function mapping signatures to path weights.

which obey the following axioms:

1. $\leq$ is a total order

   Equivalent to $\oplus$ being associative, commutative and selective in the semiring model.

2. $g(\overline{\infty})$ is the maximal element for $\leq$ and $\forall l \in L : l \triangleright \overline{\infty} = \overline{\infty}$

   Equivalent to the invalid route $\overline{\infty}$ being an identity for $\oplus$ and an annihilator for $\otimes$ in the semiring model.

3. For every trivial $s \in \Sigma_{\overline{0}}$ then $g(s)$ is preferred over any other non-trivial path weight.

   Equivalent to the trivial route $\overline{0}$ being an annihilator for $\oplus$ in the semiring model. Note that this assumption is implicit in the design of the algorithm rather than being stated explicitly in the algebra.

By associating edges with labels rather than path weights, the algebra is capable of capturing a much wider range of possible extension operations. For example, when modelling BGP, the set L could be thought of as the set of valid BGP routing policies. The expression $l \triangleright x$ would therefore represent the result of applying policy l to some incoming route x. The resulting asymmetry of the extension operator, $\triangleright$, means that the concept of it being associative no longer makes sense. Additionally the assumption of distributivity has been removed. In the paper Sobrinho goes on to prove that distributivity is not a fundamental property of routing problems, but instead is just one of several possible optional properties, each of which affect the behaviour of the routing algorithms in different ways. This is discussed in greater detail in Section 4.5.

However there still remain several issues with Sobrinho algebras as a model for routing problems:

Path operations The edge labels have no dependency on the nodes that they link, and hence it is impossible to formulate an assumption within the model that the extension operator $\triangleright$ correctly applies the operations associated with path-vector protocols. Suppose there was a set of labels that correctly performed this operation. For example l might be a label on a link from node 2 to node 3 that adds node 3 to the path stored within the path weight when extending a route from node 2. However there would be no way of preventing the label l from being assigned to a different link. As will be discussed in Section 4.3, Sobrinho worked around this limitation by making the path operations part of the algorithm rather than the algebra.

Conditional policy on paths Unfortunately the decision to make paths part of the algorithm has further undesirable consequences. In particular the extension operator can no longer inspect and make decisions according to the path associated with a route. For example the policy “reject a route if it goes through AS 17" is not directly expressible within the algebra. Such policies are regularly used in BGP.

Unnecessary complexity Finally, distinguishing between the set of signatures and the set of path weights does not add any expressive power to the model. Therefore signatures could theoretically be removed from the model.

4.1.3 Routing algebras

To address the problems listed above, this thesis proposes a third more general algebraic structure which will be referred to as a *routing algebra*. The resulting algebra is much closer to the original semiring model than the Sobrinho algebra. The only difference between it and semiring algebras is how it models the extension operation. In this new

Component	Semiring algebras	Sobrinho algebras	Routing algebras
Path weights	S	S, Σ, g	S
Choice	$\oplus$	$\leq$	$\oplus$
Extension	$\otimes$	L, $\triangleright$	E
Invalid route	$\overline{\infty}$	$\overline{\infty}$	$\overline{\infty}$
Trivial route	$\overline{0}$	$\Sigma_{\overline{0}}$	$\overline{0}$

Table 4.1: A comparison of the three models for routing problems.

structure edges are labelled with functions rather than labels or path weights. Extending a route is therefore as simple as applying the function to the path weight. For instance the weight of the result of extending a path of weight x along a link labelled with f is simply $f(x)$. While it might be tempting to have a single set of functions for labelling edges, in practice the model assigns each hypothetical link its own set of allowable functions. See Section 4.3 for an explanation for how this allows it to model path-vector protocol-like operations.

Definition 28. A *routing algebra* is a tuple $(S,\ \oplus,\ E,\ \overline{\infty},\ \overline{0})$ where:

- S is the set of path weights.
- $\oplus : S \times S \to S$ is the choice operator.
- $E : \mathbb{N} \times \mathbb{N} \to 2^{S \to S}$ is the set of edge function sets indexed by a source and destination node.
- $\overline{\infty} \in S$ is the invalid path weight.
- $\overline{0} \in S$ is the trivial path weight.

which obey the following axioms:

(R1) $\oplus$ is associative, commutative and selective.

(R2) $\overline{0}$ is an annihilator for $\oplus$.

(R3) $\overline{\infty}$ is an identity for $\oplus$.

(R4) For all nodes i and j and functions $f \in E_{ij}$ then $\overline{\infty}$ is a fixed point for f.

Assumptions (R1–R4) are all copied from the semiring algebras, where (R4) (which states that the extension of an invalid route is always the invalid route) has necessarily been adjusted to take into account that the extension operation is now modelled by functions.

Table 4.1 compares the three different algebraic models. It should be noted that there is no fundamental reason why it would not be possible to initially define choice in terms of $\leq$ rather than $\oplus$ as Sobrinho does. However the advantage of $\oplus$ is that it has a clear computational interpretation as the actual decision making procedure that occurs in routing protocols. It is hoped that this will make the model more intuitive for future protocol designers. When reasoning about routing problems, in some situations it is more convenient to use $\oplus$ and in others $\leq$ and so from here on the thesis will use both interchangeably.

4.1.4 Other algebras

It is argued that the above definition of a routing algebra captures the essence of a routing problem. Nonetheless there have been attempts to weaken some of the assumptions. In particular, semirings without the selectivity assumption have been studied extensively.

Gurney [34] proposed a non-selective extension to the semiring model in order to represent multi-path rather than single-path routing. To do this he used *minset algebras* where $\oplus$ and $\otimes$ operate over sets of routes instead of individual routes. In such a case the assumption of selectivity is relaxed to idempotence. In a similar vein Mohri [52] used even weaker non-idempotent algebras to study the k-shortest paths problem.

These multi-path generalisations are in some sense orthogonal to the core problems of single-path routing. It seems implausible that the multi-path version of a routing problem has better convergence behaviour than the original single-path version. Given that there exist many unanswered questions about single-path routing, this thesis will leave such generalisations to future work.

4.2 Vector-based routing

The previous section has described how routing algebras can be used to represent routing problems. This section now combines this representation with the theory of asynchronous iterative computations developed in Chapter 3 to construct a model for an abstract vector-based routing protocol. This model is significantly more general than previous attempts as:

- it models the protocol as a single dynamic asynchronous process rather than a sequence of static asynchronous processes (previously discussed in Section 3.2.4).
- it is capable of modelling both distance-vector and path-vector protocols (will be discussed in Section 4.3).

Its construction can be split into three parts: i) the network and the routing state, ii) the definition of a single step of the synchronous iteration, and iii) the definition of the full asynchronous iteration.

4.2.1 Network and routing state

This section describes how the network topology and the routing state are modelled. Although this thesis only considers vector-based protocols, this part of the model is applicable to both vector-based and link-state protocols, as well as many other hybrid next-hop routing schemes. Let $(S,\ \oplus,\ E,\ \overline{\infty},\ \overline{0})$ be the routing problem that the protocol is trying to solve and consider some network in which the protocol is being run.

The topology of the network is assumed to vary over time: nodes and edges may be added and removed and the weights of edges may fluctuate. As discussed in Section 3.2.4, it is problematic to model the removal and addition of nodes directly as nodes that have been removed may still have messages in flight. Instead the model assumes that membership of the network is constant over time, but at a given point in time only a subset of nodes may be actively participating in the protocol. Upon ceasing to participate, a node will no longer be able to send new messages to the rest of the network.

Let n be the number of nodes in the network. As discussed in Section 3.3 each period of stability is referred to as an epoch. Within an epoch e, the current topology is represented by an $n \times n$ matrix $\mathbf{N}^e$ where $\mathbf{N}^e_{ij} \in E_{ij}$ is the current function that the edge from node i to node j is labelled with. The absence of an edge can be represented by f_∞, the constantly invalid extension function (i.e. $f_\infty(x) \triangleq \overline{\infty}$).

However not all nodes participate during every epoch. Therefore $\mathbf{N}^e$ is not actually the adjacency matrix for the routing problem being solved during epoch e, as non-participating nodes should not accept or broadcast routes. Let e be the current epoch and p the current set of participating nodes, then $\mathbf{A}^{ep}$, the $n \times n$ adjacency matrix for the actual routing problem being solved, is defined as:

$$\mathbf{A}^{ep}_{ij} = \begin{cases} \mathbf{N}^e_{ij} & \text{if } \{i, j\} \subseteq p \\ f_\infty & \text{otherwise} \end{cases}$$

Hence nodes that are not participating are only capable of advertising the invalid route.

Next consider the state of a routing computation. Each node i has its own routing table which will be assumed to contain an entry for every other node in the network. Hence a routing table can be modelled as an element of S^n, i.e. a vector of size n over the set of path-weights S. The global state of the computation is therefore comprised of the routing tables of every node (including non-participating nodes) and is therefore an element of $S^{n \times n}$, i.e. an $n \times n$ matrix over S. Using this representation, and given a state $\mathbf{X} \in S^{n \times n}$, then the row $\mathbf{X}_i$ is the current routing table of node i and $\mathbf{X}_{ij}$ is the weight of node i's current path to node j.

Note that for brevity's sake the entry $\mathbf{X}_{ij}$ will often be referred to as i's "route" to j. This is a minor abuse of terminology as $\mathbf{X}_{ij}$ is a path weight rather than a route/path.

When describing a sequence of nodes in the network the word “path” will be used instead.

Upon initialisation every node knows that it can route to itself via the trivial route and otherwise has no knowledge of any other routes in the network. The initial state is therefore represented by the identity matrix $\mathbf{I}$ where:

$$\mathbf{I}_{ij} \triangleq \begin{cases} \overline{0} & \text{if } i = j, \\ \overline{\infty} & \text{otherwise.} \end{cases}$$

4.2.2 A single iteration

In a vector-based protocol, a node always adopts the best extension of the routes being advertised by its neighbours. Suppose $\mathbf{X}$ is the current state of the protocol, and the current topology is represented by the adjacency matrix $\mathbf{A}$. Then the iteration operator F can be defined as:

$$F(\mathbf{X})_{ij} \triangleq \begin{cases} \overline{0} & \text{if } i = j \\ \bigoplus_k \mathbf{A}_{ik}(\mathbf{X}_{kj}) & \text{otherwise} \end{cases} \tag{4.1}$$

The definition ensures that a node always uses the trivial route to route to itself, and for every other entry in its routing table it chooses between all the possible extensions of its neighbours routes. In the latter case, the resulting route is necessarily the best extension available to node i (as proved formally by Lemma 12 at the end of this section).

The definition of F as stated above is a series of n^2 equations, one for each pair of nodes i and j. However it can be represented more succinctly as a matrix equation by defining some traditional analogues to matrix addition and multiplication [4]. The sum of two states $\mathbf{X}$ and $\mathbf{Y}$ is defined as:

$$(\mathbf{X} \oplus \mathbf{Y})_{ij} \triangleq \mathbf{X}_{ij} \oplus \mathbf{Y}_{ij}.$$

The application of the adjacency matrix $\mathbf{A}$ to a state $\mathbf{X}$ is defined as:

$$\mathbf{A}(\mathbf{X})_{ij} \triangleq \bigoplus_k \mathbf{A}_{ik}(\mathbf{X}_{kj})$$

Using this notation it is easy to verify that Equation 4.1 is equivalent to:

$$F(\mathbf{X}) \triangleq \mathbf{A}(\mathbf{X}) \oplus \mathbf{I}$$

The desired solution to the routing problem is therefore a state $\mathbf{X}^*$ that is a fixed point for F:

$$\mathbf{X}^* = \mathbf{A}(\mathbf{X}^*) \oplus \mathbf{I}$$

or equivalently:

$$\forall ij : \mathbf{X}^*_{ij} = \begin{cases} \overline{0} & \text{if } i = j \\ \bigoplus_k \mathbf{A}_{ik}(\mathbf{X}^*_{kj}) & \text{otherwise} \end{cases}$$

This characterisation of the fixed point $\mathbf{X}^*$ as a state in which no node can improve by unilaterally switching its routes, suggests that vector-based routing can be viewed as an n player game. In such a game the set of functions assigned to incoming links corresponds to a node's (pure) strategy and for a given set of strategies then the fixed point $\mathbf{X}^*$, if it exists, can be characterised as a Nash equilibrium [59]. Although interesting, this thesis will not explore this connection further.

Before moving on to defining the full asynchronous model for the protocol, a couple of lemmas about F will now be proved that will be of use in later chapters.

Lemma 11. A node always routes to itself via the trivial route:

$$\forall \mathbf{X}, i : F(\mathbf{X})_{ii} = \overline{0}$$

Proof. Immediate from the definition of F. □

Lemma 12. A route adopted during an iteration is necessarily the best route available:

$$\forall \mathbf{X}, i, j, l : \bigoplus_k \mathbf{A}_{ik}(\mathbf{X}_{kj}) \leq \mathbf{A}_{il}(\mathbf{X}_{lj})$$

Proof. Consider an arbitrary state $\mathbf{X}$ and nodes i, j and l. Then the fact that $\bigoplus_k \mathbf{A}_{ik}(\mathbf{X}_{kj})$ is at least as desirable as $\mathbf{A}_{il}(\mathbf{X}_{lj})$ follows from assumption (R1) in the definition of a routing algebra as:

$$\begin{aligned}
\bigoplus_k \mathbf{A}_{ik}(\mathbf{X}_{kj}) &= \left(\bigoplus_{k \neq l} \mathbf{A}_{ik}(\mathbf{X}_{kj})\right) \oplus \mathbf{A}_{il}(\mathbf{X}_{lj}) && (\oplus \text{ is assoc and comm}) \\
&= \left(\bigoplus_{k \neq l} \mathbf{A}_{ik}(\mathbf{X}_{kj})\right) \oplus (\mathbf{A}_{il}(\mathbf{X}_{lj}) \oplus \mathbf{A}_{il}(\mathbf{X}_{lj})) && (\oplus \text{ is selective}) \\
&= \left(\left(\bigoplus_{k \neq l} \mathbf{A}_{ik}(\mathbf{X}_{kj})\right) \oplus \mathbf{A}_{il}(\mathbf{X}_{lj})\right) \oplus \mathbf{A}_{il}(\mathbf{X}_{lj}) && (\oplus \text{ is assoc}) \\
&= \left(\bigoplus_{k} \mathbf{A}_{ik}(\mathbf{X}_{kj})\right) \oplus \mathbf{A}_{il}(\mathbf{X}_{lj}) && (\oplus \text{ is assoc and comm})
\end{aligned}$$

which is exactly the definition of $\bigoplus_k \mathbf{A}_{ik}(\mathbf{X}_{kj}) \leq \mathbf{A}_{il}(\mathbf{X}_{lj})$ (see definition of $\leq$ in Appendix A). □

4.2.3 The asynchronous state function

Having defined the operator for a single synchronous iteration, it is now possible to construct an instance of the general dynamic asynchronous iteration described in Chapter 3. The state space for the iteration is:

$$S^{n \times n} = S^n \times S^n \times ... \times S^n$$

The operator indexed by epoch e and participants p is $F^{ep}(\mathbf{X}) = \mathbf{A}^{ep}(\mathbf{X}) \oplus \mathbf{I}$ where:

$$F^{ep}(\mathbf{X}) = (F^{ep}(\mathbf{X})_1, F^{ep}(\mathbf{X})_2, ..., F^{ep}(\mathbf{X})_n)$$

The non-participating state $\perp$ is the identity matrix $\mathbf{I}$. Hence the asynchronous state function for starting state $\mathbf{X}$ and schedule $(\alpha, \beta, \eta, \pi)$ is defined as:

$$\delta^t(\mathbf{X})_{ij} = \begin{cases} \mathbf{I}_{ij} & \text{if } i \notin \rho(t) \\ \mathbf{X}_{ij} & \text{else if } t = 0 \text{ or } i \notin \rho(t-1) \\ \delta^{t-1}(\mathbf{X})_{ij} & \text{else if } i \notin \alpha(t) \\ \bigoplus_k \mathbf{A}_{ik}^{\eta(t),\rho(t)}(\delta^{\beta(t,i,k)}(\mathbf{X})_{kj}) \oplus \mathbf{I}_{ij} & \text{otherwise} \end{cases}$$

Although this function looks complex, Theorem 6 from Chapter 3 allows it to be reasoned about purely by examining the properties of F. Only having to reason about F greatly simplifies the resulting proofs and Chapter 5 will prove conditions for δ being deterministically convergent without ever reasoning about δ itself. In contrast Gao and Rexford [23] and Sobrinho [62] reason directly about their equivalent of δ and, perhaps consequently, make simplifying assumptions such as in-order, reliable delivery of messages.

4.3 Moving paths from algorithm to algebra

So far this chapter has avoided discussing whether it is modelling a distance-vector protocol or a path-vector protocol. As described in Section 2.2.3, routing table entries in path-vector protocols also store the path along which the path weight was generated. This is then used to detect and eliminate path-weights generated along looping paths. These operations have historically been modelled as part of the algorithm rather than as part of the algebra. This section discusses the drawbacks of this approach, and then demonstrates that the new definition of a routing algebra is expressive enough to model these path operations within the algebra itself. This therefore unifies distance-vector and path-vector protocols into a single algorithm for the first time.

4.3.1 What is a path?

In order to discuss path-vector protocols, it is first necessary to define what a "path" is. This is not as straight-forward as one might initially think, as it is easy to unintentionally conflate the paths in the current network topology with the representation of paths used by the protocol. Although at first glance they might be similar, the fact that the network changes over time means that the path associated with a path-weight may not correspond to a path in the current adjacency matrix. Furthermore, and perhaps counter-intuitively, the path associated with a path-weight may *never* have existed in any past adjacency matrix, and instead may be constructed from fragments of real paths from various epochs.

A path is therefore defined to be a (possibly empty) list of arcs $[(i, j), (j, k), ..., (l, m)]$ where an arc (i, j) need not correspond to an edge in the current adjacency matrix. For convenience an additional path, $\bot$, will be defined that will correspond to the path of the invalid route $\overline{\infty}$. Accordingly $\bot$ represents the absence of a path and the path $[\,]$ is the empty path from any node to itself. A path is *simple* if it never contains a node more than once and the set of simple paths will be referred to as $\mathcal{P}$. The invalid path $\bot$ is considered to be simple by convention.

Note that paths have been formalised as a list of arcs rather than a list of vertices due to the ambiguity in the meaning of $[\,]$ vs $[\ i\ ]$. A convincing argument could be made that both represent the trivial path of length 0 from node i to itself. Representing paths as list of arcs neatly side-steps this issue.

Some additional notation for constructing paths is now defined. The concatenation of an arc (i, j) to a valid path p is defined as follows:

$$(i, j) :: p \equiv \begin{cases} [\ (i, j)\ ] & \text{if } p = [\,] \\ [\ (i, j),\ (j, k),\ ...\ ] & \text{if } p = [\ (j, k),\ ...\ ] \end{cases}$$

Note that the :: operator requires that the end of the arc aligns with the start of the path. The "smart" concatenation operator, $\mathbin{\hat{::}}$, over both valid and invalid paths avoids this problem:

$$(i,\ j) \mathbin{\hat{::}} p \triangleq \begin{cases} \bot & \text{if } p = \bot \text{ or } i \in p \text{ or } j \neq \mathit{source}(p) \\ (i,\ j) :: p & \text{otherwise} \end{cases}$$

The result of $(i, j) \mathbin{\hat{::}} p$ is the invalid path $\bot$ if p is the invalid path or i is already in the path or if the first node in p is not j, otherwise it returns $(i, j) :: p$. For example:

$$\begin{aligned} (5,\ 3) \mathbin{\hat{::}} [(3,\ 4),\ (4,\ 5)] &= \bot \\ (2,\ 1) \mathbin{\hat{::}} [(3,\ 4),\ (4,\ 5)] &= \bot \\ (2,\ 3) \mathbin{\hat{::}} [(3,\ 4),\ (4,\ 5)] &= [(2,\ 3),\ (3,\ 4),\ (4,\ 5)] \end{aligned}$$

Consequently if p is a simple path then $(i, j) \mathbin{\hat{::}} p$ is also guaranteed to be a simple path.

Finally the *weight* of a path with respect to the current network topology, $\mathbf{A}$, is defined recursively as follows:

$$\begin{aligned} weight_{\mathbf{A}}(\bot) &\triangleq \overline{\infty} \\ weight_{\mathbf{A}}([\,]) &\triangleq \overline{0} \\ weight_{\mathbf{A}}((i,j) :: p) &\triangleq \mathbf{A}_{ij}(weight_{\mathbf{A}}(p)) \end{aligned}$$

4.3.2 Algorithmic paths

The problem with modelling path-vector protocols is that the internal structure of S, the set of the path-weights, is opaque and so even if it is assumed that every path-weight contains a path, the model provides no mechanism for extracting the path from the path-weight. Sobrinho addressed this problem by defining the algebra over S and then having the algorithm operate not over S but over $S \times \mathcal{P}$. The algorithm itself was then altered to correctly extend paths and remove any routes that had looping paths. In the model described so far in this thesis this would be equivalent to the operator F explicitly checking for and removing looping paths. While this approach is workable, it has some significant disadvantages in practice.

1. Firstly it makes modelling many common operations such as path filtering more difficult. The choice and extension operators, $\oplus$ and E, operate on elements of S rather than $S \times \mathcal{P}$. Therefore natural routing operations such as path filtering (e.g. reject all routes through node 7 or accept only routes with paths longer than 4) cannot be easily performed as the operators have no access to the associated path. One "hack" would be to duplicate the paths internally in S. However ensuring consistency between the internal and external paths is theoretically messy. Furthermore it somewhat unsatisfactory as a model, as duplicating the path information in this fashion within a real protocol would be needlessly inefficient.

2. Secondly, adding paths to the algorithm results in distinct algorithms for distance-vector and path-vector protocols. Consequently results proved about distance-vector protocols cannot be applied directly to path-vector protocols. This is counter-intuitive as the latter is a special case of the former, and so if a distance-vector protocol is convergent then its path-vector counterpart should also be convergent.

3. Thirdly, proofs are now sensitive to how paths are implemented. Any modifications to the implementation of the path operations results in a new algorithm and therefore the old proofs no longer apply. For example AS padding in BGP duplicates nodes

within the path to artificially inflate the path length. Implementing this in Sobrinho's model would require all the proofs to be redone.

For these reasons this thesis argues that implementing paths within the algorithm breaks the algebra + algorithm abstraction that is central to algebraic routing.

4.3.3 Algebraic paths

As one of the contributions of this thesis to the underlying model, it is now demonstrated how the new routing algebra structure can be used to represent paths within the algebra.

> **Definition 29.** A *path algebra* is a routing algebra $(S,\ \oplus,\ E,\ \overline{\infty},\ \overline{0})$ equipped with a projection function $path : S \rightarrow \mathcal{P}$ which has the following properties:
>
> (P1) $\forall x \in S : x = \overline{0} \Rightarrow path(x) = [\,]$
>
> (P2) $\forall x \in S : x = \overline{\infty} \Leftrightarrow path(x) = \bot$
>
> (P3) $\forall x \in S, ij \in V, f \in E_{ij} : path(f(x)) = \begin{cases} \bot & \text{if } f(x) = \overline{\infty} \\ (i,j) \mathbin{\hat{::}} path(x) & \text{otherwise} \end{cases}$

The key insight is that the projection function *path* does not force the algebra to reveal how it stores and operates on the paths internally. Instead it only requires that the associated path can always be constructed from a path-weight, and that the construction obeys the natural properties one might expect. Note that this definition is the reason why the definition of a routing algebra requires a set of extension functions per arc. If there was only a single set of extension functions then it would be impossible to state (P3).

The notion of a path algebra addresses all the previously discussed disadvantages of Sobrinho's approach. Firstly path algebras are a subset of routing algebras and δ is defined identically for both. Hence the same proof can apply to both distance-vector and path-vector protocols. This property is used by Lemma 20 in Chapter 5. Secondly paths are now stored within the path weight and hence the choice and extension operators can inspect the paths and perform operations such as path filtering. Finally it is not specified *how* paths are represented, only that one *can* extract a path from a path-weight. Therefore changing the internal implementation only requires changing the *path* function, and hence does not require redoing the proofs. Chapter 7 discusses how this can be used to quickly and easily handle more complex path features such as path inflation.

4.3.4 Constructing path algebras

This section presents a general algebraic transformation $\mathbb{P}(\cdot)$ to turn any routing algebra into a path algebra. Obviously because of its generality, the resulting choice and extension operators cannot make decisions based on the associated paths – something that was highlighted in the previous section as a key advantage of path algebras. Nonetheless this transformation is often useful as a quick way to transform an algebra such as shortest-paths from a distance-vector routing problem to path-vector routing problem. It also demonstrates that it is possible to satisfy the requirements of a path algebra.

Given a routing algebra $\mathbb{A} = (S,\ \oplus,\ E,\ \overline{\infty},\ \overline{0})$ a path algebra can be defined as follows:

$$\mathbb{P}(\mathbb{A}) \triangleq (S_p,\ \oplus_p,\ E_p,\ \overline{\infty}_p,\ \overline{0}_p,\ path).$$

where:

- Path weights – path weights in the augmented algebra are either of the form $(s,\ p)$, where s is any valid path weight from the original algebra and p is a path, or $\overline{\infty}_p$, a new invalid path weight.
$$S_p \triangleq (S \times \mathcal{P}) \cup \{\overline{\infty}_p\}$$
- Choice – First a choice operator for paths, $\hat{\oplus}$, is defined as follows:
$$p \mathbin{\hat{\oplus}} q \triangleq \begin{cases} p & \text{if } |p| < |q| \\ q & \text{if } |q| < |p| \\ \text{lex}(p, q) & \text{otherwise} \end{cases}$$
where $\text{lex}(p, q)$ returns the smallest path in lexicographic order. This is subsequently used to define $\oplus_p$, the main choice operator, as follows:
$$\begin{aligned} \overline{\infty}_p \oplus_p (y,\ q) &\triangleq (y,\ q) \\ (x,\ p) \oplus_p \overline{\infty}_p &\triangleq (x,\ p) \\ (x,\ p) \oplus_p (y,\ q) &\triangleq \begin{cases} (x,\ p) & \text{if } x = (x \oplus y) \neq y \\ (y,\ q) & \text{if } x \neq (x \oplus y) = y \\ (x,\ p \mathbin{\hat{\oplus}} q) & \text{if } x = y \end{cases} \end{aligned}$$
- Extension – For every pair of nodes i and j and extension function $f \in E_{ij}$ then a new extension function g_f is defined as follows:
$$\begin{aligned} g_f(\overline{\infty}_p) &\triangleq \overline{\infty}_p \\ g_f(x, p) &\triangleq \begin{cases} \overline{\infty}_p & \text{if } f(x) = \overline{\infty} \text{ or } (i,\ j) \mathbin{\overset{\frown}{::}} p = \bot \\ (f(x),\ (i,\ j) \mathbin{\overset{\frown}{::}} p) & \text{otherwise} \end{cases} \end{aligned}$$
- Trivial route – The new trivial route is the old trivial route paired with the empty

path:

$$\overline{0}_p \triangleq (\overline{0},\ [\,])$$

- Path function – The path function required by the path algebra is defined as follows:

$$\begin{aligned} path(\overline{\infty}_p) &\triangleq \bot \\ path(x, p) &\triangleq p \end{aligned}$$

It is easy to verify that the resulting algebra $\mathbb{P}(\mathcal{A})$ obeys all the required axioms of a path algebra, whilst preserving the properties required for to be a routing algebra. It should be noted that there is nothing canonical about $\mathbb{P}(\cdot)$'s method of adding paths to a routing algebra. It is just one of many possible methods of doing so.

The concept of a path algebra and the construction $\mathbb{P}(\cdot)$ will be used in the next section to build several example algebras that model the routing problems being solved by various possible path-vector protocols.

4.4 Examples of routing algebras

This section will now present some examples of how common routing problems may be represented as routing algebras. Some of the examples are designed to demonstrate the flexibility of the model whilst others will be used as examples in later chapters.

4.4.1 Shortest paths algebras

The most familiar examples can be constructed around shortest path algebras of the form $(\mathbb{N}^\infty,\ \min, E, \infty, 0)$ where E is some suitable collection of additive functions. The notation f_n will be used to denote the function $f_n(x) = x + n$. Furthermore suppose that P is a predicate over the weights in $\mathbb{N}^\infty$ and let g_n be defined as:

$$g_n(x) = \begin{cases} f_n(x) & \text{if } P(x) \\ \infty & \text{otherwise} \end{cases}$$

Here are few sets of functions that result in useful routing algebras:

$$\begin{array}{lcll} E_1 & \triangleq & \{f_n \mid n \in \mathbb{N}^\infty\} & \text{(shortest-paths)} \\ E_2 & \triangleq & \{g_n \mid n \in \mathbb{N}^\infty\} & \text{(shortest-paths with filtering)} \\ E_3 & \triangleq & \{f_1,\ f_\infty\} & \text{(hop-count)} \\ E_4 & \triangleq & \{g_1,\ f_\infty\} & \text{(hop-count with filtering)} \end{array}$$

Note that because none of these algebras perform any path operations the set of functions E_{ij} is the same for all nodes i and j. Any of these algebras can be transformed into a path-algebra using the $\mathbb{P}(\cdot)$ construction.

4.4.2 Shortest widest paths algebra

Another algebra of interest is the shortest widest paths algebra $\mathbb{SWP}$. The aim of this routing problem is to find the path with the maximum bandwidth and if there are multiple such paths then ties are broken by the length of the path.

Path weights in $\mathbb{SWP}$ are of the form $(b,\ d)$ where b is the bandwidth and d is the length. Thus $\overline{0} \triangleq (\infty, 0)$ is the weight of the trivial path, while $\overline{\infty} \triangleq (0, \infty)$ is the weight of the invalid path. Path weights are chosen between by first comparing their bandwidth and then breaking ties by their length:

$$(a,\ b) \oplus (c,\ d) \triangleq \begin{cases} (a, b) & \text{if } a = \max(a, c) \neq c \\ (c, d) & \text{if } a \neq \max(a, c) = c \\ (a, \min(b, d)) & \text{else if } a = c \end{cases}$$

As with the shortest-path algebras, every edge is associated with an identical set of extension functions. The extension function for an edge with bandwidth w and length l is defined as follows:

$$f_{c,w}(b,\ d) \triangleq (\min(c,\ b),\ w + d).$$

One consequence of this definition is that the bandwidth component of a path-weight represents the bandwidth of the bottleneck link along the path it was generated by. The full set of extension functions is:

$$E_{ij} = \{f_{c,w} \mid c, w \in \mathbb{N}^{\infty}\}$$

The shortest-widest-paths path algebra $\mathbb{SWPP}$ is then defined as $\mathbb{P}(\mathbb{SWP})$.

4.4.3 Stratified shortest paths algebra

Another interesting path algebra is the stratified shortest path algebra [27]. Its construction starts with the increasing routing algebra:

$$\mathbb{INC} \triangleq (\mathbb{N}^{\infty},\ \min,\ E^{inc},\ 0,\ \infty),$$

where $E^{inc}_{ij} = \{f \mid \forall x : x \leq f(x)\}$ is the set of non-decreasing functions. The path weights of $\mathbb{INC}$ are referred to as levels and by definition the extension functions cannot decrease the level. For notational convenience this thesis further restricts E_{ij}^{inc} to only include functions that can be represented as vectors over $\mathbb{N}^{\infty}$. The application of the function $f = \langle s_0,\ s_1,\ \ldots,\ s_k \rangle$ is defined as:

$$f(i) \triangleq \begin{cases} s_i & \text{if } i \leq k \\ \infty & \text{if } i > k \end{cases}$$

For example:

$$\langle 0, \infty, 17, 3 \rangle(i) = \begin{cases} 0 & \text{if } i = 0 \\ \infty & \text{if } i = 1 \\ 17 & \text{if } i = 2 \\ 3 & \text{if } i = 3 \\ \infty & \text{if } i > 3 \end{cases}$$

This algebra can implement the customer/provider/peer policies from [23] & [38] which were discussed in Section 2.3.2.2. Imagine that level 0 represents routes from customers, level 1 represents routes from peers, and level 2 represents routes from providers. Then [62] shows that the standard customer/provider/peer policy functions can be represented as:

$$\begin{array}{ll} \langle 0,\ \infty,\ \infty \rangle & \text{(towards customers)} \\ \langle 1,\ \infty,\ \infty \rangle & \text{(towards peers)} \\ \langle 2,\ \ \ 2,\ \ \ 2 \rangle & \text{(towards providers)} \end{array}$$

The $\mathbb{P}(\cdot)$ transformation can then be used to add paths to $\mathbb{INC}$ to form the full stratified shortest paths algebra:

$$\mathbb{SPP} \triangleq \mathbb{P}(\mathbb{INC}).$$

The stratified shortest paths algebra will used in Chapter 6 to construct an example of a path-vector protocol that requires a quadratic number of iterations to converge.

4.4.4 Other algebras

An Agda formalisation of a more interesting and complex path algebra that contains many of the core features of BGP such as communities, conditional policy, path inflation and local preferences is described in Chapter 8.

4.5 Existing results

The field of algebraic routing underwent a revival after awareness of the problems with BGP discussed in Section 2.3.2.1 began to increase. In particular it was motivated by the question of which combination of BGP policies resulted in a stable protocol. This section discusses the key recent results in the field.

One of the first of the new wave of results was by Griffin et al. [33] who showed that the problem of deciding whether there exists a stable state for an arbitrary network and routing problem is NP-hard. This was a blow for routing protocol designers as it implied that it was infeasible to allow network operators the freedom to specify arbitrary policies and then verify the correctness of the chosen policies on the fly. Consequently the search then turned to finding some subset of networks and routing problems for which i) protocols were guaranteed to converge and ii) the subset was equipped with an efficient decision procedure for deciding membership.

A couple of years later Gao and Rexford [23] published the customer-peer-provider conditions for BGP. As discussed in Section 2.3.2.2, while it is possible to verify if the conditions hold, the process is expensive and requires global coordination between ASs. Furthermore the conditions were BGP-specific and didn't provide insight into the conditions required more generally for convergence.

Shortly afterwards Sobrinho [62] produced the foundational paper for the post-semiring phase of algebraic routing. In it he proposed the algebras described in Section 4.1.2 and used them to construct a model for an abstract path-vector protocol. He then showed that the assumption of distributivity in the semiring theory was merely one of several possible relationships between the choice operator and the extension operator. In particular he identified three types of routing problems: i) problems in free networks, ii) problems with strictly increasing algebras and iii) problems with distributive algebras.

These types of routing problems are now explored in more depth. Note that in the following definitions the path-weights x and y are arbitrary members of S and the extension function f is an arbitrary member of some E_{ij} for some arbitrary pair of nodes i and j.

4.5.1 Free networks

> **Definition 30.** A network topology $\mathbf{A}$ is *free* with respect to a routing algebra if for every cycle in the topology $[(v_0, v_1), (v_1, v_2), ..., (v_{m-1}, v_0)]$ and set of valid weights $x_0, x_1, ..., x_{m-1}$, then there exists a $k \leq m - 1$ such that:
>
> $$x_k < \mathbf{A}_{v_k v_{k+1}}(x_{k+1})$$
>
> where $k + 1$ is calculated mod m.

Figure 4.1 shows a visualisation of this condition. Intuition can perhaps be gained by considering a cycle that violates the condition. In such a cycle every node prefers to route via its clockwise neighbour, and so a forwarding loop will form. As an example consider a shortest-paths algebra where negative length edges are allowed. A network is free with respect to this algebra if and only if there are no negative weight cycles. If there is a negative weight cycle, then the nodes in that cycle will always prefer to continue routing round the cycle indefinitely. The name “free” originates from the fact that the network is free of such cycles.

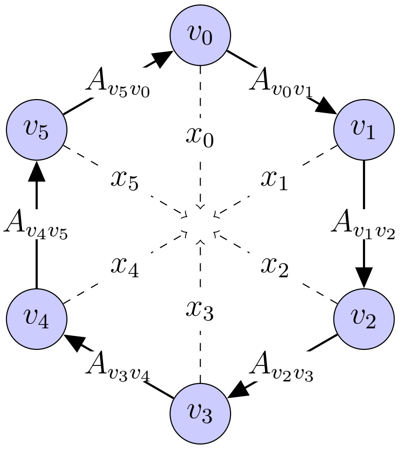


Figure 4.1: A free network forbids cycles of the above form where every node prefers to route via it’s clockwise neighbour.

Sobrinho showed that a network topology **A** being free with respect to a path algebra is both a sufficient and necessary condition for the associated path-vector protocol to converge over A. Consequently if A is not free then the protocol will diverge. Furthermore Sobrinho showed that the Gao-Rexford conditions work because the restrictions on consumer-peer-provider policies, combined with the requirement that the consumer-provider relationship forms a directed acyclic graph, is sufficient to guarantee that the network topology is free with respect to BGP’s algebra.

4.5.2 Strictly increasing

The next obvious question is what are the requirements on an algebra such that every network is free with respect to it? The answer is that the algebra must be *strictly increasing*.

> **Definition 31.** A routing algebra is *increasing* if:
>
> $$\forall f, x : x \leq f(x)$$

> **Definition 32.** A routing algebra is *strictly increasing* if:
>
> $$\forall f, x : x \neq \overline{\infty} \Rightarrow x < f(x)$$

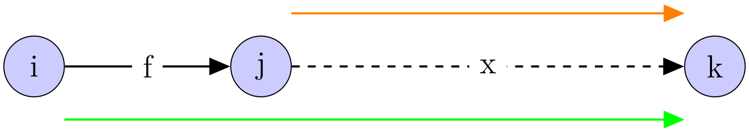


Figure 4.2: The (strictly) increasing condition says that $f(x)$, the result of extending x, must always be worse than x itself.

Essentially an algebra is increasing if extending a route never makes it better and an algebra is *strictly* increasing if extending a valid route always makes it worse.

Sobrinho showed that a path-vector protocol converges over all network topologies if and only if the underlying Sobrinho algebra is strictly increasing. Equivalently if the algebra is not strictly increasing then there exists a network for which the path-vector protocols will not converge. Intuitively this makes sense, as to converge implies reaching a stable state such that no node can improve its own route by unilaterally switching its routing choices. If the algebra is not strictly increasing, then nodes can completely ignore each other's preferences and therefore the existence of a stable state cannot be guaranteed.

It should be noted that assumption (P3) in the definition of a path algebra ensures that any path algebra that is increasing is also necessarily strictly increasing. This is because the extension of a path can never have the same weight as the original path because the former must have a longer path than the latter. Consequently the terms "increasing" and "strictly increasing" can be used interchangeably for path algebras.

4.5.3 Distributivity

Finally Sobrinho examined the classical semiring property of distributivity.

> **Definition 33.** A routing algebra is *distributive* if:
>
> $$\forall f, x, y : f(x \oplus y) = f(x) \oplus f(y)$$

In distributive algebras all nodes agree on which are the best paths through the network. To see why this is the case consider Figure 4.3. Distributivity implies that it doesn't matter whether node j chooses between routes x and y and node i extends the resulting route or node i gets to choose directly between the extensions $f(x)$ and $f(y)$.

Distributivity is not a necessary condition for a protocol to always converge. However it is a necessary condition for the final state reached to always be a global optimum. A state $\mathbf{X}^*$ is a global optimum if for every pair of nodes i and j the route $\mathbf{X}^*_{ij}$ is the best possible route from i to j in the current topology. Distributivity is therefore a highly desirable property for a routing algebra to possess.

Many simple routing problems are distributive, e.g. hop-count, shortest paths, widest paths, most reliable path. However there exist several families of interesting routing

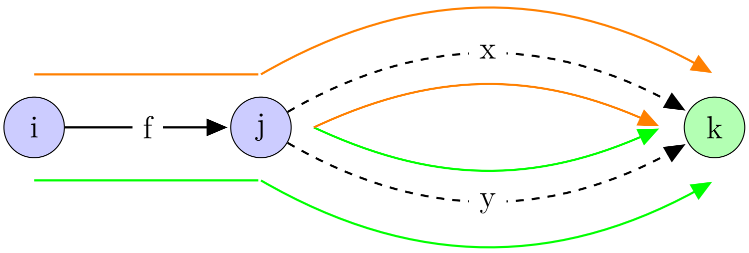


Figure 4.3: An algebra is distributive if node j chooses x over y if and only if node i chooses $f(x)$ over $f(y)$ for all f, x and y.

problems that are not distributive. In particular, as illustrated by the introductory example in Chapter 1, any algebra with conditional policy is unlikely to be distributive.

However conditional policy is not the only way of violating distributivity. For instance, despite both the shortest path and the widest path algebras being distributive, the shortest widest path algebra, described in Section 4.4.2, is not. This is because a link that acts as a bandwidth bottleneck can hide other nodes' bandwidth-based decisions. For example consider some bandwidth $k > 1$ and let $x = (1,\ 1)$, $y = (2,\ k)$ and $f = f_{1,1}$ (i.e. a link with length 1 and bandwidth 1). Then:

$$\begin{aligned}
f(x \oplus y) &= f_{1,1}((1,\ 1) \oplus (2,\ k)) \\
&= f_{1,1}(2,\ k) \\
&= (1,\ k+1) \\
&\neq (1,\ 2) \\
&= (1,\ 2) \oplus (1,\ k+1) \\
&= f_{1,1}(1,\ 1) \oplus f_{1,1}(2,\ k) \\
&= f(x) \oplus f(y)
\end{aligned}$$

Node j prefers route y over route x, despite the fact that y is much longer, because of y's higher bandwidth. However the link from node i to node j only has bandwidth 1 and so from node i's perspective it doesn't matter what the bandwidth of j's chosen route is. Therefore their routing choices conflict as node i prefers to route via x due to its shorter length and node j prefers to route via y due to its higher bandwidth.

4.5.4 Further discussion

This section now compares and contrasts the three different types of routing problems, both in terms of their implications for the make-up of the network and their suitability as correctness criteria.

Politics The three different types of routing problems can be seen as measuring the degree of co-operation between nodes in the network.

- Distributivity = “consensus” – in a distributive algebra, the nodes are in agreement over which are the best paths in the network and therefore their incentives are aligned. Consequently a globally optimal solution can be reached where every node is assigned the best possible route.

- Strictly increasing = “respect” – in a strictly increasing algebra, the nodes in the network have differing opinions over which are the best routes, but nonetheless respect each other’s opinions. If a node’s neighbour says that a route is bad than the node will respect that neighbour’s decision and will not re-advertise its extension as a good route to other neighbours. Consequently a locally optimal solution can always be reached, however not every node will be assigned their best possible route.

- Non-strictly increasing = “chaos” – in a non-strictly increasing algebra, then not only do nodes disagree about which are the best paths, but they also do not respect each other’s opinions. A node may accept a route that a neighbour says is bad and advertise it to its neighbours as a good route. Hence the protocol will only converge in free networks where it happens that there exists no negative weight cycles.

Suitability as verification criteria The goal is to ensure that a routing protocol always converges. The problem with using network freeness is that, as a consequence of the work by Griffin et al. [33] discussed at the start of this section, in general deciding whether a given network is free with respect to a protocol’s algebra is NP-hard. Furthermore in a dynamic network, the freeness property must be rechecked every time a node or a link is added. Consequently network freeness is not a practical criterion for guaranteeing convergence.

In contrast, checking that a protocol is strictly increasing only needs to be done once by the protocol designers, rather than every time the network changes. By virtue of being a necessary condition for converging over all networks, the strictly increasing property is also the minimal such property. Furthermore for most algebras it is possible to determine whether or not it is strictly increasing simply by inspection of the choice and extension operators. Consequently this thesis focuses on further exploring the properties of strictly increasing algebras.

Other uses of algebraic routing Algebraic routing has been applied to other problems in routing besides protocol convergence. For example it has been used to model opportunistic routing [45], interactions between different routing protocols [43] and mapping demands from virtual networks to physical network resources [11]. Concurrently there has a drive in the field of meta-routing [30] to better understand the ways in which routing algebras may be constructed from smaller, simpler routing algebras [34, 36]. The $\mathbb{P}(\cdot)$ construction presented in Section 4.3.4 could be seen as a contribution to this work.

4.6 Open questions to be addressed

As mentioned in the previous section, this thesis will focus on the properties of protocols with strictly increasing algebras. Although Sobrinho showed that strictly increasing path-vector protocols always converge there are still several open questions:

1. Are strictly increasing *distance*-vector protocols guaranteed to converge as well? If so then this would suggest that distance-vector protocols could be equipped with conditional policy.

2. Do strictly increasing protocols always converge to the same stable state, or does the state reached depend on the schedule and the routing state at the beginning of the epoch? If so then convergence is deterministic, and this would have the immediate consequence that problems such as BGP wedgies cannot occur in strictly increasing protocols.

3. Does the convergence of strictly increasing protocols rely on the in-order reliable delivery of messages between routers? Or is convergence still guaranteed when messages between nodes can be dropped, duplicated and reordered? If the latter, then new, lightweight vector-based protocols could be designed to use UDP rather than TCP.

4. How fast do strictly increasing path-vector protocols converge? In the worst case, distributive path-vector protocols are known to converge in $\Theta(n)$ time, where n is the number of the nodes in the network. In contrast the best known bound for strictly increasing path-vector protocols is $O(n!)$. If this exponential bound is achievable then this would have serious implications for the usability of strictly increasing protocols.

The following chapters will answer these questions.

Chapter 5

Convergence

This chapter uses the general convergence theorems for asynchronous iterations from Chapter 3 and the model constructed in Chapter 4 to prove new stronger results about the convergence of vector-based protocols for strictly increasing algebras. In particular, by constructing suitable distance functions over routing tables and routing states, it shows that F is a dynamic AMCO and hence that Theorem 6 from Chapter 3 can be used to prove that δ is deterministically convergent.

Another way the approach differs from that of Sobrinho is that initially only *finite*, strictly increasing routing algebras are considered rather than infinite, increasing path algebras. Such finite routing algebras correspond to the set of convergent distance-vector protocols. This chapter shows for the first time that such algebras are convergent and hence that distance-vector protocols can be equipped with expressive conditional policy.

In practice, and as will be discussed in greater depth at the beginning of the first section, the finite condition is restrictive and so the the chapter proceeds to construct analogous distance functions for increasing path algebras. Such algebras correspond to the set of convergent path-vector protocols. The subsequent convergence result is an improvement over the equivalent result by Sobrinho as:

1. The model more accurately describes protocols as a single dynamic iteration, rather than a sequence of static iterations (see Section 3.2.4).

2. The model is expressive enough to capture protocols equipped with path-dependent conditional policy (see Sections 4.3.2 & 4.3.3).

3. The result proves that convergence is deterministic and hence phenomena such as BGP wedgies cannot occur (see Section 2.1).

4. The proof does not assume in-order, reliable delivery of messages (see Section 2.2.2.2).

5. The proof does not require reasoning directly about the asynchronous state function. It is therefore in some sense simpler/more modular than Sobrinho's proof.

All results in this chapter have been fully formalised in Agda.

5.1 Distance-vector protocols

This section proves that when a routing algebra $(S,\ \oplus,\ E,\ \overline{\infty},\ \overline{0})$ is finite and strictly increasing then δ is deterministically convergent. To do so a distance function, d, is constructed over routing tables. It is then shown that F is an AMCO with respect to d and so satisfies the conditions for Theorem 6.

Note that in practice the finiteness condition is restrictive and excludes many routing algebras of interest. For example even the shortest-path algebra uses $\mathbb{N}$ as the set of path weights. However recall that Theorem 7 guarantees that, given a sufficient period of stability, the protocol will reconverge even after changes to the network topology. On the other hand shortest-path distance-vector protocols experience count-to-infinity problems when the state at the start of the epoch contains junk routes generated along paths that do not exist in the current topology. Hence finiteness is a necessary condition to prevent count-to-convergence in distance-vector protocols. In the real world this is reflected in the design of RIP which artificially imposes a maximum hop count to ensure that S is finite (see Section 2.3.1).

Height of routes As S is finite, all upwards closed subsets under the relation $\leq$ must also be finite. Consequently the *height* of a route can be defined as follows:

$$h(x) \triangleq |\{y \in S \mid x \leq y\}|$$

Intuitively the height of a route is proportional to its desirability. The trivial route, $\overline{0}$, is the most desirable route and so has the maximum height, referred to as H. The invalid route, $\overline{\infty}$, is the least desirable route and therefore is the route with the minimal height of 1. Therefore for every route x the following relationships hold:

$$1 = h(\overline{\infty}) \leq h(x) \leq h(\overline{0}) = H$$

Distance between routes Next a distance function $r : S \times S \to \mathbb{N}$ between routes can be defined as follows:

$$r(x, y) \triangleq \begin{cases} 0 & \text{if } x = y \\ \max(h(x), h(y)) & \text{otherwise} \end{cases}$$

By this definition the distance between a pair of distinct routes is proportional to how desirable the better of the two routes is. Intuitively this is a reasonable measure of distance as better routes are more likely to be adopted by other nodes and hence be propagated throughout the network. Consequently if two routing states disagree on the best route from i to j, then, from a convergence perspective, the seriousness of the disagreement is directly proportional to the desirability of the two conflicting routes.

Distance between routing tables The distance between a pair of routing tables is defined to be the maximum of the pairwise distances between their entries:

$$d(x, y) \triangleq \max_{j} \ r(x_j, y_j)$$

If two routing tables are identical then they will have zero distance between them, otherwise the distance between them is proportional to the most desirable route they disagree on. For all nodes i, epochs e and participants p the function d will be the distance function d_i^{ep} required by the AMCO conditions. Note that in this case, the same metric d is used for all nodes i, epochs e and sets of participants p.

Also note that, thanks to the relaxations of the asynchronous fixed point theory found in Section 3.2.3, the AMCO conditions only require d to be a quasi-semi-metric (see Appendix A for definitions). The required property $d(x, y) = 0 \Leftrightarrow x = y$ can immediately be seen to hold by the definition of r and the prior inequality $1 \leq h(x)$. This is therefore an excellent illustrative example of the utility of the relaxations. Without them it would also have been necessary to prove that d was an ultrametric and hence that it obeyed the symmetry and max triangle inequality axioms as well (the latter of which is non-trivial to prove).

Distance between routing states As required by the AMCO conditions, the distance function over routing states is defined as follows:

$$D^p(\mathbf{X}, \mathbf{Y}) \triangleq \max_{i \in p} \ d(\mathbf{X}_i, \mathbf{Y}_i)$$

where p is the set of participants. Again D measures the distance between states $\mathbf{X}$ and $\mathbf{Y}$. If all the elements of $\mathbf{X}$ and $\mathbf{Y}$ are equal then they occupy the same point in the space, otherwise the distance between them grows in proportion to the most desirable route they disagree on.

In order to show that F is an AMCO, it is necessary to prove that for all epochs e and sets of participants p then F^{ep} is both strictly contracting over orbits and strictly contracting over the resulting fixed point with respect to the set of accordant states $\mathbf{A}_p$. In fact in this case F^{ep} can be shown to be strictly contracting, which implies it is strictly contracting over both orbits and any fixed points. Instead of proving this result immediately, a smaller lemma will first be proved that shows that if the distance between all non-equal entries of two states is less than or equal to v then the distance between any two entries after F^{ep} has been applied to both states must be *strictly* less than v. This lemma has been separated out from the main proof that F^{ep} is strictly contracting as it will later be reused in Section 5.2 in the proof of convergence for path-vector protocols.

Lemma 13. For all states $\mathbf{X}$ and $\mathbf{Y}$, nodes i and j, epoch e, set of participants p, and natural number $v > 0$, then:

$$(\forall k : \mathbf{X}_{kj} \neq \mathbf{Y}_{kj} \Rightarrow r(\mathbf{X}_{kj}, \mathbf{Y}_{kj}) \leq v) \Rightarrow r(F^{ep}(\mathbf{X})_{ij}, F^{ep}(\mathbf{Y})_{ij}) < v$$

Proof. The proof is independent of both the epoch and the participants, and hence for notational clarity the dependency of F and $\mathbf{A}$ on e and p will be dropped.
Case 1: $F(\mathbf{X})_{ij} = F(\mathbf{Y})_{ij}$
Then the required inequality is immediate as:

$$\begin{aligned} r(F(\mathbf{X})_{ij}, F(\mathbf{Y})_{ij}) &= 0 && \text{(by case 1 and def. of } r\text{)} \\ &< v && \text{(by lemma assumption)} \end{aligned}$$

Case 2: $F(\mathbf{X})_{ij} \neq F(\mathbf{Y})_{ij}$
Without loss of generality assume that $F(\mathbf{X})_{ij}$ is a more desirable route than $F(\mathbf{Y})_{ij}$ and therefore:

$$F(\mathbf{X})_{ij} < F(\mathbf{Y})_{ij} \tag{5.1}$$

Case 2.1: $i = j$
A node's route to itself is always the trivial route by Lemma 11 in Chapter 4 and so if $i = j$ then $F(\mathbf{X})_{ij} = \overline{0} = F(\mathbf{Y})_{ij}$ which contradicts case 2's assumption.

Case 2.2: $i \neq j$
Consequently by the definition of F:

$$F(\mathbf{X})_{ij} = \bigoplus_{k} \mathbf{A}_{ik}(\mathbf{X}_{kj})$$

and so as $\oplus$ is selective there must exist a node k such that:

$$F(\mathbf{X})_{ij} = \mathbf{A}_{ik}(\mathbf{X}_{kj}) \tag{5.2}$$

i.e. $F(\mathbf{X})_{ij}$ is an extension of some route in $\mathbf{X}$. If $\mathbf{X}_{kj} = \overline{\infty}$ then $F(\mathbf{X})_{ij} = \overline{\infty}$ which contradicts (5.1) and therefore:

$$\mathbf{X}_{kj} \neq \overline{\infty} \tag{5.3}$$

It cannot be the case that $\mathbf{X}_{kj} = \mathbf{Y}_{kj}$ as otherwise it would be possible to prove the following:

$$\begin{aligned} F(\mathbf{X})_{ij} &= \mathbf{A}_{ik}(\mathbf{X}_{kj}) && \text{(by 5.2)} \\ &= \mathbf{A}_{ik}(\mathbf{Y}_{kj}) && \text{(by assumption)} \\ &\geq F(\mathbf{Y})_{ij} && \text{(by Lemma 12)} \end{aligned}$$

which again contradicts (5.1) and so:

$$\mathbf{X}_{kj} \neq \mathbf{Y}_{kj} \tag{5.4}$$

The required inequality can now be proved as follows:

$$\begin{aligned} r(F(\mathbf{X})_{ij}, F(\mathbf{Y})_{ij})) &= \max(h(F(\mathbf{X})_{ij}), h(F(\mathbf{Y})_{ij})) && \text{(by case 2 and def. of } r\text{)} \\ &= h(F(\mathbf{X})_{ij}) && \text{(by 5.1)} \\ &= h(\mathbf{A}_{ik}(\mathbf{X}_{kj})) && \text{(by 5.2)} \\ &< h(\mathbf{X}_{kj}) && \text{(by str. incr. \& 5.3)} \\ &\leq \max(h(\mathbf{X}_{kj}), h(\mathbf{Y}_{kj})) && \text{(by def. of max)} \\ &= r(\mathbf{X}_{kj}, \mathbf{Y}_{kj}) && \text{(by 5.4 and def. of } r\text{)} \\ &\leq v && \text{(by 5.4 and lemma ass.)} \end{aligned}$$

□

Lemma 13 can now be used to prove that F^{ep} is strictly contracting.

Lemma 14. For all epochs e and participants p, the operator F^{ep} is strictly contracting w.r.t. D^p over A_p.

Proof. Consider accordant states $\mathbf{X}$ and $\mathbf{Y}$ such that $\mathbf{X} \neq_p \mathbf{Y}$. To prove that F^{ep} is strictly contracting, the following must be shown:

$$D^p(F^{ep}(\mathbf{X}), F^{ep}(\mathbf{Y})) < D^p(\mathbf{X}, \mathbf{Y})$$

As $D^p(F^{ep}(\mathbf{X}), F^{ep}(\mathbf{Y})) = \max_{i \in p, j \in V} r(F^{ep}(\mathbf{X})_{ij}, F^{ep}(\mathbf{Y})_{ij})$ it suffices to show for all nodes $i \in p$ and $j \in V$ that

$$r(F^{ep}(\mathbf{X})_{ij}, F^{ep}(\mathbf{Y})_{ij}) < D^p(\mathbf{X}, \mathbf{Y})$$

This can be achieved by applying Lemma 13 where $v = D^p(\mathbf{X}, \mathbf{Y})$. However first it must be checked that $0 < D^p(\mathbf{X}, \mathbf{Y})$ and that for all nodes k then $\mathbf{X}_{kj} \neq \mathbf{Y}_{kj}$ implies $r(\mathbf{X}_{kj}, \mathbf{Y}_{kj}) \leq D^p(\mathbf{X}, \mathbf{Y})$.

The former holds as $\mathbf{X} \neq \mathbf{Y}$ and so there must exist entries in $\mathbf{X}$ and $\mathbf{Y}$ which are a non-zero distance apart. To show that the latter holds consider whether or not k is participating. If $k \in p$ then the required inequality holds simply from the definition of D^p. Otherwise if $k \notin p$ then as $\mathbf{X}$ and $\mathbf{Y}$ are accordant then $\mathbf{X}_{kj} = \mathbf{Y}_{kj} = \overline{\infty}$ and the inequality holds trivially as $r(\mathbf{X}_{kj}, \mathbf{Y}_{kj}) = 0$. □

Finally all the pieces to show that F is a dynamic AMCO have now been assembled.

Theorem 7. If $(S,\ \oplus,\ E,\ \overline{\infty},\ \overline{0})$ is a finite, strictly increasing routing algebra then δ is deterministically convergent.

Proof. First it will be shown that F is a dynamic AMCO. Let $d_i^{ep} = d$ for all nodes i, epochs e and participants p. Then the required conditions are fulfilled:

(DU1) d is a quasi-semi-metric

The required property $\forall x, y : d(x, y) = 0 \Leftrightarrow x = y$ is immediate from the definition of d and r.

(DU2) d is bounded

The height of a routes is bounded above by H and hence so is every distance.

(DU3) $\forall e, p$: F^{ep} is strictly contracting on orbits w.r.t. D^{ep} over A_p

An immediate consequence of applying Lemma 14 to $\mathbf{X}$ and $F(\mathbf{X})$.

(DU4) $\forall e, p$: F^{ep} is strictly contracting on $\mathbf{X}^*$ w.r.t. D^{ep} over A_p for any fixed point $\mathbf{X}^*$

An immediate consequence of applying Lemma 14 to $\mathbf{X}$ and $\mathbf{X}^*$.

(DU5) $\forall e, p, \mathbf{X} : F^{ep}(\mathbf{X}) \in A_p$.

Immediate by the definition of F^{ep} and $\mathbf{A}^{ep}$.

As F is an AMCO then Theorem 6 may be applied and so δ is deterministically convergent over $(S,\ \oplus,\ E,\ \overline{\infty},\ \overline{0})$. □

Theorem 7 therefore shows that, given a sufficient period of stability, any distance-vector protocol over a finite, strictly increasing algebra will always converge to a unique solution even in the presence of unreliable communication.

5.2 Path-vector protocols

As discussed at the start of the previous section, the requirement that the set of path weights is finite proves very restrictive in practice. How do real vector-based routing protocols get around this constraint? The principled approach is that of path-vector protocols which track and remove routes generated along looping paths. These operations are sufficient to guarantee that eventually the protocol will always reach a finite subset of *consistent* routes from which it will then converge.

The concept of consistency will now be defined more rigorously. Consider a path algebra $(S,\ \oplus,\ E,\ \overline{\infty},\ \overline{0},\ \mathit{path})$ as defined in Section 4.3.3 and an adjacency matrix $\mathbf{A}$.

> **Definition 34.** A route x is *consistent* with respect to the topology $\mathbf{A}$ if and only if:
>
> $$\mathit{weight}_{\mathbf{A}}(\mathit{path}(x)) = x$$

i.e. the route is equal to the weight of the path along which it was generated. One consequence of this definition is that every consistent route is associated with at least one simple path in the current network topology. The set of consistent routes, $C_{\mathbf{A}}$, can therefore be defined as:

$$C_{\mathbf{A}} \triangleq \{\mathit{weight}_{\mathbf{A}}(p) \mid p \in \mathcal{P}\}$$

where $\mathcal{P}$ is the set of simple paths as defined in Section 4.3.1. As $\mathcal{P}$ is finite so too is the set of consistent routes, $C_{\mathbf{A}}$. Note that whereas a given route may or may not obey the equality required for consistency, the converse equality:

$$\mathit{path}(\mathit{weight}_{\mathbf{A}}(p)) = p$$

always holds for every path p.

> **Definition 35.** A routing state $\mathbf{X}$ is *consistent* with respect to the topology $\mathbf{A}$ iff:
>
> $$\forall i, j : \mathbf{X}_{ij} \in C_{\mathbf{A}}$$

i.e. every route in use is consistent with the current network topology. The following lemmas show that consistency is preserved by the routing operations.

Lemma 15. If $x, y \in C_{\mathbf{A}}$ then $x \oplus y \in C_{\mathbf{A}}$.

Proof. Immediate from assumption (R1) that $\oplus$ is selective. □

Lemma 16. If $x \in C_{\mathbf{A}}$ then $\mathbf{A}_{ij}(x) \in C_{\mathbf{A}}$ for all nodes i and j.

Proof. Consider a consistent route x being extended along an arbitrary edge $\mathbf{A}_{ij}$. By assumption (P3) from the definition of a path algebra, either $\mathit{path}(\mathbf{A}_{ij}(x)) = \bot$ or $\mathit{path}(\mathbf{A}_{ij}(x)) = (i, j) :: \mathit{path}(x)$. In the former case $\mathbf{A}_{ij}(x)$ is consistent by (P2), and in the latter case:

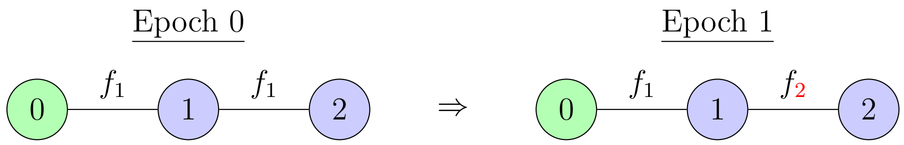
$$\begin{aligned} \mathit{weight}_{\mathbf{A}}(\mathit{path}(\mathbf{A}_{ij}(x))) &= \mathit{weight}_{\mathbf{A}}((i,j) :: \mathit{path}(x)) && \text{(by case assumption)} \\ &= \mathbf{A}_{ij}(\mathit{weight}_{\mathbf{A}}(\mathit{path}(x)) && \text{(by def. of } \mathit{weight}_{\mathbf{A}}) \\ &= \mathbf{A}_{ij}(x) && \text{(by consistency of } x) \end{aligned}$$

which is the required result. □

Lemma 17. If state $\mathbf{X}$ is consistent with $\mathbf{A}^{ep}$ then, for every epoch e and set of participants p, $F^{ep}(\mathbf{X})$ is also consistent.

Proof. This is an immediate from Lemmas 15 & 16 as every operation in F^{ep} preserves consistency. □

One consequence of Lemma 17 is that applying an update never spontaneously introduces new inconsistent routes into the routing state and hence that the only way inconsistent routes can be introduced is by a change in the network topology. Figure 5.1 demonstrates an example where this may occur.



Epoch	Time	Node 0	Node 1	Node 2
0	0	$(0, [\,])$	∞	∞
0	1	$(0, [\,])$	$(1, [(1,0)])$	∞
0	2	$(0, [\,])$	$(1, [(1,0)])$	$(2, [(2,1),(1,0)])$
1	3	$(0, [\,])$	$(1, [(1,0)])$	$(2, [(2,1),(1,0)])$
1	4	$(0, [\,])$	$(1, [(1,0)])$	$(3, [(2,1),(1,0)])$

Figure 5.1: An example of an inconsistent route being introduced into the routing state in a shortest-paths path-vector protocol where the function $f_k(x) = x + k$. At time 2 node 2's route $x = (2, [(2,1),(1,0)])$ is consistent with the topology of epoch 0 as $\mathit{weight}(\mathit{path}(x)) = (2, [(2,1),(1,0)])$. At time 3 the length of the edge $(2,1)$ increases by 1 and hence epoch 1 begins. Node 2's route is no longer consistent at time 3 as now: $\mathit{weight}(\mathit{path}(x)) = (3, [(2,1),(1,0)]) \neq (2, [(2,1),(1,0)]) = x$.

As in the previous section, in order to prove that δ converges over increasing path algebras it is necessary to find a quantity that always decreases when applying F^{ep}. The key insight is that the set of consistent routes is finite and therefore it is possible to reuse the distance functions defined in Section 5.1 when calculating the distance between

$$\begin{tikzcd}
\text{Distance vector} & & & \text{Path vector} & \\
h(x) \arrow[dd] & = & h_c(x) \arrow[d] & & h_i(x) \arrow[d] \\
 & /\!\!/ & r_c(x,y) \arrow[dr] & & r_i(x,y) \arrow[dl] \\
r(x,y) \arrow[d] & & & r(x,y) \arrow[d] & \\
d(x,y) \arrow[d] & & & d(x,y) \arrow[d] & \\
D(\mathbf{X},\mathbf{Y}) & & & D(\mathbf{X},\mathbf{Y}) &
\end{tikzcd}$$

Figure 5.2: The structure of the distance functions in the paper.

two consistent states. Furthermore, as F^{ep} preserves consistency, then F^{ep} is strictly contracting over this distance function with respect to the set of consistent states. The remaining problem is to find a quantity that decreases when applying F^{ep} to a routing state that contains inconsistent routes. The solution is to consider the lengths of the inconsistent paths in the state. Let $s(\mathbf{X})$ be the length of the shortest inconsistent path in the inconsistent state $\mathbf{X}$.

Lemma 18. For all $\mathbf{X}$ such that $F^{ep}(\mathbf{X})$ is inconsistent then $s(\mathbf{X}) < s(F^{ep}(\mathbf{X}))$.

Proof. The contraposition of Lemma 17 implies that the shortest inconsistent route in $F^{ep}(\mathbf{X})$ must be an extension of some inconsistent route in $\mathbf{X}$. By (P3) the paths of the two routes must be of the form $(i, j) :: p$ and p respectively and by definition $s(\mathbf{X}) \leq |p|$. Hence $s(\mathbf{X}) \leq |p| < |(i, j) :: p| = s(F^{ep}(\mathbf{X}))$. □

One simple consequence of Lemma 18 is that, in the absence of further topology changes, eventually all inconsistent routes must be flushed from the state. This follows as if the length of the path of the shortest inconsistent route continues to increase then eventually all inconsistent routes must be of at least length n. Hence their path will necessarily contain a loop and therefore they will be flushed from the routing state as a consequence of assumption (P3). Hence after n applications of F^{ep} the state must be consistent.

By Lemma 18 then "$n - s(\mathbf{X})$" provides the required strictly decreasing quantity for inconsistent states. The two separate strictly decreasing quantities can then be combined to form a unified strictly decreasing distance function over both inconsistent and consistent routes. Figure 5.2 shows how all the distance functions fit together.

Inconsistent height of routes With the above discussion in mind, the *inconsistent height* of a route $h_i : S \to \mathbb{N}$ is defined as:

$$h_i(x) \triangleq \begin{cases} 1 & \text{if } x \in C_{\mathbf{A}^{ep}} \\ (n+1) - |path(x)| & \text{otherwise} \end{cases}$$

where n is the number of nodes in the network. All consistent routes have the minimum height 1, and the maximum height is $n + 1$ as the *path* function only returns simple paths. To explicitly highlight the parallels with the previous section, this maximum height will be called H_i. Therefore for all routes x then the following relationships hold:

$$1 = h_i(\overline{0}) \leq h_i(x) \leq n + 1 = H_i$$

Inconsistent distance function The inconsistent height can be used to define the inconsistent distance function $r_i : S \times S \to \mathbb{N}$ as follows:

$$r_i(x, y) \triangleq \max(h_i(x), h_i(y))$$

This will always be used to compare routes at least one of which is inconsistent. Therefore the inconsistent distance between two routes is proportional to the length of the shortest inconsistent path. Note this definition does not contain a check for equality as in practice it will never be used to compare two equal routes.

Consistent distance function As discussed, the set of routes consistent with the current network topology is finite. Consequently the previous metric r from Section 5.1 may be used to compare consistent routes. This will be renamed to $r_c : C_{\mathbf{A}^{ep}} \times C_{\mathbf{A}^{ep}} \to \mathbb{N}$. The maximum such consistent distance H will also be renamed to H_c.

Distance between routes Given both the consistent and inconsistent distance functions, then the overall distance function over routes $r : S \times S \to \mathbb{N}$ is defined as:

$$r(x, y) = \begin{cases} 0 & \text{if } x = y \\ r_c(x, y) & \text{if } x \neq y \text{ and } \{x, y\} \subseteq C_{\mathbf{A}^{ep}} \\ H_c + r_i(x, y) & \text{if } x \neq y \text{ and } \{x, y\} \nsubseteq C_{\mathbf{A}^{ep}} \end{cases}$$

If the two routes are equal then the distance between them is zero. If the two routes are not equal but both consistent then r_c is used to compute the distance between them. Otherwise at least one is inconsistent and r_i is used instead. The maximum consistent height H^c is added to $r_i(x, y)$ to later ensure that the required strictly contracting properties hold. In particular the fact that the distance between inconsistent routes is always greater than the distance between consistent routes guarantees that the distance is still strictly decreasing when the last inconsistent route is flushed from the routing state.

Distance function over routing tables As before, the distance function over routing tables can now be defined as:

$$d(x, y) = \max_{j} \; r(x_j, y_j)$$

The distance between two routing tables is therefore proportional to the length of the shortest inconsistent route, or, if no such route exists, the desirability of the best consistent route. As in Section 5.1 it is possible, but unnecessary, to show that d is an ultrametric.

Distance function over states Finally, as required by Theorem 6, the distance function over routing states, D^p, is defined as follows:

$$D^p(\mathbf{X}, \mathbf{Y}) = \max_{i \in p} \; d(\mathbf{X}_i, \mathbf{Y}_i)$$

where p is the set of participants.

Unlike the corresponding constructions in Section 5.1, the functions h_i, r_i, r_c, r, d and D^p all depend on the current topology $\mathbf{A}^{ep}$, and therefore the current epoch e and set of participants p. Hence if being strictly accurate they should be written as h_i^{ep}, r_i^{ep}, r_c^{ep}, r^{ep}, d^{ep} and D^{ep}. However for the sake of readability this dependency is left implicit.

As in the proof for distance-vector protocols, it will be shown that F^{ep} is an AMCO. However, unlike previously, F^{ep} is not strictly contracting with respect to D^p. Therefore it is is necessary to prove that F^{ep} is both strictly contracting on orbits and strictly contracting on the resulting fixed point separately. As before a couple of useful auxiliary lemmas are proved first.

Lemma 19. For all epochs e and participants p if $F^{ep}(\mathbf{X})_{ij}$ is inconsistent then there exists a node k such that $\mathbf{X}_{kj}$ is inconsistent and $\mathbf{X}_{kj} \neq F^{ep}(\mathbf{X})_{kj}$.

Proof. For clarity this proof omits the dependency of F and $\mathbf{A}$ on e and p. As the route $F(\mathbf{X})_{ij}$ is inconsistent it must be an extension of some inconsistent route in $\mathbf{X}$ by Lemma 17. Therefore there exists a node l such that $F(\mathbf{X})_{ij} = \mathbf{A}_{il}(\mathbf{X}_{lj})$ and $\mathbf{X}_{lj}$ is inconsistent.

If $\mathbf{X}_{lj} \neq F(\mathbf{X})_{lj}$ then l is the required node. Otherwise if $\mathbf{X}_{lj} = F(\mathbf{X})_{lj}$ then $F(\mathbf{X})_{lj}$ is inconsistent. Therefore the entire argument can be repeated with $F(\mathbf{X})_{lj}$. However as $F(\mathbf{X})_{ij} = \mathbf{A}_{ij}(\mathbf{X}_{lj}) = \mathbf{A}_{ij}(F(\mathbf{X})_{lj})$ the path of $F(\mathbf{X})_{lj}$ must be strictly shorter than the path of $F(\mathbf{X})_{ij}$. The length of the path cannot decrease indefinitely and therefore this argument must eventually terminate. □

The next lemma corresponds to Lemma 13 in the proof of convergence for distance-vector protocols and says that once again if the distance between all non-equal entries of two states is bounded above by v then the distance between any two entries after F^{ep} has been applied must be *strictly* less than v.

Lemma 20. For every state $\mathbf{X}$, nodes i and j, epoch e, set of participants p, and natural number $v > 0$, then:

$$(\forall k : \mathbf{X}_{kj} \neq F^{ep}(\mathbf{X})_{kj} \Rightarrow r(\mathbf{X}_{kj}, F^{ep}(\mathbf{X})_{kj}) \leq v) \Rightarrow r(F^{ep}(\mathbf{X})_{ij}, (F^{ep})^2(\mathbf{X})_{ij}) < v$$

Proof. For clarity this proof omits the dependency of F on e and p.
<u>Case 1</u>: $F(\mathbf{X})_{ij} = F^2(\mathbf{X})_{ij}$
Then the inequality is immediate as:

$$\begin{aligned} r(F(\mathbf{X})_{ij}, F^2(\mathbf{X})_{ij}) &= 0 && \text{(by Case 1 and def. of } r\text{)} \\ &< v && \text{(by lemma ass.)} \end{aligned}$$

<u>Case 2</u>: $F(\mathbf{X})_{ij} \neq F^2(\mathbf{X})_{ij}$ and $F(\mathbf{X})_{ij}$ and $F^2(\mathbf{X})_{ij}$ are both consistent.
<u>Case 2.1</u>: $\mathbf{X}$ is consistent
If $\mathbf{X}$ is consistent then $F(\mathbf{X})$ must also be consistent and so all routes involved are consistent. As the path algebra is (strictly) increasing then the required inequality is therefore immediate from Lemma 13 from the previous section.

<u>Case 2.2</u>: $\mathbf{X}$ is inconsistent
If $\mathbf{X}$ is inconsistent then there must exist nodes k and l such that $\mathbf{X}_{kl}$ is inconsistent and $\mathbf{X}_{kl} \neq F(\mathbf{X})_{kl}$. To see why, consider whether $F(\mathbf{X})$ is consistent. If $F(\mathbf{X})$ is consistent then any inconsistent entry in $\mathbf{X}$ will suffice. If $F(\mathbf{X})$ is inconsistent, then there exists an inconsistent entry $F(\mathbf{X})_{ml}$ in which case the required k may be obtained from Lemma 19. The inequality then follows as:

$$\begin{aligned} r(F(\mathbf{X})_{ij}, F^2(\mathbf{X})_{ij}) &= r_c(F(\mathbf{X})_{ij}, F^2(\mathbf{X})_{ij}) && \text{(by Case 2 and def. of } r\text{)} \\ &< H_c + r_i(\mathbf{X}_{kl}, F(\mathbf{X})_{kl}) && \text{(as } r_c \text{ is bounded by } H_c\text{)} \\ &= r(\mathbf{X}_{kl}, F(\mathbf{X})_{kl}) && \text{(by def. of } r \text{ and } \mathbf{X}_{kl} \neq F(\mathbf{X})_{kl}\text{)} \\ &\leq v && \text{(by lemma ass. and } \mathbf{X}_{kl} \neq F(\mathbf{X})_{kl}\text{)} \end{aligned}$$

<u>Case 3</u>: $F(\mathbf{X})_{ij} \neq F^2(\mathbf{X})_{ij}$ and $F(\mathbf{X})_{ij}$ or $F^2(\mathbf{X})_{ij}$ is inconsistent.
As either $F(\mathbf{X})$ or $F^2(\mathbf{X})$ is inconsistent then $\mathbf{X}$ must be inconsistent as well. Let $\mathbf{X}_{kl}$ be

the shortest inconsistent route in $\mathbf{X}$. As all inconsistent routes in $F(\mathbf{X})$ and $F^2(\mathbf{X})$ are an extension of inconsistent routes in $\mathbf{X}$. Hence the path of $\mathbf{X}_{kl}$ must be shorter than the path of all inconsistent routes in $F(\mathbf{X})$ and $F^2(\mathbf{X})$ and so $r_i(F(\mathbf{X})_{ij}, F^2(\mathbf{X})_{ij}) < h_i(\mathbf{X}_{kl})$ and $\mathbf{X}_{kl} \neq F(\mathbf{X})_{kl}$. The inequality then follows as:

$$\begin{aligned} r(F(\mathbf{X})_{ij}, F^2(\mathbf{X})_{ij}) &= H_c + r_i(F(\mathbf{X})_{ij}, F^2(\mathbf{X})_{ij}) && \text{(by Case 2 and def. of } r\text{)} \\ &< H_c + h_i(\mathbf{X}_{kl}) && \text{(by argument above)} \\ &\leq H_c + \max(h_i(\mathbf{X}_{kl}), h_i(F(\mathbf{X})_{kl})) && \text{(by def. of max)} \\ &= r(\mathbf{X}_{kl}, F(\mathbf{X})_{kl}) && \text{(by def. of } r \text{ and } \mathbf{X}_{kl} \neq F(\mathbf{X})_{kl}\text{)} \\ &\leq v && \text{(by lem. ass. and } \mathbf{X}_{kl} \neq F(\mathbf{X})_{kl}\text{)} \end{aligned}$$

Hence the required inequality holds in all cases. □

The required strictly contracting on orbits and strictly contracting on fixed point properties can now be proved using the above lemma.

Lemma 21. For all epochs e and participants p the function F^{ep} is strictly contracting on orbits with respect to D^p over A_p.

Proof. Consider an arbitrary epoch e and set of participants p and state $\mathbf{X} \in A_p$. Then it must be shown that if $\mathbf{X} \neq F^{ep}(\mathbf{X})$ then:

$$D^p(F^{ep}(\mathbf{X}), (F^{ep})^2(\mathbf{X})) < D^p(\mathbf{X}, F^{ep}(\mathbf{X}))$$

As $D^p(F^{ep}(\mathbf{X}), (F^{ep})^2(\mathbf{X})) = \max_{i \in p, j \in V} r(F^{ep}(\mathbf{X})_{ij}, (F^{ep})^2(\mathbf{X})_{ij})$ it suffices to show for all nodes $i \in p$ and $j \in V$ that

$$r(F^{ep}(\mathbf{X})_{ij}, (F^{ep})^2(\mathbf{X})_{ij}) < D^p(\mathbf{X}, F^{ep}(\mathbf{X}))$$

This can be proved by applying Lemma 20 where $v = D^p(\mathbf{X}, F^{ep}(\mathbf{X}))$. However to apply the lemma it must be verified that $0 < D^p(\mathbf{X}, F^{ep}(\mathbf{X}))$ and that for all nodes k then $\mathbf{X}_{kj} \neq \mathbf{Y}_{kj} \Rightarrow r(\mathbf{X}_{kj}, F^{ep}(\mathbf{X})_{kj}) \leq D^p(\mathbf{X}, F^{ep}(\mathbf{X}))$.

The former holds as $\mathbf{X} \neq F^{ep}(\mathbf{X})$ and so there must exist entries in $\mathbf{X}$ and $F^{ep}(\mathbf{X})$ which are a non-zero distance apart. To show that the latter holds consider whether or not node k is participating. If $k \in p$ then the required inequality holds simply from the definition of D^p. If $k \notin p$ then, as $\mathbf{X}$ and $F^{ep}(\mathbf{X})$ are accordant, $\mathbf{X}_{kj} = \overline{\infty} = F^{ep}(\mathbf{X})_{kj}$ and therefore the inequality holds trivially as $r(\mathbf{X}_{kj}, F^{ep}(\mathbf{X})_{kj}) = 0$. □

Lemma 22. If the path algebra is increasing then σ is contracting on its fixed point $\mathbf{X}^*$.

Proof. The proof has the same structure as Lemmas 20 & 21. The only major difference is that some of the cases in the corresponding version of Lemma 20 are redundant as the fixed point $\mathbf{X}^*$ is guaranteed to be consistent. This must be the case, as if $\mathbf{X}^*$ was inconsistent then applying F^{ep} would increase the length of the shortest inconsistent path. Interested readers can find the remaining details in the Agda formalisation [19]. □

Theorem 8. Given an increasing path algebra $(S,\ \oplus,\ E,\ \overline{\infty},\ \overline{0},\ path)$ then δ is deterministically convergent.

Proof. First it will be shown that F is a dynamic AMCO. Let $d_i^{ep} = d$ for all nodes i, epochs e and set of participants p. Then the conditions are fulfilled as:

(DU1) d is a quasi-semi-metric

The required property $\forall x, y : d(x, y) = 0 \Leftrightarrow x = y$ is immediate from the definition of d and r.

(DU2) d is bounded

The maximum distance r, and therefore d, can return is $H^c + n$.

(DU3) $\forall e, p$: F^{ep} is strictly contracting on orbits w.r.t. D^{ep} over A_p

Proved in Lemma 21.

(DU4) $\forall e, p$: F^{ep} is strictly contracting on $\mathbf{X}^*$ w.r.t. D^{ep} over A_p for any fixed point $\mathbf{X}^*$

Proved in Lemma 22.

(DU5) $\forall e, p, \mathbf{X} : F^{ep}(\mathbf{X}) \in A_p$.

By the definition of F^{ep}.

As F is an AMCO then Theorem 6 may be applied and so δ is deterministically convergent over $(S,\ \oplus,\ E,\ \overline{\infty},\ \overline{0},\ path)$. □

5.3 Conclusions

This chapter has presented the first proof that finite, strictly increasing distance-vector protocols are deterministically convergent. In particular this shows that RIP-like distance-vector protocols can include BGP-like conditional policy. Secondly it has presented a new proof that (strictly) increasing path-vector protocols are deterministically convergent. As discussed, this proof has many advantages over the previous proof by Sobrinho including: guaranteeing deterministic convergence, capturing protocols with path-dependent conditional policy and not requiring an assumption of in-order reliable delivery of messages.

Chapter 6

Rate of convergence

The previous chapter proved the correctness of both finite, strictly increasing distance-vector protocols and increasing path-vector protocols. However, as with any algorithm, correctness is not the only consideration. It is easy to show that in the worst case the rate of convergence for finite, strictly increasing routing algebras is $\Theta(|S|)$. However the rate of convergence for increasing path algebras is only known to lie between $\Omega(n)$ and $O(n!)$.

This chapter addresses this gap in the theory by first demonstrating an increasing path algebra and a family of graphs which require $\Theta(n^2)$ iterations to converge hence raising the lower bound to $\Omega(n^2)$. Secondly it presents a new proof of convergence that guarantees that convergence will occur in at most $n^2 - n + 1$ iterations hence lowering the upper bound to $O(n^2)$. Together these results provide a new tight bound for the worst-case of $\Theta(n^2)$.

Note that while the upper bound of $O(n^2)$ has been formalised in Agda, the lower bound of $\Omega(n^2)$ is the only result in this thesis that has not been formalised. This is because a) it is time consuming to formally reason about the entire execution trace of an abstract family of graphs, b) being a lower bound there are less severe practical consequences if it turns out to be incorrect and c) execution traces that exhibit convincing quadratic behaviour have been generated using Python scripts.

6.1 Measuring the rate of convergence

Measuring the rate of convergence of asynchronous processes is always inherently tricky. While it might be tempting to count the number of "events" that occur (e.g. messages sent/received, routing table entries updated), this measure suffers from several problems. Firstly, many of these events happen in parallel and therefore it is an inaccurate measure for the actual time required to converge. Secondly it has been shown that even the simplest routing problems, such as shortest-paths, may require an exponential number of events to occur in the worst case [40]. Consequently, it is hard to make comparisons between the different classes of algebras as everything requires an exponential number of events.

Instead this chapter measures the number of *synchronous* iterations required for convergence i.e. the first k such that $\sigma^{k+1}(\mathbf{X}) = \sigma^k(\mathbf{X})$ for some starting state $\mathbf{X}$. This is equivalent to assuming that all nodes update and broadcast their routing tables synchronously.

Why is this the right thing to measure? Firstly it separates out the complexity inherent to the routing algebra from the complexity introduced by the asynchronous nature of the computation. Secondly, Lemma 9 in Chapter 3 showed that the number of synchronous iterations is proportional to the number of pseudocycles required for the convergence of asynchronous iteration. The average length of a pseudocycle is dependent on the update frequency and the delay and loss rates of the links. Several papers have used this relationship to construct probabilistic models for the convergence time of asynchronous iterations [14, 65].

One subtlety to note is that, as a routing protocol is an online algorithm, the rate of *reconvergence* is much more interesting than the rate of convergence from some idealised starting state. Consequently in the following analysis the synchronous iteration will be assumed to start from some arbitrary state $\mathbf{X}$ rather than the initial state $\mathbf{I}$.

6.2 Previous work

Table 6.1 shows the existing state of knowledge for convergence bounds.

	Distance-vector	Path-vector
Distributive	$\Theta(\lvert S\rvert)$	$\Theta(n)$
Strictly increasing	$\Theta(\lvert S\rvert)$	$\Omega(n) \leq ??? \leq O(n!)$

Table 6.1: Existing knowledge about the number of synchronous iterations required for convergence in the worst-case. S is the set of path-weights and n is the size of the network.

Distance-vector protocols For distance-vector protocols it is easy to show that regardless of whether or not the algebra is distributive or strictly increasing the worst-case rate of convergence is $\Theta(|S|)$ where $|S|$ is the size of the set of path weights. The lower bound comes from the example of count-to-convergence shown in Figure 2.5 in Section 2.2.2.3. The upper bound is a direct result of the proof in Chapter 5. This result is the main reason why distance-vector protocols are generally considered to be impractical without either artificially limiting $|S|$ or the addition of some mechanism of preventing count-to-convergence. For example RIP artificially limits $|S|$ to 16 (see Section 2.3.1).

Path-vector protocols Path-vector protocols scale much better than distance-vector protocols and in the distributive world are guaranteed to converge in $\Theta(n)$ iterations where n is the size of the network. However the corresponding result for increasing[1] path-vector routing problems is still unknown. It is trivial to obtain a lower bound of $\Omega(n)$ for the worst-case by considering a linear network. Likewise an upper bound of $O(n!)$ is easily obtainable from Lemma 21: every iteration the distance between states decreases by at least 1 and the maximum distance is proportional to the number of consistent routes – i.e. the number of paths in the current network topology.

Clearly if in the worst case $n!$ iterations are actually required, then the time taken for the protocol to converge is infeasible in all but the smallest networks. Consequently any vector-based protocol based on either strictly-increasing algebras or free networks would be of little use in the general case. However, in practice it is surprisingly difficult to construct an example that does not converge in linear time. This provides circumstantial evidence that the worst case is unlikely to be factorial.

6.3 Lower bound: $\Omega(n)$ to $\Omega(n^2)$

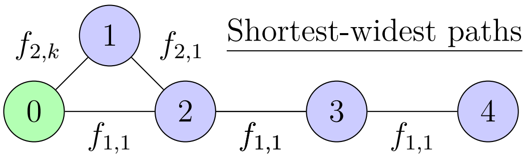
This section will construct a (strictly) increasing path algebra and a family of network configurations Q_n that take $\Theta(n^2)$ iterations to converge, hence increasing the lower bound for the worst-case from $\Omega(n)$ to $\Omega(n^2)$. For simplicity this section only considers the routing decisions for a single fixed destination, node 0, and therefore ignores all other parts of the global routing state. Additionally, for notational brevity, paths are represented as sequences of nodes, rather the sequences of edges used elsewhere in the thesis.

6.3.1 Count-to-convergence induced by non-distributivity

In order to construct a quadratic example it is first necessary to understand the relationship between violations of distributivity and the phenomenon of count-to-convergence. As described in Section 2.2.2.3, count-to-convergence is known to afflict distance-vector routing protocols. In distributive algebras it only occurs after changes to the network topology when nodes continue to exchange "junk routes" that are inconsistent with new topology. However in non-distributive algebras these junk routes may be generated even in the absence of changes to the network topology and so count-to-convergence may occur at any point in the computation.

Figure 6.1 uses the non-distributive shortest-widest paths algebra ($\mathbb{SWP}$) described in Section 4.4.2 to demonstrate this. In particular it uses the example violation of distributivity described in Section 4.5.3. Assume that k is some large even number.

[1] Recall that there is no difference between an increasing and a strictly increasing path algebra.



Time	0	1	2	3	4
0	$(\infty, \mathbf{0})$	$(0, \infty)$	$(0, \infty)$	$(0, \infty)$	$(0, \infty)$
1	-	$(\mathbf{2}, \mathbf{k})$	$(1, 1)$	$(0, \infty)$	$(0, \infty)$
2	-	-	$(\mathbf{2}, \mathbf{k{+}1})$	$(1, 2)$	$(0, \infty)$
3	-	-	-	$(1, k{+}2)$	$(1, 3)$
4	-	-	-	$(1, 4)$	$(1, k{+}3)$
5	-	-	-	$(1, k{+}2)$	$(1, 5)$
6	-	-	-	$(1, 6)$	$(1, k{+}3)$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
k	-	-	-	$(1, k)$	$(1, k{+}3)$
$k{+}1$	-	-	-	$(\mathbf{1}, \mathbf{k{+}2})$	$(1, k{+}1)$
$k{+}2$	-	-	-	-	$(\mathbf{1}, \mathbf{k{+}3})$

Figure 6.1: Using a distributivity violation to induce count-to-convergence in the shortest-widest-paths algebra. Green/red cells indicate that the route has got better/worse since the last iteration. Bold indicates that the node has converged.

Nodes 1 and 2 form what will be referred to as a *latch*. At time 1 node 1 *opens* the latch by adopting the route generated along the path $[1, 0]$. At the same time node 2 *loads* the route generated along path $[2, 0]$. Then at time 2 node 2 *closes* the latch by adopting the higher bandwidth route along path $[2, 1, 0]$. But at the same time node 3 adopts the junk route through $[3, 2, 0]$ that it prefers due to the previously described distributivity violation. After time 2 the junk route (and its extensions) are exchanged back-and-forth between nodes 3 and 4. Finally, convergence is reached at time $k + 2$.

Path-vector protocols are designed to solve the count-to-convergence problem by prohibiting nodes from adopting routes with a path containing themselves. For example at time 4 the algebra $\mathbb{P}(\mathbb{SWP})$ would prevent node 3 from extending node 4's junk route as the resulting path $[3, 4, 3, 2, 1, 0]$ would have a loop in it. Hence whereas $\mathbb{SWP}$ requires $k + 2$ iterations to converge in this example, $\mathbb{P}(\mathbb{SWP})$ only requires 4 iterations.

Although path-vector protocols succeed in preventing count-to-convergence by guaranteeing that a junk routes will be removed after n iterations, this does not immediately solve the rate of convergence question. As strictly increasing protocols can generate junk routes at any point in the iteration, it is conceivable that each junk route could trigger an exponentially-growing cascade of further junk routes. The question is how long can such junk continue to be generated?

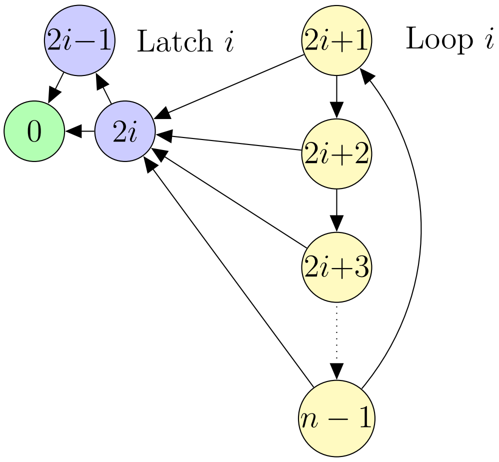


Figure 6.2: The i^{th} gadget in Q_n. Junk routes are generated by the latch and then continue to circulate for $n - 2i - 1$ iterations in the loop.

6.3.2 An example of $O(n^2)$ convergence

This section uses the concept of the latch described in Figure 6.1 to construct a family of network configurations $\{Q_n \mid n \in \mathbb{N}\}$ designed specifically to repeatedly generate and circulate junk routes as long as possible. The network Q_n has n nodes labelled 0 to $n - 1$. The nodes are connected in such a way as to form m *gadgets* where $m = \lfloor \frac{n-1}{2} \rfloor$. The i^{th} gadget contains a latch formed by the nodes $\{2i - 1, 2i\}$ and a loop formed by the nodes $\{2i + 1, \cdots, n - 1\}$. Figure 6.2 illustrates the i^{th} gadget. Each gadget is constructed so that the junk generated by its latch circulates for as long as possible in its loop.

The operation of a gadget will now be described in detail. Let t be the activation time of gadget i. At this time all the nodes from 1 to $2i - 1$ are assumed to have converged and latch i has just been opened and loaded. That is node $2i - 1$ has opened latch i by accepting the route generated along the path $[2i - 1, 0]$ and node $2i$ has loaded the latch by accepting the route generated along the path $[2i, 0]$. Epoch i then continues:

- At time $t + 1$ node $2i$ closes the latch by adopting the path $[2i, 2i - 1, 0]$. At the same time all the nodes in loop i extend node $2i$'s route with the path $[2i, 0]$.

- At time $t + 2$ latch i is closed. Node $2i$ is advertising the route generated along the path $[2i, 2i - 1, 0]$ but due to a distributivity violation, all the nodes in loop i prefer the extensions of the junk route of the path $[2i, 0]$ that was advertised at time $t + 1$. Each node in loop i will accept the extension of the junk route through its counter-clockwise neighbour. For example, node $2i + 1$ takes the path $[2i + 1, 2i + 2, 2i, 0]$, while node $n - 1$ takes the path $[n - 1, 2i + 1, 2k, 0]$.

- This process continues with each node in loop i repeatedly accepting the junk routes advertised by its counter-clockwise neighbour.

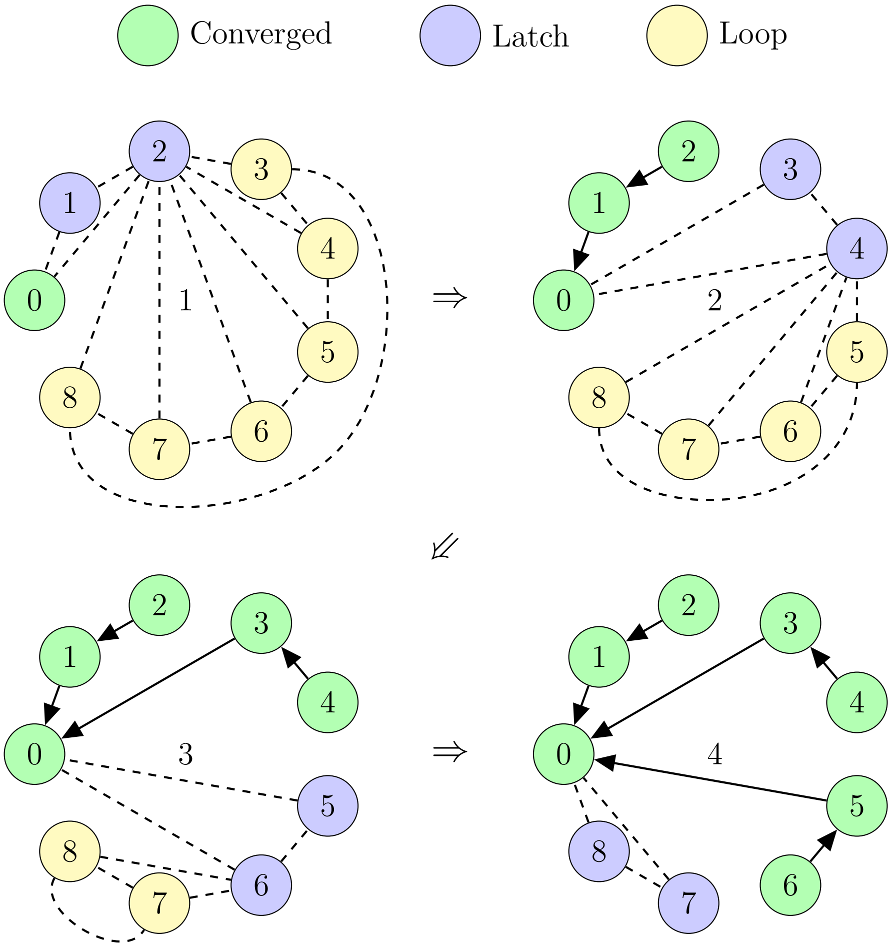


Figure 6.3: The structure of Q_9 consisting of 4 gadgets. Each gadget takes $\Theta(n)$ iterations to flush the junk route generated by the latch from the loop before triggering the next gadget. Dashed lines are the links forming the current gadget, while solid lines are the final next-hop choices after the nodes have converged.

- At time $t+n-2i-1$, it is no longer possible for this to continue as the path of the junk routes will contain every node in loop i. For example, node $2i+1$ takes the path $[2i+1, 2i+2, 2i+3, \cdots, n-1, 2k, 0]$, while node $n-1$ takes the path $[n-1, 2i+1, 2i+2, \cdots n-2, 2k, 0]$.
- Therefore at time $t+n-2i$, these junk routes will be discarded and $i+1^{th}$ gadget will be initialised.

Each gadget is designed so that flushing the junk route from its loop, opens the latch of the next gadget. The overall structure can be seen in Figure 6.3 which shows the four latches that comprise Q_9.

Why does Q_n require $\Theta(n^2)$ iterations to converge? The distributivity violation of latch i generates junk routes which propagate around loop i for $\Theta(n)$ iterations. When these junk routes are finally flushed from loop i, they trigger latch $i+1$. Quadratic converge is then achieved because there are $\Theta(n)$ such gadgets.

More formally let $T(n)$ be the convergence time of Q_n. Consider the general case for Q_{n+2}, and the time taken for the 1st gadget to finish executing. It takes 1 iteration for the junk route to escape the 1st latch, and $n-1$ iterations for the junk route to then be flushed from the 1st loop, giving a total of n iterations. Nodes 1 & 2 will then have converged. After removing them from the graph and relabelling the remaining nodes, the new graph is exactly Q_n. Hence it is possible to formulate a recurrence relation for $T(n)$ as:

$$T(3) = 2, \qquad T(4) = 3, \qquad T(n+2) = n + T(n).$$

This is clearly quadratic by inspection. Due to parity issues, solving it is messy, but it can be simplified to:

$$T(n) = \frac{1}{4}n^2 - \frac{1}{2}n + c$$

where $c = 1$ if n is even and $c = \frac{5}{4}$ when n is odd.

The next question is whether this scheme is realisable within a strictly increasing algebra? The answer is yes, but only if the algebra is suitably expressive. The algebra that will be used is the stratified shortest paths algebra, $\mathbb{SSP}$, described in Section 4.4.3. This section does not provide a truly general description of the policies for each of the links in Q_n as they would be unintelligible and hence provide little insight.

Instead Figure 6.4 describes the polices required for just one member of the family, Q_9. It lists the entries of a matrix $\mathbf{B}_{ij}$ from which it is possible to construct Q_9's adjacency matrix $\mathbf{A}_{ij}$ as follows. If the figure does not contain an entry for $\mathbf{B}_{ij}$, then $\mathbf{A}_{ij} = f_{\infty'}$. Otherwise, if $\mathbf{B}_{ij} = f$ then $\mathbf{A}_{ij} = g_f$ as defined in Section 4.4.3. Note that for all edges $(i, 0)$ the policy $\mathbf{B}_{i0} = \langle i-1 \rangle$ ensures that the latch of the next gadget cannot activate before the previous gadget has finished executing.

Figure 6.5 presents a full execution trace for the 17 iterations required for the convergence of Q_9. It is hopefully clear that having constructed the example for Q_9 that it is possible to generalise the policies to Q_n in the obvious way.

6.4 Upper bound: $O(n!)$ to $O(n^2)$

The previous section explored an example of a strictly increasing path algebra which converged in $\Theta(n^2)$ iterations. Do there exist even more pathological algebras out there? It turns out that the answer to this question is no. This section will present a proof that every (strictly) increasing path algebra converges in at most n^2 iterations.

The proof is in some sense similar to the corresponding proofs for distributive path algebras (e.g. [16]). At each step the proof considers the set of nodes that have already converged. Next it identifies the node with the best path into the set of converged nodes.

i	j	Policy	Implements
1	0	$\langle 0 \rangle$	opens latch 1
2	0	$\langle 1 \rangle$	loads latch 1
2	1	$\langle 0 \rangle$	closes latch 1
3	0	$\langle 2 \rangle$	opens latch 2
3	2	$\langle \infty, 1 \rangle$	distributivity violation 1
3	4	$\langle \infty, 1 \rangle$	loop 1
4	0	$\langle 3 \rangle$	loads latch 2
4	2	$\langle \infty, 1 \rangle$	distributivity violation 1
4	3	$\langle \infty, \infty, 2 \rangle$	closes latch 2
4	5	$\langle \infty, 1 \rangle$	loop 1
5	0	$\langle 4 \rangle$	opens latch 3
5	2	$\langle \infty, 1 \rangle$	distributivity violation 1
5	4	$\langle \infty, \infty, \infty, 3 \rangle$	distributivity violation 2
5	6	$\langle \infty, 1, \infty, 3 \rangle$	loops 1, 2
6	0	$\langle 5 \rangle$	loads latch 3
6	2	$\langle \infty, 1 \rangle$	distributivity violation 1
6	4	$\langle \infty, \infty, \infty, 3 \rangle$	distributivity violation 2
6	5	$\langle \infty, \infty, \infty, \infty, 4 \rangle$	closes latch 3
6	7	$\langle \infty, 1, \infty, 3 \rangle$	loops 1, 2
7	0	$\langle 6 \rangle$	opens latch 4
7	2	$\langle \infty, 1 \rangle$	distributivity violation 1
7	4	$\langle \infty, \infty, \infty, 3 \rangle$	distributivity violation 2
7	6	$\langle \infty, \infty, \infty, \infty, \infty, 5 \rangle$	distributivity violation 3
7	8	$\langle \infty, 1, \infty, 3, \infty, 5 \rangle$	loops 1,2,3
8	0	$\langle 7 \rangle$	loads latch 4
8	2	$\langle \infty, 1 \rangle$	distributivity violation 1
8	3	$\langle \infty, 1 \rangle$	loop 1
8	4	$\langle \infty, \infty, \infty, 3 \rangle$	distributivity violation 2
8	5	$\langle \infty, \infty, \infty, 3 \rangle$	loop 2
8	6	$\langle \infty, \infty, \infty, \infty, \infty, 5 \rangle$	distributivity violation 3
8	7	$\langle \infty, \infty, \infty, \infty, \infty, 5, 6 \rangle$	loop 3 and closes latch 4

Figure 6.4: Constructing the adjacency matrix of Q_9. This lists the entries of $\mathbf{B}_{ij}$. If $\mathbf{B}_{ij}$ is not listed, then $\mathbf{A}_{ij} = f_{\infty_p}$. Otherwise, if $\mathbf{B}_{ij} = f$ then $\mathbf{A}_{ij} = g_f$ as defined in Section 4.4.3. The annotation "distributivity violation i" means that the policy implements one of the distributivity violations in gadget i and "loop i" means that the policy propagates the junk route around in loop i.

Time	Node	Level	Path	Event
1	1	**0**	**1, 0**	open latch 1
1	2	1	2, 0	load latch 1
1	3	2	3, 0	
1	4	3	4, 0	
1	5	4	5, 0	
1	6	5	6, 0	
1	7	6	7, 0	
1	8	7	8, 0	
2	2	**0**	**2, 1, 0**	close latch 1
2	3	1	3, 2, 0	violation 1
2	4	1	4, 2, 0	violation 1
2	5	1	5, 2, 0	violation 1
2	6	1	6, 2, 0	violation 1
2	7	1	7, 2, 0	violation 1
2	8	1	8, 2, 0	violation 1
3	3	1	3, 4, 2, 0	junk circulates in loop 1
3	4	1	4, 5, 2, 0	
3	5	1	5, 6, 2, 0	
3	6	1	6, 7, 2, 0	
3	7	1	7, 8, 2, 0	
3	8	1	8, 3, 2, 0	
4	3	1	3, 4, 5, 2, 0	junk circulates in loop 1
4	4	1	4, 5, 6, 2, 0	
4	5	1	5, 6, 7, 2, 0	
4	6	1	6, 7, 8, 2, 0	
4	7	1	7, 8, 3, 2, 0	
4	8	1	8, 3, 4, 2, 0	
5	3	1	3, 4, 5, 6, 2, 0	junk circulates in loop 1
5	4	1	4, 5, 6, 7, 2, 0	
5	5	1	5, 6, 7, 8, 2, 0	
5	6	1	6, 7, 8, 3, 2, 0	
5	7	1	7, 8, 3, 4, 2, 0	
5	8	1	8, 3, 4, 5, 2, 0	
6	3	1	3, 4, 5, 6, 7, 2, 0	junk circulates in loop 1
6	4	1	4, 5, 6, 7, 8, 2, 0	
6	5	1	5, 6, 7, 8, 3, 2, 0	
6	6	1	6, 7, 8, 3, 4, 2, 0	
6	7	1	7, 8, 3, 4, 5, 2, 0	
6	8	1	8, 3, 4, 5, 6, 2, 0	

Figure 6.5: The execution trace for σ in Q_9. Green cells indicate that the route has improved. Red cells indicate the route has gotten worse. The step in which a node has converged is shown in **bold**, and after that the node's state is no longer listed. In the initial state at time 0 (not listed) only node 0 has converged and all other nodes have routes ∞_p. The notation "violation" stands for distributivity violation.

Time	Node	Level	Path	Event
7	3	1	$3, 4, 5, 6, 7, 8, 2, 0$	
7	4	1	$4, 5, 6, 7, 8, 3, 2, 0$	
7	5	1	$5, 6, 7, 8, 3, 4, 2, 0$	junk circulates
7	6	1	$6, 7, 8, 3, 4, 5, 2, 0$	in loop 1
7	7	1	$7, 8, 3, 4, 5, 6, 2, 0$	
7	8	1	$8, 3, 4, 5, 6, 7, 2, 0$	
8	3	**2**	$\mathbf{3, 0}$	open latch 2
8	4	3	$4, 0$	load latch 2
8	5	4	$5, 0$	
8	6	5	$6, 0$	
8	7	6	$7, 0$	
8	8	7	$8, 0$	
9	4	**2**	$\mathbf{4, 3, 0}$	close latch 2
9	5	3	$5, 4, 0$	violation 2
9	6	3	$6, 4, 0$	violation 2
9	7	3	$7, 4, 0$	violation 2
9	8	3	$8, 4, 0$	violation 2
10	5	3	$5, 6, 4, 0$	
10	6	3	$6, 7, 4, 0$	junk circulates
10	7	3	$7, 8, 4, 0$	in loop 2
10	8	3	$8, 5, 4, 0$	
11	5	3	$5, 6, 7, 4, 0$	
11	6	3	$6, 7, 8, 4, 0$	junk circulates
11	7	3	$7, 8, 5, 4, 0$	in loop 2
11	8	3	$8, 5, 6, 4, 0$	
12	5	3	$5, 6, 7, 8, 4, 0$	
12	6	3	$6, 7, 8, 5, 4, 0$	junk circulates
12	7	3	$7, 8, 5, 6, 4, 0$	in loop 2
12	8	3	$8, 5, 6, 7, 4, 0$	
13	5	**4**	$\mathbf{5, 0}$	open latch 3
13	6	5	$6, 0$	load latch 3
13	7	6	$7, 0$	
13	8	7	$8, 0$	
14	6	**4**	$\mathbf{6, 5, 0}$	close latch 3
14	7	5	$7, 6, 0$	violation 3
14	8	5	$8, 6, 0$	violation 3
15	7	5	$7, 8, 6, 0$	junk circulates
15	8	5	$8, 7, 6, 0$	in loop 3
16	7	**6**	$\mathbf{7, 0}$	open latch 4
16	8	7	$8, 0$	load latch 4
17	8	**6**	$\mathbf{8, 7, 0}$	close latch 4

Figure 6.5: The execution trace for σ in Q_9. Green cells indicate that the route has improved. Red cells indicate the route has gotten worse. The step in which a node has converged is shown in **bold**, and after that the node's state is no longer listed. In the initial state at time 0 (not listed) only node 0 has converged and all other nodes have routes ∞_p. The notation "violation" stands for distributivity violation.

It then proves that after an additional k iterations this node will have converged. This process can be repeated for each node giving a total of $k(n-1)$ iterations.

For distributive algebras it can be shown that $k = 1$, resulting in a total of $n-1$ iterations. However in an increasing, non-distributive path algebra a single iteration may not suffice because of the junk routes generated by distributivity violations. The key insight is that junk routes capable of interfering with the convergence of the node with the best path into the converged set, can only persist for at most n iterations before necessarily containing a loop and hence being flushed. Hence $k = n$ for strictly increasing algebras, resulting in a total of $n^2 - n + 1$ iterations required for convergence in the worst case.

6.4.1 Additional definitions

Some of the terms such as "junk routes" used informally in previous sections will now be rigorously defined. Let $(S,\ \oplus,\ E,\ \overline{\infty},\ \overline{0},\ \mathit{path})$ be an increasing path algebra and consider a network of n nodes represented by an adjacency matrix $\mathbf{A}$. Let $\mathbf{X}$ be the initial state and without loss of generality let j be the destination node.

The first concept that needs to be defined is that of a single node having converged. A first attempt might be to say that a node is *fixed at time t* if it never again changes its current route.

> **Definition 36.** A node i is *fixed at time t* iff
>
> $$\forall s \geq t : \sigma^s(\mathbf{X})_{ij} = \sigma^t(\mathbf{X})_{ij}$$
>
> where $\mathcal{F}_t$ is the set of fixed nodes at time t.

As shown by the following lemmas, this definition has many desirable properties: the set of fixed nodes is monotonically increasing over time, the source node is fixed at time 1 and that both the path and the path-weight of a fixed node are aligned.

Lemma 23. $s \leq t \Rightarrow \mathcal{F}_s \subseteq \mathcal{F}_t$.

Proof. Immediate by the definition of $\mathcal{F}_t$ and the transitivity of $\leq$. □

Lemma 24. $j \in \mathcal{F}_1$.

Proof. Consider a time $s \geq 1$. Then the required result:

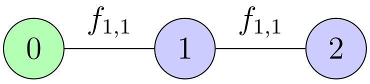
$$\sigma(\mathbf{X})_{jj} = F(\mathbf{X})_{jj} = \overline{0} = F(\sigma^{s-1}(\mathbf{X}))_{jj} = \sigma^s(\mathbf{X})_{jj}$$

is immediate, with the middle two equalities holding by Lemma 11 in Chapter 4 □

Lemma 25. If $path(\sigma^t(\mathbf{X})_{ij}) = (i,k) :: p$ and $i \in \mathcal{F}_t$ then $\sigma^t(\mathbf{X})_{ij} = \mathbf{A}_{ik}((\sigma^t(\mathbf{X})_{kj})$ and $p = path(\sigma^t(\mathbf{X})_{kj})$.

Proof. As $i \in \mathcal{F}_t$ it must be the case that $\sigma^t(\mathbf{X})_{ij} = \sigma^{t+1}(\mathbf{X})_{ij}$ and so $path(\sigma^{t+1}(\mathbf{X})_{ij}) = (i,k) :: p$. Assumption (P3) from the definition of a path algebra therefore implies that $\sigma^{t+1}(\mathbf{X})_{ij} = \mathbf{A}_{ik}(\sigma^t(\mathbf{X})_{kj})$ and that $p = path(\sigma^t(\mathbf{X})_{kj})$. Again as $i \in \mathcal{F}_t$, it must be case that $\sigma^t(\mathbf{X})_{ij} = \mathbf{A}_{ik}(\sigma^t(\mathbf{X})_{kj})$. □

However $\mathcal{F}_t$ lacks one important property. Perhaps counter-intuitively, it is not the case that all nodes in the path of a fixed node are necessarily fixed themselves. Therefore the set of fixed nodes in $\mathcal{F}_t$ do not form a tree rooted at the source node j. This is a problem, as some of the later proofs (in particular Lemma 30) will need to perform induction over the tree of converged nodes.



Time	Node 0	Node 1	Node 2
0	$(\infty, 0, [\,])$	$(2, 1, [(1,0)])$	$(1, 2, [(2,1), (1,0)])$
1	$(\infty, 0, [\,])$	$(1, 1, [(1,0)])$	$(1, 2, [(2,1), (1,0)])$
2	$(\infty, 0, [\,])$	$(1, 1, [(1,0)])$	$(1, 2, [(2,1), (1,0)])$

Figure 6.6: Example of an unfixed node on the path of a fixed node for the shortest-widest-paths algebra. The weight of the $(1,0)$ arc has just changed from $f_{2,1}$ to $f_{1,1}$, i.e. the bandwidth of the link has decreased by 1.

The underlying cause of this problem is that the iteration is starting from an arbitrary state $\mathbf{X}$ rather than from the initial state $\mathbf{I}$. For example consider Fig. 6.6 where the bandwidth of arc $(1,0)$ has decreased from 2 to 1. Node 1 initially contains an inconsistent route, but then adopts the new valid route at time 1. Hence node 1 is not in $\mathcal{F}_0$. However the bandwidth of the arc $(2,1)$ acts as a bottleneck, and therefore node 2 never notices the change in the bandwidth of node 1's route. Hence node 2 is in $\mathcal{F}_0$ but node 1 is not.

Instead a stronger notion of convergence is required. Namely a node has converged if all the nodes in its path to the source are fixed (this includes the node itself).

Definition 37. A node i has *converged* at time t iff:

$$\forall k \in path(\sigma^t(\mathbf{X})_{ij}) : k \in \mathcal{F}_t$$

where $\mathcal{C}_t$ is the set of converged nodes at time t.

Again the set of converged nodes obeys the same desirable properties as the set of fixed nodes, but also additionally fulfils the property that all of the nodes in the path of a converged node have themselves converged.

Lemma 26. $s \leq t \Rightarrow \mathcal{C}_s \subseteq \mathcal{C}_t$.

Proof. Immediate by the definition of $\mathcal{C}_t$ and Lemma 23. □

Lemma 27. $j \in \mathcal{C}_1$.

Proof. $\sigma^1(\mathbf{X})_{jj} = F(\mathbf{X})_{jj} = \overline{0}$ by Lemma 11 in Chapter 4. As $\mathit{path}(\overline{0}) = [\,]$ by assumption (P1) then the required result $\forall k \in \mathit{path}(\sigma^1(\mathbf{X})_{jj}) : k \in \mathcal{F}_1$ holds trivially. □

Lemma 28. If $i \in \mathcal{C}_t$ and $l \in \mathit{path}(\sigma^t(\mathbf{X})_{ij})$ then $l \in \mathcal{C}_t$.

Proof. This is proved by induction over the path of i's route at time t.
Case 1 & 2: $\mathit{path}(\sigma^t(\mathbf{X})_{ij}) = \bot$ or $\mathit{path}(\sigma^t(\mathbf{X})_{ij}) = [\,]$
Then there cannot exist a node $l \in \mathit{path}(\sigma^t(\mathbf{X})_{ij})$ and so the statement is vacuously true.
Case 3: $\mathit{path}(\sigma^t(\mathbf{X})_{ij}) = (i, k) :: p$
If $l \in \mathit{path}(\sigma^t(\mathbf{X})_{ij})$ then either $l = i$ or $l \in p$. In the former case then $l \in \mathcal{C}_t$ as $i \in \mathcal{C}_t$. In the latter case, Lemma 25 can be used to prove that $\mathit{path}(\sigma^t(\mathbf{X})_{kj}) = p$. Therefore as $i \in \mathcal{C}_t$ then all nodes in p are in $\mathcal{F}_t$ and therefore $k \in \mathcal{C}_t$. Hence, as $l \in \mathit{path}(\sigma^t(\mathbf{X})_{kj})$, the induction hypothesis can be used to prove $l \in \mathcal{C}_t$. □

The term "junk route" was used informally in Section 6.3 to reason about count-to-convergence. A junk route is a route whose path conflicts with the routing choices made by the nodes along that path. For example consider latch 1 in the trace of Q_9 in Figure 6.5. At time 2, node 3's route is junk because node 3 believes that its packets will travel along the path $(3, 2, 0)$. However this conflicts with the routing choice of node 2 and in reality next-hop forwarding will ensure that the message travels along the path $(3, 2, 1, 0)$.

In practice it's easier to define when a route is not junk. Intuitively a route is *real* at time t if all the routing decisions of nodes along its path are consistent with one another.

Definition 38. A node i is *real* at time t if

$$\forall (k, l) \in \mathit{path}(\sigma^t(\mathbf{X})_{ij}) : \sigma^t(\mathbf{X})_{kj} = \mathbf{A}_{kl}(\sigma^t(\mathbf{X})_{lj})$$

where $\mathcal{R}_t$ is the set of real nodes at time t.

Similarly to the sets of fixed and converged nodes, the set of real nodes obeys some desirable properties: the source node's route is always real after time 1, if a node has converged than it's route is necessarily real and finally all the nodes along the path of a real node are also real. However unlike $\mathcal{F}$ and $\mathcal{C}$, the set of real nodes $\mathcal{R}$ does not necessarily grow monotonically over time as distributivity violations can repeatedly generate fresh junk routes.

Lemma 29. $\forall t > 0 : j \in \mathcal{R}_t$.

Proof. By the same reasoning as in Lemma 27, the path of j is always $[\,]$ and hence the result holds vacuously. □

Lemma 30. $\mathcal{C}_t \subseteq \mathcal{R}_t$.

Proof. Assume $i \in \mathcal{C}_t$. It will now be proven that $i \in \mathcal{R}_t$ by induction on node i's path.
Case 1 & 2: $path(\sigma^t(\mathbf{X})_{ij}) = \bot$ or $path(\sigma^t(\mathbf{X})_{ij}) = [\,]$
Then $i \in \mathcal{R}_t$ vacuously as there are no arcs in either path.
Case 3: $path(\sigma^t(\mathbf{X})_{ij}) = (i,k) :: p$
Consider an arbitrary arc $e \in path(\sigma^t(\mathbf{X})_{ij})$. Either $e = (i,k)$ or $e \in p$. If the former then the realness property $\sigma^t(\mathbf{X})_{ij} = \mathbf{A}_{ik}(\sigma^t(\mathbf{X})_{kj})$ holds by Lemma 25. If the latter, then $path(\sigma^t(\mathbf{X})_{kj}) = p$ by Lemma 25. Hence $k \in \mathcal{C}_t$ by Lemma 28 and therefore the inductive hypothesis can be applied to obtain the required property $\sigma^t(\mathbf{X})_{lj} = \mathbf{A}_{lm}(\sigma^t(\mathbf{X})_{mj})$ for any arc (l,m) in p. □

Lemma 31. For all nodes i and l and times t if $i \in \mathcal{R}_t$ then $l \in path(\sigma^t(\mathbf{X})_{ij})$ implies $l \in \mathcal{R}_t$.

Proof. The result is proved by induction over the path of node i. If $path(\sigma^t(\mathbf{X})_{ij}) = [\,]$ or $\bot$ then the result holds vacuously for all l. Likewise if $i = l$. Otherwise $path(\sigma^t(\mathbf{X})_{ij}) = (i,k) :: p$ for some node k and path p, and as $i \in \mathcal{R}_t$ then $p = path(\sigma^t(\mathbf{X})_{kj})$. Hence as all the routing decisions along $path(\sigma^t(\mathbf{X})_{ij})$ are correctly aligned so must every routing decision along $path(\sigma^t(\mathbf{X})_{kj})$. Therefore $k \in \mathcal{R}_t$ and $l \in path(\sigma^t(\mathbf{X})_{kj})$ and so the result holds by the inductive hypothesis. □

Note that if the iteration started in the initial state $\mathbf{I}$, it could be proved that $\mathcal{F}_t \subseteq \mathcal{R}_t$. However this is not true when starting in an arbitrary state $\mathbf{X}$. Figure 6.6 provides a counterexample as at time 0, node 2 is fixed but not real.

To assist with notation, a time-indexed family of orderings over arcs is defined as follows:

$$(i,k) \preceq^t (l,m) \triangleq \mathbf{A}_{ik}(\sigma^t(\mathbf{X})_{kj}) \leq \mathbf{A}_{lm}(\sigma^t(\mathbf{X})_{mj})$$

Later on Lemma 36 will require a proof that some node l permanently adopts the route through one of its neighbour m. Intuitively $(i,k) \preceq^t (l,m)$ means that the route $\mathbf{A}_{ik}(\sigma^t(\mathbf{X})_{kj})$ and some of its extensions may be capable of threatening l's adoption of the route through m at some point in the future.

6.4.2 Main proof

The main idea behind the proof of quadratic convergence is to keep track of $P(t)$, the set of nodes that can be *proved* to have converged at time t. This is a (possibly strict) subset of $\mathcal{C}_t$ the nodes that have *actually* converged at time t. It will then be shown that while there exist nodes not in $P(t)$, then at least one such node can be proven to have converged by time $t+n$. This node can therefore be added to $P(t+n)$. The whole process is then repeated until all n nodes have converged.

Consider a time $\tau \geq 1$ and assume $j \in P(\tau)$. Denote $P(\tau)$'s complement as $\overline{P(\tau)}$. Let $E(\tau)$ be the set of all arcs going from $\overline{P(\tau)}$ to $P(\tau)$ i.e. the in-cutset of $P(\tau)$. Let $e_{\text{min}}(\tau) = (i_{\text{min}}(\tau), k_{\text{min}}(\tau))$ be a minimal arc in $E(\tau)$ with respect to $\preceq^{\tau}$ (see Figure 6.7). The aim is to show that $i_{\text{min}}(\tau) \in \mathcal{C}_{\tau+n}$ and therefore that $i_{\text{min}}(\tau)$ can added as a member of $P(\tau+n)$. In order to improve readability, the dependency of $e_{\text{min}}(\tau)$, $i_{\text{min}}(\tau)$ and $k_{\text{min}}(\tau)$ on τ will be dropped and instead they will be written as just e_{min}, i_{min} and k_{min}.

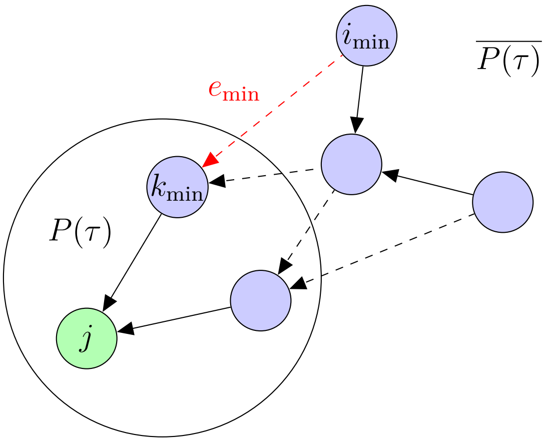


Figure 6.7: Construction of the arc $e_{\text{min}} = (i_{\text{min}}, k_{\text{min}})$. $P(\tau)$ is the set of nodes that can be proven to have converged at time τ. Dashed edges are members of $E(\tau)$, the cutset of $P(\tau)$ and $\overline{P(\tau)}$. The arc e_{min} is a minimal member of $E(\tau)$ with respect to $\preceq^{\tau}$.

What can stop node i_{min} from routing through k_{min}? In the world of distributive algebras nothing can, as the node i_{min} either uses a route from $P(\tau)$ or $\overline{P(\tau)}$. In the former case, the route through k_{min} is by definition the best route out of $P(\tau)$. In the latter case, any route out of $\overline{P(\tau)}$ must necessarily be an extension of some route out of $P(\tau)$ and hence distributivity ensures that any route through $\overline{P(\tau)}$ cannot be better than the route through k_{min}. However this line of reasoning does not hold for non-distributive algebras, as junk routes generated before time τ may continue to persist in $\overline{P(\tau)}$ after time τ. These junk routes can prevent i_{min} from adopting the route through k_{min}. The aim will now to be show that such junk routes must have been flushed from the network after a further n iterations, after which i_{min} will necessarily permanently adopt the route through k_{min}.

Firstly it will be shown that a real node in $\overline{P(\tau)}$ can never threaten e_{min}.

Lemma 32. For all $t \geq \tau$ if $k \in \mathcal{R}_t$ and $k \in \overline{P(\tau)}$ then $e_{\min} \preceq^t (i, k)$ for all nodes i.

Proof. The proof proceeds by induction on the path of $\sigma^t(\mathbf{X})_{kj}$.
Case 1: $path(\sigma^t(\mathbf{X})_{kj}) = [\,]$
If the path is empty then it must be the case that $k = j$, and as $k \notin P(\tau)$ then $j \notin P(\tau)$. This contradicts the assumption at the start of this section that the source node, j, is in $P(\tau)$.
Case 2: $path(\sigma^t(\mathbf{X})_{kj}) = \bot$
Then the required inequality holds by the following reasoning:

$$\begin{aligned}
\mathbf{A}_{i_{\min}k_{\min}}(\sigma^t(\mathbf{X})_{k_{\min}j}) &\leq \infty && \text{(by (R3))} \\
&= \mathbf{A}_{ik}(\infty) && \text{(by (R4))} \\
&= \mathbf{A}_{ik}(\sigma^t(\mathbf{X})_{kj}) && \text{(by (P2) and case 2)}
\end{aligned}$$

Case 3: $path(\sigma^t(\mathbf{X})_{kj}) = (k, l) :: p$ for some node l
If $l \in P(\tau)$ then (k, l) is an element of the cutset E and, as $e_{\min}$ is a minimal member of E, then $e_{\min} \preceq^t (k, l)$. Otherwise if $l \notin P(\tau)$ then the inductive hypothesis can be applied as $l \in \mathcal{R}_t$ by Lemma 31 and so again $e_{\min} \preceq^t (k, l)$. Therefore in either case $e_{\min} \preceq^t (k, l)$.

Hence the required inequality holds by the following reasoning:

$$\begin{aligned}
\mathbf{A}_{i_{\min}k_{\min}}(\sigma^t(\mathbf{X})_{k_{\min}j}) &\leq \mathbf{A}_{kl}(\sigma^t(\mathbf{X})_{lj}) && \text{(by } e_{\min} \preceq^t (k, l)\text{)} \\
&= \sigma^t(\mathbf{X})_{kj} && \text{(by } k \in \mathcal{R}_t\text{)} \\
&\leq \mathbf{A}_{ik}(\sigma^t(\mathbf{X})_{kj}) && \text{(by } F \text{ increasing)}
\end{aligned}$$

□

One consequence of Lemma 32 is that the only nodes that are capable of preventing $i_{\min}$ from routing through $k_{\min}$, are those that are not real and that have an extension that threatens $e_{\min}$. Such routes will be referred to as *dangerous.*

> **Definition 39.** A node k's route is *dangerous* at time $t \geq \tau$ if:
>
> $$k \notin \mathcal{R}_t \wedge \exists i : (i, k) \prec^t e_{\min}$$
>
> where $\mathcal{D}_t^\tau$ is the set of dangerous nodes at time $t \geq \tau$.

It is important to note that routes are dangerous only in relation to the current minimal arc. It will now be shown that after time τ all dangerous routes must be an extension of some other dangerous route. In other words fresh dangerous routes cannot be generated after time τ.

Lemma 33. For all $t \geq \tau$ and $k \in \mathcal{D}^{\tau}_{t+1}$ then there exists l such that $\sigma^{t+1}(\mathbf{X})_{kj} = \mathbf{A}_{kl}(\sigma^{t}(\mathbf{X})_{lj})$ and $l \in \mathcal{D}^{\tau}_{t}$.

Proof. It is immediate that $k \notin \mathcal{R}_{t+1}$ from the definition of $\mathcal{D}^{\tau}_{t+1}$. As the routes $\overline{0}$ and $\overline{\infty}$ are trivially real, k's route cannot be either $\overline{0}$ or $\overline{\infty}$. Therefore $\sigma^{t+1}(\mathbf{X})_{kj} = \mathbf{A}_{kl}(\sigma^{t}(\mathbf{X})_{lj})$ for some node l. It is now shown that $l \in \mathcal{D}^{\tau}_{t}$ by proving $(k, l) \prec^{t} e_{\min}$ and $l \notin \mathcal{R}_{t}$.

As $k \in \mathcal{D}^{\tau}_{t+1}$ then there exists a node i such that $(i, k) \prec^{t+1} e_{\min}$. The following chain of reasoning can now be used to prove that $(k, l) \prec^{t} e_{\min}$:

$$\begin{aligned}
\mathbf{A}_{kl}(\sigma^{t}(\mathbf{X})_{lj}) &= \sigma^{t+1}(\mathbf{X})_{kj} && \text{(by prev. reasoning)} \\
&\leq \mathbf{A}_{ik}(\sigma^{t+1}(\mathbf{X})_{kj}) && \text{(by } F \text{ increasing)} \\
&< \mathbf{A}_{i_{\min}k_{\min}}(\sigma^{t+1}(\mathbf{X})_{k_{\min}j}) && \text{(by } (i,k) \prec^{t+1} e_{\min}) \\
&= \mathbf{A}_{i_{\min}k_{\min}}(\sigma^{t}(\mathbf{X})_{k_{\min}j}) && \text{(by } k_{\min} \in \mathcal{C}_{t})
\end{aligned}$$

It remains to show that $l \notin \mathcal{R}_{t}$. Clearly $k \notin P(\tau)$ as that would contradict $k \notin \mathcal{R}_{t+1}$ by Lemma 30. If $l \in P(\tau)$ then (k, l) is an element of the cutset E and, as $e_{\min}$ is a minimal member of E, then $e_{\min} \preceq^{t+1} (k, l)$. Otherwise if $l \notin P(\tau)$ then $e_{\min} \preceq^{t+1} (k, l)$ by applying Lemma 32. Therefore either way $e_{\min} \preceq^{t+1} (k, l)$.

Assume $l \in \mathcal{R}_{t}$. This can be shown to be a contradiction as it was assumed that $(i, k) \prec^{t+1} e_{\min}$ but it is now possible to show that $e_{\min} \preceq^{t+1} (i, k)$ as follows:

$$\begin{aligned}
\mathbf{A}_{i_{\min}k_{\min}}(\sigma^{t+1}(\mathbf{X})_{k_{\min}j}) &= \mathbf{A}_{i_{\min}k_{\min}}(\sigma^{t}(\mathbf{X})_{k_{\min}j}) && \text{(by } k_{\min} \in \mathcal{C}_{t}) \\
&\leq \mathbf{A}_{kl}(\sigma^{t}(\mathbf{X})_{lj}) && \text{(by } e_{\min} \preceq^{t+1} (k,l)) \\
&= \sigma^{t+1}(\mathbf{X})_{kj} && \text{(by prev. reasoning)} \\
&\leq \mathbf{A}_{ik}(\sigma^{t+1}(\mathbf{X})_{kj}) && \text{(by } F \text{ increasing)}
\end{aligned}$$

Hence $l \notin \mathcal{R}_{t}$ and so $l \in \mathcal{D}^{\tau}_{t}$. □

Lemma 33 will now be used to prove that all dangerous routes must have been flushed from the routing state by time $t + n$.

Lemma 34. $\mathcal{D}^{\tau}_{\tau+n} = \emptyset$

Proof. By Lemma 33 the length of the paths of the dangerous routes must increase by 1 each iteration. Hence after n iterations the path of every dangerous route must necessarily contain a loop. As routes containing looping paths are removed, after n iterations all dangerous routes must have been eliminated from the routing state. □

It will now be shown that at any time after $\tau + n$ the route through $k_{\min}$ is guaranteed to be the best route on offer to $i_{\min}$.

Lemma 35. For every node t and time $t \geq \tau + n$ then:

$$(i_{\min}, k_{\min}) \preceq^t (i_{\min}, k)$$

Proof. Consider an arbitrary node k.
<u>Case 1</u>: $k \in P(\tau)$
Then the inequality is immediate as $e_{\min}$ is a minimal arc into $P(\tau)$.
<u>Case 2</u>: $k \notin \mathcal{R}_t$
Suppose the inequality does not hold. Then k's route is dangerous. But by Lemma 34 no dangerous routes exist at time t. Hence the inequality must hold.
<u>Case 3</u>: $k \notin P(\tau)$ and $k \in \mathcal{R}_t$
Then the inequality is immediate by Lemma 32. □

Hence at any time $t \geq \tau + n$ then $i_{\min}$ must route through $k_{\min}$.

Lemma 36. For every time $t \geq \tau + n$ then :

$$\sigma^t(\mathbf{X})_{i_{\min}j} = \mathbf{A}_{i_{\min}k_{\min}}(\sigma^{t-1}(\mathbf{X})_{k_{\min}j})$$

Proof. As $i_{\min} \notin P(\tau)$ and $j \in P(\tau)$ then clearly $i_{\min} \neq j$. Hence the inequality can be proved as follows:

$$\begin{aligned} \sigma^t(\mathbf{X})_{i_{\min}j} &= \bigoplus_k \mathbf{A}_{i_{\min}k}(\sigma^{t-1}(\mathbf{X})_{kj}) && \text{(by def of } \sigma \text{ and } i_{\min} \neq j) \\ &= \mathbf{A}_{i_{\min}k_{\min}}(\sigma^{t-1}(\mathbf{X})_{k_{\min}j}) && \text{(by Lemma 35)} \end{aligned}$$

□

It is now finally possible to prove that $i_{\min}$ will have converged at time $\tau + n$.

Lemma 37. $i_{\min} \in \mathcal{C}_{\tau+n}$

Proof. To prove that $i_{\min}$ has converged at time $\tau + n$ it is necessary to prove that both it and all the nodes on its path are fixed at time $\tau + n$. Consider an arbitrary $t \geq \tau + n$ then:

$$\begin{aligned} \sigma^{\tau+n}(\mathbf{X})_{i_{\min}j} &= \mathbf{A}_{i_{\min}k_{\min}}(\sigma^{\tau+n-1}(\mathbf{X})_{k_{\min}j}) && \text{(by Lemma 36)} \\ &= \mathbf{A}_{i_{\min}k_{\min}}(\sigma^{t-1}(\mathbf{X})_{k_{\min}j}) && \text{(by } k_{\min} \in \mathcal{C}_\tau) \\ &= \sigma^t(\mathbf{X})_{i_{\min}j} && \text{(by Lemma 36)} \end{aligned}$$

and hence $i_{\min}$ is fixed at time $\tau + n$.

It must now be shown that all nodes in $i_{\min}$'s path are fixed at time $\tau+n$. By Lemma 36 $i_{\min}$ routes through $k_{\min}$ for all times $t \geq \tau + n$. As $k_{\min}$ is in $P(\tau)$ then all nodes in its path are fixed at time t as $P(\tau) \subseteq \mathcal{C}_\tau \subseteq \mathcal{C}_{\tau+n} \subseteq \mathcal{C}_t \subseteq \mathcal{F}_t$. Hence all nodes in the path of $i_{\min}$ are fixed at time $\tau + n$ and so $i_{\min} \in \mathcal{C}_{\tau+n}$. □

Theorem 9. For any increasing path algebra $(S,\ \oplus,\ E,\ \overline{\infty},\ \overline{0},\ \mathit{path})$ and $n \times n$ adjacency matrix then σ converges from any starting state $\mathbf{X}$ in at most n^2 iterations.

Proof. Let $P(t)$ be the set of nodes that can be *proven* to have converged at time t. Clearly $P(0) = \emptyset$. At time 1 the source node j has converged by Lemma 27 and so let $P(1) = \{j\}$. By fixing the current time as τ, it is possible to repeatedly generate a new node $i_{\min}(\tau)$ not yet in $P(\tau)$ via the construction shown in Figure 6.7. Lemma 37 proves that $i_{\min}(\tau)$ will have converged at time $\tau + n$. Thus let $P(\tau + n) = P(\tau) \cup \{i_{\min}(\tau)\}$.

By repeating the process above $n - 1$ times, every node will necessarily be a member of $P(n(n-1)+1)$ and so every node has been proved to have converged at time n^2-n+1. □

6.5 Conclusions

This section has studied the rate of convergence of increasing path-vector protocols, and proved a new, tight worst-case bound of $\Theta(n^2)$. This has significant implications for protocol design as it implies that even when the algebra is non-distributive then path-vector protocols will still converge relatively quickly. This contrasts the best previous upper bound which suggested the worst case might require as many as $n!$ iterations.

Additionally, it is hypothesised that quadratic convergence is unlikely to occur in practice. The example described in Section 6.3 is *extremely* sensitive to small changes in the policies, with even small alterations drastically reducing the number of iterations required for convergence. Furthermore, the policies in question require that every node in the network participates in multiple global loops which seems unlikely to occur in today's networks given the hierarchical nature of many modern network architectures.

There remain several open questions in the area. Firstly it does not appear the case that all increasing, non-distributive path algebras have a quadratic worst-case. For example the shortest-widest paths algebra has the property that one distributivity violation can never trigger a second distributivity violation and hence appears to only require $2n-2$ iterations to converge in the worst-case. Consequently it is hypothesised that there may be a hierarchy of increasing path algebras whose worst-case convergence time ranges from linear to quadratic.

Secondly the rate of the convergence of path-vector protocols in free networks is still an open question. The proof presented in Section 6.4 fails at Lemma 32 for non-increasing algebras. In such an algebra, real routes as well as junk routes may prevent $i_{\min}$ from using $k_{\min}$'s route. Therefore it is not necessarily the case that $i_{\min}$ will be the next node to converge. It is possible that redefining what it means to be the minimal edge may provide a path to a new proof. It seems unlikely however that the quadratic convergence bound could be preserved in such a proof.

Chapter 7

Practicalities

In any mathematical model of a real-world system there is a fundamental tension between adding implementation details to better describe a particular system, and preserving generality so that it can describe many different systems. So far this thesis has very much erred on the side of generality, and has reasoned about vector-based routing protocols in a very abstract manner. The downside of this is that certain assumptions made in the model may not match up exactly with the properties of real-world routing protocols.

For example, in BGP there exist valid routes x and y for which $x \oplus y \neq y \oplus x$, and therefore technically $\oplus$ is not commutative as was assumed in the definition of a routing algebra. However it is “morally” commutative in that the protocol will never be required to choose between such a pair of routes. The naive response to this would be to generalise the notion of a routing algebra in the right way and redo all the proofs. However depending on the exact generalisation this might be quite a complex process and significantly complicate the proofs. Furthermore each new generalisation would require redoing the proofs in their entirety. To get around this problem the technique of *bisimulation* may be used: if the behaviour of δ over a commutative algebra $\mathbb{A}$ is indistinguishable from the behaviour of δ over a non-commutative algebra $\mathbb{B}$ then if $\mathbb{A}$ is convergent then $\mathbb{B}$ must also be convergent.

This chapter presents three examples of such small mismatches between the theory and real world protocols and shows how each of them be addressed without altering the core proofs. The techniques described will then be used in Chapter 8 to prove the correctness of a protocol containing many of the features of BGP. In some sense this is an important test of the model’s adequacy, i.e. that the level of abstraction required to ensure the generality the proofs is not so high as to prevent it being applied to real-world protocols.

7.1 Relaxation of assumptions on choice

As described in this chapter’s introduction, the choice operators of a path-vector protocol may only “morally” obey the properties required by a routing algebra. This is because during the operation of the protocol a node need never choose between two routes advertised by the same neighbour. Consequently a choice operator whose final tie-breaking procedure involves comparing the neighbours' IDs need not be commutative when used to choose between two routes from the same neighbour. The notion of whether or not a pair of routes can ever be compared during the execution of δ can be formalised as follows.

Definition 40. Routes x and y are *comparable* if one of the following holds:

1. $x = \overline{0}$, $y = \overline{\infty}$ or $x = \overline{\infty}$, $y = \overline{0}$ or $x = \overline{\infty}, y = \overline{\infty}$.
2. one of x and y belongs to the set $\{\overline{0}, \overline{\infty}\}$ and there exists nodes i and j, function $f \in E_{ij}$ and path-weight $z \in S$ such that the other is equal to $f(z)$.
3. there exists nodes i, j, k such that $j \neq k$, functions $f \in E_{ij}$ and $g \in E_{ik}$ and path-weights $w, z \in S$ such that $x = f(w)$ and $y = f(z)$.

where C is the set of comparable pairs of routes.

Even if the choice operator is only commutative for comparable pairs of routes, it is almost always possible to construct an alternative choice operator that is fully commutative. Consider a path algebra $(S, \oplus_1, E, \overline{\infty}, \overline{0}, \mathit{path})$ where $\oplus_1$ is only commutative over the set of comparable pairs. Suppose $\oplus_2$ is an alternative fully commutative choice operator. Then a composite choice operator $\oplus_3$ can be defined as follows:

$$x \oplus_3 y = \begin{cases} x \oplus_1 y & \text{if } (x, y) \in C \\ x \oplus_2 y & \text{otherwise} \end{cases}$$

It is easy to show that $\oplus_3$ is commutative over all pairs of routes, and that at the same time its behaviour is indistinguishable from that of $\oplus_1$ for all comparable pairs. Hence the algebra $(S, \oplus_1, E, \overline{\infty}, \overline{0}, \mathit{path})$ should have exactly the same convergence properties as the algebra $(S, \oplus_3, E, \overline{\infty}, \overline{0}, \mathit{path})$.

To formalise this, it is necessary to find a general relationship between two algebras such that the resulting asynchronous iteration functions are bisimilar. In practice, this simply requires a pointwise correspondence between each of the components of the algebras:

Definition 41. Algebras $\mathbb{A}$ and $\mathbb{B}$ are *bisimilar* if there exists a relation $\sim$ between $S_{\mathbb{A}}$ and $S_{\mathbb{B}}$ such that:

1. $\overline{\infty}_{\mathbb{A}} \sim \overline{\infty}_{\mathbb{B}}$
2. $\overline{0}_{\mathbb{A}} \sim \overline{0}_{\mathbb{B}}$

3. $\forall w, x \in C_{\mathbb{A}}, y, z \in C_{\mathbb{B}} : (w \sim y) \wedge (x \sim z) \Rightarrow (w \oplus_{\mathbb{A}} x) \sim (y \oplus_{\mathbb{B}} z)$

4. $\forall x \in C_{\mathbb{A}}, y \in C_{\mathbb{B}}, i, j \in V, f \in E^{\mathbb{A}}_{ij}, g \in E^{\mathbb{B}}_{ij} : (f \sim g) \wedge (x \sim y) \Rightarrow f(x) \sim g(y)$

Note that the algebras in this definition are not necessarily *routing* algebras in the technical sense, as one or both of them may not obey the required properties. In the case of non-commutative choice operators described above, $S_{\mathbb{A}} = S_{\mathbb{B}}$ and therefore the relation $\sim$ is simply equality.

Theorem 10. If algebras $\mathbb{A}$ and $\mathbb{B}$ are bisimilar then δ is deterministically convergent over $\mathbb{A}$ if and only if δ is deterministically convergent over $\mathbb{B}$.

Proof. The proof that δ preserves the bisimilarity relation $\sim$, and hence convergence, proceeds by induction over the definition of δ and time t in the obvious way. See the Agda library if interested in the full details of the proof. □

Hence in the case of the non-commutative algebras described above, if it can be shown that $\mathbb{A} = (S, \oplus_3, E, \overline{\infty}, \overline{0}, path)$ is an increasing path algebra and that is bisimilar to $\mathbb{B} = (S, \oplus_1, E, \overline{\infty}, \overline{0}, path)$ then $\mathbb{B}$ is deterministically convergent, even if $\mathbb{B}$ is not an increasing path algebra itself.

7.2 Path inflation

The choice operator in BGP first chooses the route with the highest local preference and, in cases where the local preference is equal, next chooses the route with the shortest AS path. Local preference values are assigned specific semantics within each AS and are erased when the route is exported across AS boundaries. Therefore the question arises how network operators in one AS can make one of its advertised routes more desirable than another without relying on the local preference value. BGP solves this problem using *path inflation*, where routers artificially lengthen the AS path by padding the beginning of the path with copies of the current AS. The more the path is inflated, the less desirable it becomes.

The problem with this process is that the inflated paths now have loops in them and therefore are no longer simple. Furthermore, if the *path* function simply extracted the path from the path-weight it would violate assumption (P3) which requires the following to hold:

$$\forall x \in S, ij \in V, f \in F_{ij} : path(f(x)) = \begin{cases} \bot & \text{if } f(x) = \overline{\infty} \\ (i,j) \mathbin{::} path(x) & \text{otherwise} \end{cases}$$

For example for a path weight x and a function $f \in E_{ij}$ then it is possible that the path of the extension $f(x)$ might be:

$$path(f(x)) = (i, i) :: (i, i) :: (i, j) :: path(x)$$

In reality this problem is easy to address as there is no requirement that the *path* function must simply return the internal path. Instead it can be redefined to perform a pre-processing step of *deflating* the path before returning it, thereby stripping out any extra instances of the current AS. It is easy to show that this new implementation of *path* obeys (P3).

7.3 Hierarchical paths

The final problem addressed is that BGP stores the path along which a route was generated as two separate entries in the path-weight. The first entry contains the top-level AS path, and the second entry contains the router level path within the current AS. A route is discarded when a loop forms in either the AS path or the router path. Furthermore when a route is exported across AS boundaries, the router level path is erased in order to avoid revealing the internal topology of an AS's network to its competitors.

As with path inflation, the erasure of the router level path violates assumption (P3) in the definition of a path algebra. However, unlike path inflation which *adds* redundant information to the path, erasing the router level path *removes* the information needed to satisfy (P3). Therefore it is impossible to fix this problem by altering the definition of the *path* function.

Instead, as with the problem of non-commutativity discussed earlier, it may be solved by bisimulation. Imagine an almost identical algebra that did not erase the router level path, but at the same time hid this additional information from the choice and extension operators. This new algebra's behaviour would by definition be indistinguishable to that of the original algebra. Yet at the same time the full path could now be extracted by the *path* function, hence allowing the new algebra to satisfy assumption (P3). This scheme could even be generalised to networks that had more than the two level-hierarchy found in BGP.

7.4 Conclusions

The definition of routing algebras and path algebras capture the core operations and properties of idealised distance-vector and path-vector protocols. As with most models however, real-life implementations differ from this idealised form. This section has shown how non-commutative choice, path inflation and hierarchical paths can all be handled in a principled manner, without the need to alter the proofs in the previous chapter. This provides strong supporting evidence that the level of abstraction chosen for the model is the right one.

Chapter 8

Agda: proofs and protocols

All of the work discussed thus far, with the exception of the execution trace of Q_n, has been formalised in Agda [53]. Agda is a dependently typed language that is expressive enough that both programs and proofs may be written in it. It can therefore be used as both a proof assistant and a functional programming language.

Initially this chapter discusses the advantages of having formalised the proofs. It then provides a high-level overview of the resulting library, which is available online [19]. Finally it demonstrates how this library may be used to formally prove the correctness of vector-based protocols. The example algebra chosen contains many of the features of BGP including local preferences, communities, path-dependent conditional policy and path inflation. Unlike BGP, the policies are defined in such a way that local preferences can only increase. Consequently the protocol is guaranteed to be convergent. It is important to note that this protocol is not being presented as the solution to the problems with BGP, but merely as a demonstration of how far the theory has progressed.

8.1 Advantages of formalisation

The results presented in the previous chapters are relatively technical, and while they have been checked by multiple people and most have undergone peer review, this does not guarantee that they are correct. Üresin & Dubois's paper on the theory of asynchronous iterations [64] is an excellent example of this. Despite being an important and widely cited result in the literature, during formalisation it was found that not only were the proofs of Propositions 3 and 4 incorrect, but the claims themselves were untrue. This is discussed in more detail in [68]. Formalising the proofs in Agda minimises the risk of such a scenario by allowing them to be verified by a computer. This shifts the burden on the reader from checking the correctness of the proofs to only checking that the statement of the proofs is correct.

Every mathematical result in this thesis down to the most trivial lemma has been formalised. This includes the generalisation of the work by Üresin & Dubois [64] and Gurney [35]. The only exception is the claim that in the general case the family of graphs Q_n takes $\Omega(n^2)$ iterations to converge. The reasons for this were explained in the introduction to Chapter 6.

As well as increasing the reader's confidence in the correctness of the work, formalisation has several other advantages. Firstly the act of formalisation itself has been invaluable in creating and shaping the proofs in this thesis. For example, the initial pen-and-paper versions of the proofs in Chapter 5 that showed that the metrics were strictly contracting were poorly structured and contained duplicated logic in several places. The process of formalising them highlighted these inefficiencies and led to the much improved presentation found in this thesis. Secondly, dependencies between assumptions and proofs may be checked far more quickly and easily in a proof assistant. In a pen-and-paper proof it is often difficult to be sure that an assumption that appears to be unnecessary has not been used implicitly anywhere. It is doubtful that the relaxations of the ACO and AMCO conditions, described in Section 3.2.3, would have been found without having formalised them.

8.2 The library

This section presents an overview of the resulting Agda library, which is publicly available online [19]. As part of the library, there exists a README file that matches up the proofs and definitions in this thesis with the associated Agda code. It is hoped that the library will be of use to others, either to extend the algebraic theory with new results or to verify the design of new protocols.

The code for a few of the more important definitions from the previous chapters are now shown in order to demonstrate how the code closely mirrors the mathematical definitions. This thesis does not provide a tutorial in Agda but Agda's use of unicode allows code to closely match the naming conventions used throughout the thesis, and so it is hoped that the code is understandable to non-experts. Interested readers may find several excellent introductions to Agda online [2].

The underlying structure of a routing algebra is formalised as dependant record type in the library:

```
record RawRoutingAlgebra : Set₁ where
  field
    S       : Set
    _⊕_     : S → S → S
    E       : ∀ {n} → Node n → Node n → Set
    0#      : S
```

```
∞#       : S

_≈_      : Rel S 0ℓ
_▷_      : ∀ {n} {i j : Node n} → E i j → S → S

≈-decEq : IsDecEquivalence _≈_
⊕-cong   : ∀ {w x y z} → w ≈ x → y ≈ z → w ⊕ y ≈ x ⊕ z
▷-cong   : ∀ {n} {i j : Node n} (f : E i j) {x y : S} → x ≈ y → f ▷ x ≈ f ▷ y
```

Although the first five fields of the RawRoutingAlgebra record match the notation used in the definition of a routing algebra, there are a couple of extra fields needed in the formalisation. Firstly because Agda uses type theory rather than set theory as a foundation, it is difficult to directly capture the notion of a set of functions. The field _▷_ is therefore the application operator, and f▷x should be simply read as $f(x)$. Secondly, equality is usually taken for granted in written mathematics. However, in a proof assistant the notion of equality _≈_ needs to be made explicit. The three additional assumptions ≈-decEq, ⊕-cong and ▷-cong ensure that the equality relation is a decidable equivalence and that the choice and extension operations are well-behaved with respect to it.

The record IsRoutingAlgebra captures the concept of a RawRoutingAlgebra obeying the required properties to be a routing algebra:

```
record IsRoutingAlgebra (algebra : RawRoutingAlgebra) : Set where
  field
    ⊕-sel        : Selective _⊕_
    ⊕-comm       : Commutative _⊕_
    ⊕-assoc      : Associative _⊕_
    ⊕-zeroʳ      : RightZero 0# _⊕_
    ⊕-identityʳ  : RightIdentity ∞# _⊕_
    ▷-fixedPoint : ∀ {n} {i j : Fin n} (f : E i j) → f ▷ ∞# ≈ ∞#
```

Given a routing algebra and an adjacency matrix $\mathbf{A}$ then the iteration operator

$$F(\mathbf{X})_{ij} = \bigoplus_{k} \mathbf{A}_{ik}(\mathbf{X}_{kj}) \oplus \mathbf{I}_{ij}$$

is defined in Agda as follows:

```
F : RoutingMatrix → RoutingMatrix
F X i j = foldr _⊕_ (I i j) (tabulate (λ k → A i k ▷ X k j))
```

The final statement of the Theorem 8 which proves that strictly increasing path algebras are convergent is formalised as:

```
increasingPathAlgebras-converge : ∀ (𝔸 : RawRoutingAlgebra) →
                                  IsRoutingAlgebra 𝔸 →
                                  IsPathAlgebra 𝔸 →
                                  IsIncreasing 𝔸 →
                                  ∀ {n : ℕ} (N : Network 𝔸 n) →
                                  Convergent (F 𝔸 N)
```

where the formal definition of Convergent is:

```
record Convergent (F : AsyncIterable) : Set where
  field
    k*         : Epoch → Subset n → ℕ
    x*         : Epoch → Subset n → S
    x*-fixed   : ∀ e p → F e p (x* e p) ≈ x* e p
    x*-reached : ∀ (x₀ : S) (s : Schedule n) {t₁ t₂ t₃ : 𝕋} →
                 IsMultiPseudoperiodic s (k* (η s t₁) (ρ s t₁)) [ t₁ , t₂ ] →
                 IsSubEpoch s [ t₂ , t₃ ] →
                 δ F s x₀ t₃ ≈ x* (η s t₁) (ρ s t₁)
```

It should be noted that the formalisation has not been a trivial effort. The development contains over 27,000 lines of code, 10,000 of which have been submitted to the Agda standard library [67]. The routing library itself is split over 203 files and requires 2.5 minutes to type check on a midrange laptop.

8.3 An example algebra

This section now presents an example of how the library can be used to develop a safe-by-design routing protocol. The example chosen is a path-vector algebra that contains many of the features of BGP such as local preferences, community values, path inflation and conditional policies. The policies can perform various operations including path-filtering and modifying local preferences and communities. The conditions themselves are implemented using a simple language of predicates that have the ability to inspect the entire route including the path and the communities. The algebra is a generalisation of the Stratified Shortest Paths algebra described in Section 4.4.3.

It differs from the algebra underlying today's BGP in two crucial ways. Firstly, BGP allows ASs to hide their local preference values and set it to arbitrary values for routes imported from other ASs. Consequently BGP's algebra is not (strictly) increasing. The algebra described below circumvents this problem by only allows policies to increase the local preference value. Secondly, the algebra does not implement the MED attribute found in BGP as the current implementation of MED is known to be non-associative [31].

Finally there is a small cosmetic change in the semantics of the local preference value. Instead of higher local preference values being more desirable, as in BGP, in this algebra lower local preference values are more desirable. To avoid the resulting ambiguity, the name "level" is therefore used for these values instead of "local preference". This is purely a cosmetic change introduced for convenience in the formalisation, and could easily be reversed, albeit at the cost of reducing the readability of the code.

Again it should be emphasised that this algebra is not being presented as a practical solution to the problems in BGP. The open question of whether hidden information is compatible with increasing algebras will be discussed in Section 9.3.2. Instead the aim is to show that most of the features of BGP are inherently well-behaved. Furthermore it provides a demonstration of how far the theory has progressed that it is now possible to formally prove the correctness of such a complex protocol.

8.3.1 Path weights

Each path weight in the algebra contains i) a level, ii) a set of communities the route belongs to and iii) the path along which the route was generated. This is defined in Agda as follows:

```
data Route : Set where
  invalid : Route
  valid   : Level → CommunitySet → Path → Route
```

where the invalid route is obviously:

```
∞#  : Route
∞# = invalid
```

and the trivial route is the valid route with a level of 0, an empty set of communities and the empty path:

```
0#  : Route
0# = valid 0 ∅ []
```

8.3.2 Choice

Next the choice operator, $\oplus$ is defined. First the route with the lowest level is chosen. In the event of a tie, the route with the shortest path is chosen. Finally if still tied then the paths are compared lexicographically.

```
_⊕_ : Op₂ Route
x@(invalid)      ⊕ y                    = y
```

```
x                ⊕ y@(invalid)        = x
x@(valid l cs p) ⊕ y@(valid m ds q) with compare l m
... | tri< l<m = x
... | tri> m<l = y
... | tri≈ l=m with compare (length p) (length q)
...   | tri< |p|<|q| = x
...   | tri> |q|<|p| = y
...   | tri≈ |p|=|q| with p ≤_lex? q
...      | yes p≤q = x
...      | no q≤p = y
```

As discussed in Section 7.1, this operator is not necessarily commutative for routes that originate from the same neighbour. This is because it never inspects the sets of communities and therefore when choosing between a pair of routes which differ only in their community sets, the order in which the routes are chosen between matters. Likewise the trivial route is not strictly an annihilator for ⊕. However it is commutative for any two routes coming from different neighbours, and likewise the trivial route is the annihilator when it is compared against any extension of a neighbour's route.

8.3.3 Extension

Next the set of extension functions will be defined. To start with, a simple yet expressive language for conditions is constructed that can be used by the policy language to make decisions.

```
data Condition : Set where
  _and_   : Condition → Condition → Condition
  _or_    : Condition → Condition → Condition
  not     : Condition → Condition
  inPath  : Node      → Condition
  inComm  : Community → Condition
  hasPref : Level     → Condition
```

The semantics of the conditional language is defined by the evaluate function that determines whether a given condition is holds for a particular route.

```
evaluate : Condition → Route → Bool
evaluate (p and q)   x               = evaluate p x ∧ evaluate q x
evaluate (p or q)    x               = evaluate p x ∨ evaluate q x
evaluate (not p)     x               = not (evaluate p x)
evaluate (inComm c) (valid l cs p) = ⌊ c ∈? cs ⌋
evaluate (hasPref  k) (valid l cs p) = ⌊ k ≟ l ⌋
evaluate (inPath   i) (valid l cs p) = ⌊ i ∈? p ⌋
evaluate (inComm  c) invalid         = false
evaluate (hasPref  k) invalid        = false
evaluate (inPath   i) invalid        = false
```

The policy language is then defined as follows:

```
data Policy : Set₁ where
  reject      : Policy
  incrLevelBy : ℕ → Policy
  addComm     : Community → Policy
  delComm     : Community → Policy
  inflate     : ℕ → Policy
  _;_         : Policy → Policy → Policy
  if_then_    : Condition → Policy → Policy
```

The semantics of the language is defined by the function that applies policies to routes:

```
apply : Policy → Route → Route
apply p               invalid        = invalid
apply reject          x              = invalid
apply (incrLevelBy x) (valid l cs p) = valid (l + x) cs p
apply (addComm c)     (valid l cs p) = valid l (add c cs) p
apply (delComm c)     (valid l cs p) = valid l (remove c cs) p
apply (inflate n)     (valid l cs p) = valid l cs (inflate p n)
apply (p ; q)         x              = apply q (apply p r)
apply (if c then p)   x              = if (evaluate c r) then (apply p r) else r
```

For example a policy that increases the level of all routes through node 8 when the route does not belong to community 2 and rejects all routes through node 3 can be written as:

```
policy : Policy
policy = if 8 inPath and not inComm 2 then
           incrLevelBy 100 ; addComm 2 ;
         if 3 inPath then
           reject
```

One important point to note is that the policy language only provides the ability to increase a route's level, and hence it is impossible to define a non-increasing policy.

Next the set of edge functions is defined. Every extension function from i to j is parametrised by a policy.

```
data E {n} (i j : Node n) : Set₁ where
  edge : Policy → E i j
```

The result of applying an extension function to a route is specified by the application operator as follows:

```
_▷_ : ∀ {n} {i j : Node n} → E i j → Route → Route
_▷_ {_} {_} {_} _           invalid         = invalid
_▷_ {_} {i}  {j}  (edge pol) (valid x cs p) with (i , j) ⇔? p | i ∈? p
... | no _        | _         = invalid
... | yes _       | yes _     = invalid
... | yes ij⇔p | no i∈p = apply pol (valid x cs ((i , j) :: p))
```

where $(i\ ,\ j)$ ⇔? p tests if the edge (i, j) is a valid extension of path p, i.e. if $j = src(p)$, and i ∉? p tests whether or not i already exists in p, i.e. if the resulting path would contain a loop. Therefore the extension of the invalid route always results in the invalid route, and the extension of a valid route is invalid if the resulting path is not a valid simple path, and otherwise the path is extended and the specified policy is applied.

8.3.4 Path function

The path function required by the definition of a path algebra can be defined as follows:

```
path : Route → Path
path invalid          = ⊥
path (valid _ _ p) = deflate p
```

Hence the invalid route gets mapped to the invalid path, and valid routes return the deflated version of their internal path. As discussed in Section 7.2 the deflation step is needed at otherwise the path wouldn't obey assumption (P3) in the definition of a path algebra.

8.3.5 Convergence

In order to prove that the algebra:

$$\mathbb{A} = (\mathsf{Route}, \oplus, \mathsf{E}, \infty\#, 0\#, \mathsf{path})$$

is deterministically convergent it must be shown that either it is a strictly increasing path algebra or it is bisimilar to such an algebra. The algebra is strictly increasing, as it impossible to write a policy that decreases the level and the definition of the extension function guarantees that the path length always strictly increases. It also satisfies all the axioms of a path algebra. However it does not fulfil the requirements to be a routing algebra, as $\oplus$ is only commutative over comparable pairs of routes. Likewise `0#` is only technically an annihilator when paired with a comparable route.

This problem can be overcome by the bisimulation technique described in Section 7.1. First an alternative choice operator is defined that is identical to $\oplus$ except in the case when the two paths are equal, in which case it then compares the set of communities lexicographically. This new operator is commutative and observably indistinguishable from $\oplus$ during the operation of the protocol. Hence it can be proved that this new algebra is both bisimilar to $\mathbb{A}$ and is a strictly increasing path algebra. The Agda formalisation of Theorem 8 from Chapter 5 and Theorem 10 from Chapter 7 can therefore be used to prove that a path-vector protocol based on $\mathbb{A}$ is deterministically convergent.

What does this result mean? Consider a dynamic network in which network operators have complete freedom to write any policy they like, and in which nodes and links are constantly added and removed and individual policies may be altered at any time. The result guarantees that if at any point in time the network experiences a sufficiently long period of stability then the protocol will always converge to a fixed point. Furthermore this fixed point is uniquely determined by the current network topology and is therefore unaffected by both the state at the start of the period of stability and the distribution of message arrival times.

Chapter 9

Conclusion

This chapter summarises the contributions this thesis has made to the theory of routing as well as to that of asynchronous iterative algorithms in general. It then goes on to discuss some possible applications of the results. Finally it describes some of the open theoretical questions in routing.

9.1 Contributions

This thesis has made two contributions to the general asynchronous theory. Firstly it has weakened the sufficient requirements for the static asynchronous processes to always converge to a unique fixed point. Although it is unclear whether this allows any new asynchronous processes to be proven to converge, at the very least it lessens the burden of proof for users of the theorems. Secondly it has proposed a new more general model for asynchronous processes in which both the set of participants and the underlying computation may change over time. This is a more realistic model for "always on" processes such as routing and consensus algorithms.

Type	Property	Convergence	Deterministic	Rate
Distance vector	Free network	?	?	?
	Str. increasing	**Local**	**Yes**	$\boldsymbol{\Theta}(\vert\mathbf{S}\vert)$
	Distributive	Global	Yes	$\Theta(\vert S\vert)$
Path vector	Free network	Local	?	?
	Str. increasing	Local	**Yes**	$\boldsymbol{\Theta}(\mathbf{n^2})$
	Distributive	Global	Yes	$\Theta(n)$

Table 9.1: Current state of knowledge. Entries in bold are new contributions by this thesis.

Table 9.1 shows the state of knowledge about the various classes of vector-based routing protocols. This thesis has significantly reduced the unknowns around the behaviour of vector-based protocols with strictly increasing algebras. This class contains all vector-based protocols that are always guaranteed to converge. In particular the thesis has shown for the first time that strictly increasing distance-vector protocols converge. It has also shown that convergence for all such vector-based protocols is deterministic and so the final routing solution reached does not depend on the ordering of messages exchanged or how the current network configuration was reached. Finally it has proved a new, tight bound of $\Theta(n^2)$ on the worst case rate of convergence for such path-vector protocols. This is significantly better than the previous upper bound of $O(n!)$.

9.2 Applications

This section explores some of the applications of the work in this thesis.

9.2.1 Protocol design

The work has implications for the design of vector-based protocols. Firstly it demonstrates that it is possible to add conditional policy to distance-vector protocols. This is mostly an unexplored area.

Secondly, the guarantee of deterministic convergence implies that the underlying algebra being strictly increasing not only ensures convergence but also guarantees that problems such as BGP wedgies cannot arise. This means that no further measures would be needed by network operators to avoid such problems.

Thirdly, the fact that the proofs of convergence do not assume reliable communication between nodes, opens up the possibility of designing a new, lightweight path-vector protocol that runs over UDP rather than TCP.

Finally, the accompanying Agda library may be used directly to quickly and easily formally verify the algebra underlying a protocol design. This means that a formal proof of correctness of a hypothetical protocol can be created in a matter of hours.

9.2.2 Verifying BGP policies in data centres

The unsolved problem of how to ensure strict increasingness without revealing policy means that applying the work directly to BGP in the internet is a little way off. However BGP and similar protocols are now commonly being deployed in local networks such as data centres. In such environments, the network is owned and managed by a single operator and so the hidden informations concerns do not arise. Therefore in such situations it should be practical to ensure that all polices are indeed strictly increasing.

Furthermore, recently work such as Propane [6], has developed tools for the automatic verification of policies. Such tools take high-level specifications of the desired traffic flow and compile it down to BGP policies. These tools could incorporate an additional verification step to check whether all the resulting policies were strictly increasing. If not, then the tool could at the very least warn the user that convergence is not guaranteed.

9.2.3 Further afield

All of the algebras discussed in this thesis have been interpreted through the lens of routing problems. However this does not mean that there do not exist other interpretations. The matrix iteration:

$$F(\mathbf{X}) = \mathbf{A}(\mathbf{X}) \oplus \mathbf{I}$$

is reasonably general and it is not inconceivable to imagine that this equation appears in other fields, either in Computer Science or beyond. It may be that the theory developed in this thesis and other related work is applicable to other iterative algorithms in other fields. Likewise, it may be that other fields may have insights into the iterative process that would prove useful in routing.

9.3 Open questions

Finally some of the remaining open theoretical questions in routing are discussed.

9.3.1 Free networks

Obviously it would be useful to address the remaining gaps in Table 9.1 about the convergence in free networks [62]. It seems likely that finite distance-vector protocols will also converge in free networks to a local optimum. It is less clear what the worst case rate of convergence for path-vector protocols would be in free networks and whether convergence remains deterministic.

9.3.2 Hidden information

As discussed, the algebra of BGP is not strictly increasing because an AS will erase a route's local preference before advertising it externally to another AS. This is so as to avoid exposing their own internal proprietary routing policies. At first glance it appears impossible that any routing protocol that allows nodes to erase information that $\oplus$ uses to make its decision can ever be strictly increasing. However it is conceivable that some signalling mechanism, such as community values, could be used to allow ASs to coordinate and avoid this problem. Therefore the more general question of whether or not a strictly increasing algebra can hide information is still open. This question could be answered by either finding a protocol with hidden information that is convergent over all networks and yet does not require global coordination between the nodes or by proving that no such protocol exists.

If the latter is the case then a different solution may be required to BGP's problems. For instance, although politically difficult, in such a case it might be necessary to make local preferences visible and to have an industry standard describing the semantics of different local preference values.

9.3.3 Further rate of convergence results

Not all non-distributive strictly increasing path algebras converge in $O(n^2)$. Consider the shortest-widest algebra used as an example throughout this paper. Although it is not distributive, nonetheless it is *locally distributive* as it satisfies the following property:

$$h(f(x) \oplus f(y)) \leq a, b \leq h(f(x \oplus y)) \Rightarrow g(a) \oplus g(b) = g(a \oplus b)$$

This says that any time there is a distributivity violation $f(x) \oplus f(y) < f(x \oplus y)$, then distributivity holds locally for any two routes sandwiched between any extension of this violation. Hence junk routes generated by a distributivity violation cannot be used to generate further junk routes. Although there is not yet a rigorous proof of this, it is conjectured that the shortest-widest-paths path algebra always converges in $2n$ iterations.

Furthermore it seems likely there is an infinite hierarchy of such algebras between distributivity and strict-increasingness. The notion of local distributivity can be generalised as follows. Consider the total ordering implied by $\oplus$. Let D^k be the set of k^{th}-level distributive ranges defined as follows:

$$\begin{aligned}
&[a, b] \in D^0 \triangleq a = b \\
&[a, b] \in D^k \triangleq \forall x, y \in [a, b] : \forall f, h : [h(f(x) \oplus f(y)), h(f(x \oplus y))] \in D^{k-1}
\end{aligned}$$

It is hypothesised that if the range $[\overline{0}, \overline{\infty}]$ is k^{th}-level distributive then the path-vector protocol is guaranteed to converge in $\Theta(kn)$ iterations.

9.3.4 Translation to hard state protocols

Throughout this work, the asynchronous iterations have been modelled as soft-state processes where nodes are assumed exchange messages at regular intervals. In a hard-state protocol a node only sends information to its neighbours when the information changes. Given the generalisations to schedules described in Chapter 3 that relaxed the requirement that every node must continue to activate indefinitely, it should be relatively easily to provide a general translation from a hard-state protocol to a soft-state protocol.

When it comes to routing there are a few extra complexities that could be added to the hard-state model. For example BGP performs a process called rate limiting [29] where routing updates are not sent immediately whenever a change in a node's routing decision is made. Instead there is a delay before sending out an update and any further changes to the node's routing decisions during this time can be merged into one routing update. This has the effect of both saving bandwidth and damping down transient routing oscillations.

9.3.5 Alternative forms of vector-based protocols

Recently a new generation of distance-vector routing protocols, such as EIGRP [1] and Babel [15], have emerged that alter the standard vector-based algorithm in order to prevent the propagation of junk routes [24]. For instance Babel prevents routers from accepting routes that do not obey its *feasibility* conditions. Any route is feasible if accepting it cannot cause a forwarding loop. If accepting it could possibly cause a forwarding loop, then the router will use packet sequence numbers to request confirmation from the originating router that the route is safe.

Whilst the rejection of infeasible routes can be modelled by the current algebraic theory, the requests for confirmation cannot. It would be interesting to see if the model described in this thesis could be updated to apply to Babel and other similar protocols.

Bibliography

[1] R Albrightson, JJ Garcia-Luna-Aceves, and Joanne Boyle. EIGRP–a fast routing protocol based on distance vectors. Technical report, CISCO Systems, 1994.

[2] Anonymous. Agda tutorials, 2018. URL `https://wiki.portal.chalmers.se/agda/pmwiki.php?n=Main.Othertutorials`. Last accessed 07-12-2018.

[3] Roland C Backhouse and Bernard A Carré. Regular algebra applied to path-finding problems. *IMA Journal of Applied Mathematics*, 15(2):161–186, 1975.

[4] John S Baras and George Theodorakopoulos. Path problems in networks. *Synthesis Lectures on Communication Networks*, 3(1):1–77, 2010.

[5] Gérard M Baudet. Asynchronous iterative methods for multiprocessors. Technical report, Department of Computer Science, Carnegie-Mellon University, Pittsburgh, 1976.

[6] Ryan Beckett, Ratul Mahajan, Todd Millstein, Jitendra Padhye, and David Walker. Don't mind the gap: Bridging network-wide objectives and device-level configurations. In *Proceedings of the 2016 ACM SIGCOMM Conference*, 2016.

[7] Richard Bellman. On a routing problem. *Quarterly of applied mathematics*, 16(1): 87–90, 1958.

[8] Dimitri P Bertsekas, John N Tsitsiklis, et al. A survey of some aspects of parallel and distributed iterative algorithms. Technical report, Massachusetts Institute of Technology, Laboratory for Information and Decision Systems, 1989.

[9] Dimitri P Bertsekas, Robert G Gallager, and Pierre Humblet. *Data networks*, volume 2. Prentice-Hall International New Jersey, 1992.

[10] Stefano Bistarelli, Ugo Montanari, and Francesca Rossi. Soft constraint logic programming and generalized shortest path problems. *Journal of Heuristics*, 8(1):25–41, 2002.

[11] Juan F Botero, Miguel Molina, Xavier Hesselbach-Serra, and José R Amazonas. A novel paths algebra-based strategy to flexibly solve the link mapping stage of VNE problems. *Journal of Network and Computer Applications*, 36(6):1735–1752, 2013.

[12] Richard L Burden and J. Douglas Faires. *Numerical analysis*. Cengage Learning, 1985.

[13] Bernard A Carré. An algebra for network routing problems. *IMA Journal of Applied Mathematics*, 7(3):273–294, 1971.

[14] Henri Casanova, Michael G Thomason, and Jack J Dongarra. Stochastic performance prediction for iterative algorithms in distributed environments. *Journal of Parallel and Distributed Computing*, 58(1):68–91, 1999.

[15] Juliusz Chroboczek. The BABEL routing protocol, RFC 6126. *Quagga Routing Software Suite, GPL licensed*, 2011.

[16] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. *Introduction to Algorithms, Third Edition*. The MIT Press, 3rd edition, 2009. ISBN 0262033844, 9780262033848.

[17] Matthew L Daggitt and Timothy G Griffin. Rate of convergence of increasing path-vector routing protocols. In *ICNP proceedings*, 2018.

[18] Matthew L Daggitt, Alexander J T Gurney, and Timothy G Griffin. Asynchronous convergence of policy-rich distributed Bellman-Ford routing protocols. In *SIGCOMM proceedings*. ACM, 2018.

[19] Matthew L Daggitt, Ran Zmigrod, and Timothy G. Griffin. Routing library, 2019. URL `https://github.com/MatthewDaggitt/agda-routing/tree/thesis`.

[20] Edsger W Dijkstra. A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271, 1959.

[21] Lester R Ford Jr. Network flow theory. Technical report, RAND CORP SANTA MONICA CA, 1956.

[22] Andreas Frommer and Daniel B Szyld. On asynchronous iterations. *Journal of computational and applied mathematics*, 123(1):201–216, 2000.

[23] Lixin Gao and Jennifer Rexford. Stable internet routing without global coordination. *IEEE/ACM Transactions on Networking (TON)*, 9(6):681–692, 2001.

[24] Jose J Garcia-Lunes-Aceves. Loop-free routing using diffusing computations. *IEEE/ACM Transactions on Networking (TON)*, 1(1):130–141, 1993.

[25] Jonathan S Golan. *Semirings and their Applications.* Springer Science & Business Media, 2013.

[26] Michel Gondran and Michel Minoux. *Graphs, dioids and semirings: new models and algorithms*, volume 41. Springer Science & Business Media, 2008.

[27] Timothy G Griffin. The stratified shortest-paths problem. In *COMSNETS*, pages 1–10, 2010.

[28] Timothy G Griffin and Geoff Huston. BGP wedgies. RFC 4264, The Internet Engineering Taskforce, 2005. URL `http://www.ietf.org/rfc/rfc4264.txt`.

[29] Timothy G Griffin and Brian J Premore. An experimental analysis of BGP convergence time. In *Network Protocols, 2001. Ninth International Conference on*, pages 53–61. IEEE, 2001.

[30] Timothy G Griffin and Joäo L Sobrinho. Metarouting. In *ACM SIGCOMM Computer Communication Review*, volume 35, pages 1–12. ACM, 2005.

[31] Timothy G Griffin and Gordon Wilfong. Analysis of the MED oscillation problem in BGP. In *Proceedings of 10th IEEE International Conference on Network Protocols*, pages 90–99. IEEE, 2002.

[32] Timothy G Griffin, F Bruce Shepherd, and Gordon Wilfong. Policy disputes in path-vector protocols. In *Network Protocols, 1999.(ICNP'99) Proceedings. 7th International Conference on*, pages 21–30. IEEE, 1999.

[33] Timothy G Griffin, F Bruce Shepherd, and Gordon Wilfong. The stable paths problem and interdomain routing. *IEEE/ACM Transactions on Networking (ToN)*, 10(2): 232–243, 2002.

[34] Alexander J T Gurney. *Construction and verification of routing algebras.* PhD thesis, University of Cambridge, 2009. URL `http://agurney.net/pubs/Gurney09`.

[35] Alexander J T Gurney. Asynchronous iterations in ultrametric spaces. *arXiv preprint arXiv:1701.07434*, 2017.

[36] Alexander J T Gurney and Timothy G Griffin. Lexicographic products in metarouting. In *2007 IEEE International Conference on Network Protocols*, pages 113–122. IEEE, 2007.

[37] Pascal Hitzler and Anthony Seda. *Mathematical Aspects of Logic Programming Semantics.* CRC Press, 2016.

[38] Geoff Huston. Interconnection, peering and settlements: Parts I & II. *Internet Protocol Journal (Cisco)*, 2, 1999.

[39] Geoff Huston. IPv6 – evolution or revolution. *The ISP Column*, 2006.

[40] Howard Karloff. On the convergence time of a path-vector protocol. In *Proceedings of the fifteenth annual ACM-SIAM symposium on Discrete Algorithms*, pages 605–614. Society for Industrial and Applied Mathematics, 2004.

[41] Anatole Katok and Boris Hasselblatt. *Introduction to the modern theory of dynamical systems*, volume 54. Cambridge university press, 1995.

[42] Craig Labovitz, G Robert Malan, and Farnam Jahanian. Internet routing instability. *IEEE/ACM Transactions on Networking (TON)*, 6(5):515–528, 1998.

[43] Franck Le, Geoffrey G Xie, and Hui Zhang. Theory and new primitives for safely connecting routing protocol instances. *ACM SIGCOMM Computer Communication Review*, 41(4):219–230, 2011.

[44] Thomas Lengauer and Dirk Theune. Unstructured path problems and the making of semirings. In *Workshop on Algorithms and Data Structures*, pages 189–200. Springer, 1991.

[45] Mingming Lu and Jie Wu. Opportunistic routing algebra and its applications. In *INFOCOM 2009, IEEE*, pages 2374–2382. IEEE, 2009.

[46] Bruce M Maggs and Serge A Plotkin. Minimum-cost spanning tree as a path-finding problem. *Information Processing Letters*, 26(6):291–293, 1988.

[47] Gary Malkin. RFC 2453: RIP version 2. *Accessed November*, 1998.

[48] Alberto Martelli. A Gaussian elimination algorithm for the enumeration of cut sets in a graph. *Journal of the ACM (JACM)*, 23(1):58–73, 1976.

[49] Danny McPherson, Vijay Gill, Daniel Walton, and Alvaro Retana. Border gateway protocol (BGP) persistent route oscillation condition. RFC 3345, The Internet Engineering Taskforce, 2002. URL `http://www.ietf.org/rfc/rfc3345.txt`.

[50] J Miellou, Didier El Baz, and Pierre Spiteri. A new class of asynchronous iterative algorithms with order intervals. *Mathematics of Computation of the American Mathematical Society*, 67(221):237–255, 1998.

[51] John Milnor. On the concept of attractor. In *The theory of chaotic attractors*, pages 243–264. Springer, 1985.

[52] Mehryar Mohri. Semiring frameworks and algorithms for shortest-distance problems. *Journal of Automata, Languages and Combinatorics*, 7(3):321–350, 2002.

[53] Ulf Norell. Dependently typed programming in Agda. In *Proceedings of the 4th International Workshop on Types in Language Design and Implementation*, 2009.

[54] Jon Postel. User datagram protocol. *RFC 768*, 1980.

[55] Jon Postel. Transmission control protocol specification. *RFC 793*, 1981.

[56] Sibylla Priess-Crampe and Paulo Ribenboim. Fixed points, combs and generalized power series. In *Abhandlungen aus dem Mathematischen Seminar der Universität Hamburg*, volume 63, pages 227–244. Springer, 1993.

[57] Yakov Rekhter, Tony Li, and Susan Hares. A border gateway protocol 4 (BGP-4). RFC, The Internet Engineering Taskforce, 2005.

[58] François Robert. *Discrete iterations: a metric study*, volume 6. Springer Science & Business Media, 2012.

[59] Robert W Rosenthal. A class of games possessing pure-strategy nash equilibria. *International Journal of Game Theory*, 2(1):65–67, 1973.

[60] Günter Rote. A systolic array algorithm for the algebraic path problem (shortest paths; matrix inversion). *Computing*, 34(3):191–219, 1985.

[61] Alexander Schrijver. On the history of the shortest path problem. *Documenta Mathematica*, 17:155, 2012.

[62] João L Sobrinho. An algebraic theory of dynamic network routing. *IEEE/ACM Transactions on Networking (TON)*, 13(5):1160–1173, 2005.

[63] Aydin Üresin and Michel Dubois. Generalized asynchronous iterations. In *International Conference on Parallel Processing*, pages 272–278. Springer, 1986.

[64] Aydin Üresin and Michel Dubois. Parallel asynchronous algorithms for discrete data. *Journal of the ACM (JACM)*, 37(3):588–606, 1990.

[65] Aydin Üresin and Michel Dubois. Effects of asynchronism on the convergence rate of a class of iterations. Technical report, CENG 95-05, USC, 1995.

[66] Kannan Varadhan, Ramesh Govindan, and Deborah Estrin. Persistent route oscillations in inter-domain routing. *Computer networks*, 32(1):1–16, 2000.

[67] Various. Agda standard library, 2019. URL `https://github.com/agda/agda-stdlib`. Version 1.0.

[68] Ran Zmigrod, Matthew L Daggitt, and Timothy G Griffin. An Agda formalization of Üresin & Dubois' asynchronous fixed-point theory. *9th International Conference on Interactive Theorem Proving (ITP)*, July 2018.

Appendix A

General theory

This appendix covers some of the more common mathematical concepts used in the main thesis. The definitions are standard unless stated otherwise.

A.1 Sets

These common sets are used throughout:

- $\mathbb{N}$ is the set of natural numbers.
- $\mathbb{N}^\infty = \mathbb{N} \cup \{\infty\}$.
- $\mathbb{R}$ is the set of real numbers.
- $\mathbb{R}^+ = \{x \geq 0 \mid x \in \mathbb{R}\}$.
- S is used as an abstract set.
- 2^S is the power set of S.
- S^n is the set of vectors of size n over the set S.
- $S^{n \times n}$ is the set of $n \times n$ matrices over the set S.

A.2 Operators

Given a binary operator $\oplus$ over a set S, some common properties are as follows:

> **Definition 42.** $\oplus$ is *associative* if $\forall x, y, z : (x \oplus y) \oplus z = x \oplus (y \oplus z)$.

Definition 43. $\oplus$ is *commutative* if $\forall x, y : x \oplus y = y \oplus x$.

Definition 44. $\oplus$ is *idempotent* if $\forall x : x \oplus x = x$.

Definition 45. $\oplus$ is *selective* if $\forall x, y : x \oplus y \in \{x, y\}$.

Definition 46. e is an *identity* for $\oplus$ if $\forall x : x \oplus e = x = e \oplus x$.

Definition 47. e is an *annihilator* for $\oplus$ if $\forall x : x \oplus e = e = e \oplus x$.

A.3 Relations

Given a binary relation $\leq$ over a set S, some common properties are as follows:

Definition 48. $\leq$ is *reflexive* if $\forall x : x \leq x$.

Definition 49. $\leq$ is *transitive* if $\forall x, y, z : x \leq y \wedge y \leq z \Rightarrow x \leq z$.

Definition 50. $\leq$ is *antisymmetric* if $\forall x, y : x \leq y \wedge y \leq x \Rightarrow x = y$.

Definition 51. $\leq$ is *total* if $\forall x, y : x \leq y \vee y \leq x$.

Definition 52. e is a *minimal element* for $\leq$ if $\forall x : e \leq x$.

Definition 53. e is a *maximal element* for $\leq$ if $\forall x : x \leq e$.

Definition 54. $<$ is *dense* if $\forall x, y : x < y \Rightarrow \exists z : x < z < y$.

Definition 55. $\leq$ is a *total ordering* if $\leq$ is reflexive, transitive, antisymmetric and total.

There is a correspondence between total relations and selective functions. Given a selective operator $\oplus$, the relation, $\leq$, can be defined as

$$x \leq y \triangleq x \oplus y = x$$

and given a total relation $\leq$, the natural choice operator, $\oplus$, can be defined as:

$$x \oplus y = \begin{cases} x & \text{if } x \leq y \\ y & \text{otherwise} \end{cases}$$

The properties translate as follows:

$\oplus$ is associative	$\Rightarrow$	$\leq$ is transitive
$\oplus$ is commutative	$\Rightarrow$	$\leq$ is antisymmetric
$\oplus$ is idempotent	$\Rightarrow$	$\leq$ is reflexive
$\oplus$ is selective and commutative	$\Rightarrow$	$\leq$ is total
$\oplus$ is associative	$\Leftarrow$	$\leq$ is transitive and antisymmetric
$\oplus$ is commutative	$\Leftarrow$	$\leq$ is antisymmetric
$\oplus$ is idempotent	$\Leftarrow$	$\leq$ is reflexive
$\oplus$ is selective	$\Leftarrow$	$\leq$ is total
e is an identity for $\oplus$	$\Leftrightarrow$	e is a minimal element for $\leq$
e is an annihilator for $\oplus$	$\Leftrightarrow$	e is a maximal element for $\leq$

This thesis works with total orderings for the most part, and so this equivalence can be used to swap between the two interchangeably depending on which form is more convenient to reason with.

A.4 Functions

> **Definition 56.** An element e is a *fixed point* for function f if $f(e) = e$.

> **Definition 57.** A set S is *closed* over a function f if for all $x \in S$ then $f(x) \in S$.

> **Definition 58.** A *quasi-semi-metric* over a set S is a function d that satisfies:
>
> 1. : $d(x, y) = 0 \Leftrightarrow x = y$

Definition 59. A *quasi-metric* over a set S is a function d that satisfies:

1. $d(x, y) = 0 \Leftrightarrow x = y$
2. $d(x, y) = d(y, x)$

Definition 60. A *metric* over a set S is a function d that satisfies:

1. $d(x, y) = 0 \Leftrightarrow x = y$
2. $d(x, y) = d(y, x)$
3. $d(x, z) \leq d(x, y) + d(y, z)$

Definition 61. An *ultrametric* over a set S is a function d that satisfies:

1. $: d(x, y) = 0 \Leftrightarrow x = y$
2. $: d(x, y) = d(y, x)$
3. $: d(x, z) \leq \max(d(x, y), d(y, z))$

Definition 62. A distance function d is *bounded* if:

$$\exists\ d_{max} : \forall x, y : d(x, y) \leq d_{max}$$

A.5 Bachmann-Landau notation

The rate of growth of functions can be described using Bachmann-Landau notation. Let f and g be functions from $\mathbb{N} \to \mathbb{R}^+$ then:

Definition 63. $f(n) \in \Omega(g(n))$ if there exists a $c \in \mathbb{R}^+$ and $k \in \mathbb{N}$ such that for all $n \geq k$ then $cg(n) \leq f(n)$ – i.e. f's growth is asymptotically bounded below by g.

Definition 64. $f(n) \in \Theta(g(n))$ if there exists $c, d \in \mathbb{R}^+$ and $k \in \mathbb{N}$ such that for all $n \geq k$ then $cg(n) \leq f(n) \leq dg(n)$ – i.e. f's growth is asymptotically the same as g.

Definition 65. $f(n) \in O(g(n))$ if there exists a $c \in \mathbb{R}^+$ and $k \in \mathbb{N}$ such that for all $n \geq k$ then $f(n) \leq cg(n)$ – i.e. f's growth is asymptotically bounded above by g.