Computational analysis of complex systems: Discrete foundations, algorithms, and the Internet

Habilitation zur Erlangung der
Lehrbefähigung für die Informatik
der Technischen Universität München

vorgelegt von

Sven Kosub

aus

Rudolstadt

München, Mai 2007

Eingereicht am: 10.05.2007

bei dem Habilitationsamt
der Technischen Universität München

1. Gutachter: Prof. Dr. Ernst W. Mayr
2. Gutachter: Prof. Dr. Anja Feldmann

Preface

I am deeply grateful to all which have contributed to this work by their comments and discussions, by hints, advice, and patience. In particular, I would like to thank Ernst W. Mayr, Moritz G. Maaß, Benjamin Hummel, Hanjo Täubig, Christopher M. Homan, Stefan Eckhardt, Olaf Maennel, Arne Wichmann, Thomas Schwabe, Alexander Offtermatt-Souza, Noam Nisan, Moshe Babaioff, Liad Blumrosen, Ilan Petrov, Ahuva Mu'alem, Ron Lavi, Jennifer Rexford, Wolfgang Mühlbauer, Harald Schiöberg, and Vinay Aggarwal. For allowing me to present joint work here, many thanks are due to my coauthors and colleagues Christopher M. Homan, Benjamin Hummel, Moritz G. Maaß, and Hanjo Täubig.

The algorithms group ("LEA") at the Technische Universität München has always provided an inspiring and warm research atmosphere. I thank all group members sharing valuable lifetime with me in the last six years. In alphabetical order, I would like to thank my not-yet-mentioned colleagues and former colleagues Matthias Baumgart, Thomas Bayer, Arno Buchner, Dmytro Chibisov, Jens Ernst, Nicolas Fournier, Victor Ganzha, Stefanie Gerke, Jan Griebsch, Alexander Hall, Volker Heun, Klaus Holzapfel, Martin Marciniszyn, Werner Meixner, Stephan Micklitz, Michal Mnuk, Johannes Nowak, Stefan Pfingstl, Martin Raab, Jan Remy, Ingo Rohloff, Ulrich Rührmair, Sandeep Sadanandan, Mark Scharbrodt, Christian Scheideler, Thomas Schickinger, Angelika Steger, Peter Ullrich, Ulrich Voll, Andreas Weißl, and Sebastian Wernicke.

Furthermore, I am greatly indebted to the Minerva Foundation of the Max-Planck-Gesellschaft for granting financial support for a research stay at The Hebrew University of Jerusalem where some of the initial results of this thesis could be obtained.

Table of Contents

List of Figures

1. Introduction

As large-scale socio-technical systems have grown in relevance for critical infrastructure, there is an increasing need for tools to control the functionalities at all system levels. The far-reaching sequence of system incidents in the continental European electricity grid on November 4, 2006, by the disconnection of an extra high voltage line over the Ems River, which finally separated the grid into three disjoint parts and affected 15 million households with significant power imbalance, demonstrates exemplarily how small local changes can lead to drastic global system consequences [48, 31]. Avoiding such negative implications through effective methods of forecasting and intervening is a central and not-solved theme of complex systems research.

In view of the possibly immense costs caused by system failures, identifying the responsible origins afterwards is another aspect of importance. Particularly in decentralized environments, such as communication networks, the interpretation of monitored data and the inference of the relevant system information is often a non-trivial problem. As *the inherent contingency of complex systems excludes the total avoidance of unintended system behavior*—notably this applies to systems with a high degree of social or economic interactions—, reliable and fast *ex post* analysis techniques are fundamentally required to contain damages, such as network downtimes in the Internet after link failures.

Motivated by these challenges, we address in this thesis questions concerning a computational understanding of real-world complex systems inspired by the ex post perspective. The typical problem in a *retrospective* ex post analysis is an inference problem, i.e., given a set of observations in a credible model of the complex system under study, which unobservable or non-monitored entities, events, or data in the past could have led to the set of observations? A *prospective* ex post analysis would ask further which events are to be expected based on the inferred data from the past. The approaches in this thesis relate to the retrospective ex post analysis.

We make both theoretical and application-oriented contributions to the computational analysis of complex systems. As a theoretical contribution we present, discuss, and investigate a mathematical framework capable to serve as a discrete foundation for ex post analyses. The specific empirical point of view taken by the ex post perspective is reflected by the following principles:

– Each component of a dynamical system is a possible subject of observation.
– Observations are finite and extensional.

A consequence of these principles is that strong system idealizations, such as synchronicity or uniformity, which are prerequisite to standard models of complex systems should be avoided. The principles characterize the analysis setup as a bottom-up approach, i.e., coming from

the data, in difference to the top-down approach, i.e., coming from the models, which is pre-dominant in formal specification and model checking.

The application representative for theory building is Internet inter-domain routing using the Border Gateway Protocol (BGP). We provide an in-depth study of the inference of business relationships from sets of observations. Algorithmic solutions for this problem are essential ingredients to the identification of link failures and, more generally, an understanding of the dynamics of the BGP route propagation process.

In the following we give a short overview on the contributions of this thesis and their positions in a larger context.

Discrete foundations

When, quite generally, a system is "any collection of interdependent and interacting elements which act together in a mutual effort to achieve some (usually specifiable) goal"[109], then

- the goal defines the functionality of the system as a whole,
- the way in which the elements act and interact determines the dynamics of the system, i.e., the sequences of operations to achieve patterns associated with functionality, and
- the degree of interdependence of the elements defines the complexity of a system.

A complex system is certainly a system with high degree of interdependence.

System complexity has at least two facets—quantity and quality. The quantitative aspect of complexity is measured by the number of elements that interact in a system. The qualitative aspect of complexity is measured by the organizational arrangement of the interdependences of elements. Accordingly, complexity has been divided into disorganized and organized complexity [161]. Purely disorganized complexity is characterized by a very large, i.e., metaphorically an astronomical, number of elements which are more or less individually independent. Methods of probability theory and statistical mechanics are appropriate for solving problems of disorganized complexity. Problems of organized complexity "will often involve a considerable number of variables. The really important characteristic ... lies in the fact that these problems ... show the essential feature of organization"[161]. The systems addressed in this thesis are systems with high organized complexity (and often significant disorganized complexity).

Consistently, a general model of discrete complex systems has at least three components [8, 16]:

- a graph representing the structural interdependencies, i.e., how the elements are organizationally embedded into the local neighborhood within a system
- a set of local transition functions representing the functional interdependencies, i.e., how the elements act in response to actions of their neighborhood
- an activation sequence of subsystems, formally a mapping from a discrete time-scale to sets of elements, representing the dynamics of a system, i.e., in which order elements are allowed to act according to their transition functions

This conception provides enough modelling and observation flexibility to meet the requirements exposed by the principles of ex post analyses.

A criterion for the organizedness of a system is modularity. In a system with high organized complexity, many interactions regularly involve only elements of certain groups. Such groups

form operationally closed subsystems or modules. Speaking systems-theoretically, modularization is the evolutionary answer of a system to an increasing environmental complexity (i.e., the complexity of a larger system, the system is embedded into); if the external complexity exceeds the system's capacity of manageable complexity, it tends to react with forming operationally closed modules with differentiated subfunctionalities [139]. The establishment of Autonomous Systems as a second level in the Internet routing hierarchy is an idealtypical example how modularity reduces the environmental complexity for systems, in this case for routers.

Suitable frameworks for system classifications should respect the modular structure of complex systems, i.e., their operational closedness. The approach chosen in this thesis is to consider network classes closed under graph transformation rules—concretely, the minor relation—and function classes closed under superposition—concretely, we employ clone theory. We mention that for activation sequences (or update schedules) there is not yet an appropriate mathematical theory available. Complex systems can be grouped into several families depending on the class-membership of their components. As we will see, the proposed framework allows exhaustive, meaningful, and sharp distinctions between system classes with respect to computational analyzability.

The method of choice for any analysis of complex systems in general is a computer simulation [109]. As such simulation of real-world system are usually costly and time-consuming the question arises under which circumstances a simulation can be "short-cut". A simulation short-cut is any algorithm producing the outcome of a simulation faster than executing a step-by-step simulation does. Typically, simulation short-cuts are identified with polynomial algorithms [28, 148]. Hence, an important goal in the analysis of complex systems is a comprehensive catalogue of system classes paired with polynomial-time algorithms solving a certain analysis problem for all systems within the class. Such catalogues are only known for very restricted system classifications. In this thesis we will completely describe the "islands of tractability" for finding and counting fixed points in discrete dynamical systems with binary action sets.

The Internet

An idealtypical example of a complex system is the Internet. By its very architecture, it combines to a significant extent social interaction with technical infrastructure and environments. Additionally, due to its sheer size, many characteristic phenomena of complexity, such as modularization and contingent behavior, can be observed.

A fundamental functionality of the Internet is the provision of transmission paths over routers from any source to any destination where devices are connected to the Internet. Early protocols, such as RIP [75, 103] and OSPF [115, 116], guarantee the selection of shortest paths between any pair of devices. Relying on these protocols, with an increasing number of participants, routers become congested by communication overhead caused by the dynamics of route dissemination or some of them by heavy load due to a high betweenness centrality, i.e., the number of shortest paths a router is part of. A solution to these two problems—scalability and administrative autonomy [93]—is provided by the Border Gateway Protocol (BGP) [128, 127].

Based on modularization BGP controls the routing problem on a higher level. Routers and hosts are aggregated in networks or Autonomous Systems (ASes) which are typically

formed due to their geographical settlements. An AS is operationally closed as it has a self-contained intra-domain (intra-AS) routing strategy which can differ from AS to AS. Routes through the Internet can now be considered from the viewpoint of which ASes they traverse. We obtain a new routing problem at the AS level—inter-domain (inter-AS) routing—which however involves a much smaller graph.

BGP solves the inter-domain routing problem in the following way: For some special destination, each AS exports a complete, *best* route, not only the distance, to those neighbors that have been selected according to a local routing policy. Depending on the routes received, a new best route may be chosen and, if this is the case, a new wave of route propagations starts. If no AS selects a new best route, the process ends (for this special destination). Which neighbors are selected by the local policy for best-route exports is based on subjective and private preferences typically determined by contractual business relationships among neighbored ASes.

The influence of routing policies on the dynamics of BGP route propagation is the subject of many research activities but is still poorly understood (see, e.g., [73, 136, 96, 71, 95, 94, 144, 102, 55]). There are mathematical reasons why BGP cannot give guarantees for the convergence of the route propagation process [67, 73]. The lack of convergence guarantees significantly impacts Internet route stability in the case of faults. Several experiments have pointed out that inter-domain routers, after some fault has happened, may take several minutes to overcome temporary routing table fluctuations generated by the BGP path selection process [95, 94]. To what extent such delayed convergence can be caused by protocol divergence is not clear. We even have difficulties to observe protocol divergence [73]. This can be attributed to our current inability to algorithmically solve the following peculiar change-analysis problem: Given a set of observed BGP updates, which AS has originated the first update message leading to the observations?

Despite some architectural and methodological approaches [67, 55] satisfactory solutions to change analysis in the Internet are currently out of reach. In this thesis we concentrate on the AS relationship inference as an important subproblem. Because the policy-influencing business contracts between ASes are subject to privacy, precise algorithms for uncovering this information from observable BGP routes are requested. Here, a purely combinatorial approach to this problem is presented. We introduce notions of acyclicity to the interrelationship graph and investigate the complexity of the Acyclic Type-of-Relationship problem for various problem specifications. Tests on real-world data show that the algorithms obtained produce the least number of misclassifications compared to existing standard algorithms. Moreover, the experimental findings demonstrate that acyclicity assumptions should be an integral part of interrelationship models.

Organization of the thesis

This thesis is divided into two parts. In the first part we propose and investigate our mathematical framework for finite, discrete dynamical systems

– Chapter 3 contains a detailed description of our model for finite dynamical systems and its components. We give introductions to relevant parts of the theory of graph minors and clone theory. We also discuss the pograph approach to update schedules which is useful for black-box simulations of systems.

– In Chapter 4 we prove dichotomy theorems for the complexity of the Fixed-Point Existence and the Fixed-Point Counting problem for boolean dynamical systems. We make a distinction between lookup-table, formula, and circuit representations. We also discuss the possibility of extending dichotomy theorems to larger domains.

The second part is devoted to the Internet routing as a case study in complex systems.

– Chapter 5 contains a systematic and comprehensive introduction to the Border Gateway Protocol.
– In Chapter 6 we present our combinatorial approach to the AS Relationship Inference problem. Based on the introduced acyclicity conditions we intensively study the Acyclic Type-of-Relationship problem (in 56 problem versions). Furthermore, a heuristic for finding good orientations on real-world data is designed. We complete the study with an experimental section.

Chapter 2 provides the necessary mathematical notations and useful concepts.

Publications

Original contributions presented in this thesis are taken from the following research papers:

[81] B. F. Hummel and S. Kosub. Acyclic type-of-relationship problems on the Internet: An experimental analysis.
Technical Report TUM-I0709, Institut für Informatik, Technische Universität München, February 2007. Submitted for publication.

[88] S. Kosub. Dichotomy results for the fixed-point existence problem in boolean dynamical systems.
Technical Report TUM-I0701, Institut für Informatik, Technische Universität München, January 2007. Submitted for publication.

[89] S. Kosub and C. M. Homan. Dichotomy results for fixed point counting in boolean dynamical systems.
Technical Report TUM-I0706, Institut für Informatik, Technische Universität München, January 2007. Submitted for publication.

[90] S. Kosub, M. G. Maaß, and H. Täubig. Acyclic type-of-relationship problems on the Internet.
In *Proceedings of the 3rd Workshop on Combinatorial and Algorithmic Aspects of Networking (CAAN'2006)*, volume 4235 of *Lecture Notes in Computer Science*, pages 98–111. Springer-Verlag, Berlin, 2006.
A complete version is available as Technical Report TUM-I0605, Institut für Informatik, Technische Universität München, March 2006 [91].

The following parts of the thesis are covered by these papers:

– Most of the results of Chapter 4 are contained in [88, 89]; Section 4.3 is from [88] (except Subsection 4.3.4 which is solely written for this thesis) and Section 4.4 is from [89].
– The content of Chapter 6 is contained in [90, 91] (Section 6.3 and Section 6.4) and in [81] (Section 6.5 and Section 6.6)

The remaining parts are solely written for this thesis. Each result or definition without any contribution by the author is given a reference to the relevant literature.

2. Fundamentals

In this chapter we discuss relevant terminology and notation for sets, relations, and graphs, some fundamental algorithms, and a few other mathematical preliminaries.

2.1 Sets and relations

We denote the set of integers by $\mathbb{Z}$, the set of non-negative integers by $\mathbb{N}$, and the set of positive integers by $\mathbb{N}_+$. $\mathbb{Z}_2$ denotes the Galois field GF[2].

Sets

The empty set is denoted by $\emptyset$. For an arbitrary set A, $\mathcal{P}(A)$ denotes the power set of A, i.e., the family of all subsets of A, and $\mathcal{P}_+(A)$ denotes the set $\mathcal{P}(A) \setminus \{\emptyset\}$. For an arbitrary finite set A, its cardinality is denoted by $\|A\|$. Let A and B be any sets. Then $A \setminus B$ denotes the difference of A with B, i.e., the set of all elements that are in A but not in B. $A \times B$ denotes the cartesian product, i.e, the set of all pairs (a, b) with $a \in A$ and $b \in B$. For $m \in \mathbb{N}_+$, define $A^m =_{\text{def}} \underbrace{A \times \cdots \times A}_{m \text{ times}}$. Let M be any fixed basic set. For a set $A \subseteq M$, its complement in the basic set M is denoted by $\overline{A}$, i.e., $\overline{A} = M \setminus A$. A multiset A is allowed to contain elements many times. The multiplicity of an element x in a multiset A is the number of occurrences of x in A. The cardinality of a multiset A is also denoted by $\|A\|$.

Functions

Let M and M' be any sets, and let $f : M \to M'$ by any function. The domain of f which we denote by D_f is the set of all $x \in M$ such that $f(x)$ is defined. A function f is total if the domain of f is M. For a set $A \subseteq D_f$, let $f(A) = \{f(x) \mid x \in A\}$ denote the image of A under f. In particular, the range of f which is denoted by R_f is the set $f(D_f)$. For a set $A \subseteq M$, the restriction of a total function f to A is denoted by $f[A]$. The inverse of f is denoted by f^{-1}, i.e, $f^{-1} : M' \to \mathcal{P}(M)$ such that for all $y \in M'$, $f^{-1}(y) = \{x \in M \mid f(x) = y\}$. If $f^{-1}(y)$ is at most a singleton then we omit the braces. The pre-image of A under f is the set $f^{-1}(A) = \{x \in M \mid f(x) \in A\}$.

We use two notations for composition of functions. If f and f' are functions with $f : M \to M'$ and $f' : M' \to M''$, then $(f' \circ f)$ is the function mapping from M to M'' which is defined for all $x \in M$ as $(f' \circ f)(x) =_{\text{def}} f'(f(x))$. In contrast, we use $f \cdot f'$ to denote $f' \circ f$.

A function $f : M \to M'$ is bijective if f is surjective, i.e., $R_f = M'$ and injective, i.e., for all $y \in R_f$, $f^{-1}(y)$ is a singleton. Suppose $M' = M$ and M is finite. In this case a bijective

function f is a permutation. Suppose $M = \{1, 2, \ldots, n\}$. A *cycle* $(i_1\ i_2\ \ldots\ i_k)$ of length k of the permutation $\pi : M \to M$ is a sequence $(i_1, i_2, \ldots, i_k)$ such that $\pi(i_j) = i_{j+1}$ for $1 \leq j < k$ and $\pi(i_k) = i_1$. Each permutation allows a decomposition into cycles.

Orders

In more detail the following can be found in any textbook (e.g., [69, 38]) about theory of orders and lattices.

Let P be any set. A *partial order* on P (or order, for short) is a binary relation $\leq$ on P that is reflexive, antisymmetric, and transitive. The set P equipped with a partial order $\leq$ is said to be a *partially ordered set* (for short, *poset*). Usually, we talk about the poset P. Where it is necessary we write $(P, \leq)$ to specify the order. A poset P is a *chain* if for all $x, y \in P$ it holds that $x \leq y$ or $y \leq x$ (i.e., any two elements are comparable with respect to $\leq$). Such an order is also called a *total order*. A poset P is an *antichain* if for all $x, y \in P$ it holds that $x \leq y$ implies that $x = y$ (i.e., no two elements are comparable with respect to $\leq$).

We consider $\mathbb{N}$ to be ordered by standard total order on the natural numbers. If a set A is partially ordered by $\leq$ then A^m can be considered to be ordered by the *vector-ordering*, i.e., $(x_1, \ldots, x_m) \leq (y_1, \ldots, y_m)$ if and only if for all $i \in \{1, \ldots, m\}$, $x_i \leq y_i$.

An important tool for representing posets is the *covering relation* $\prec$. Let P be a poset and let $x, y \in P$. We say that x is covered by y (or y covers x), and write $x \prec y$, if $x < y$ and $x \leq z < y$ implies that $x = z$. The latter condition is demanding that there be no element z of P with $x < z < y$. A finite poset P can be drawn in a diagram consisting of points (representing the elements of P) and interconnecting lines (indicating the covering relation) as follows: To each element x in P associate a point $P(x)$ in the picture which is above all points $P(y)$ associated to elements y less than x, and connect points $P(x)$ and $P(y)$ by a line if and only if $x \prec y$. A poset can have different representation by diagrams.

Let P and P' be posets. A map $\varphi : P \to P'$ is said to be *monotone* (or order-preserving) if $x \leq y$ in P implies $\varphi(x) \leq \varphi(y)$ in P'. We say that φ is an *(order-)isomorphism* if φ is monotone, injective, and surjective. Two posets P and P' are *isomorphic*, in symbols $P \cong P'$, if there exists an isomorphism $\varphi : P \to P'$. Isomorphic poset shall be considered to be not essentially different: Two finite posets are isomorphic if and only if they can be drawn with identical diagrams.

Words

Sometimes we make no difference between m-tuples $(x_1, \ldots, x_m)$ over a finite set M and words $x_1 \ldots x_m$ of length m over M. Such finite sets are called alphabets. Let Σ be a finite alphabet. Σ^* is the set of all finite words that can be built with letters from Σ. For $x, y \in \Sigma^*$, $x \cdot y$ (or xy for short) denotes the concatenation of x and y. The empty word is denoted by ε. For a word $x \in \Sigma^*$, $|x|$ denotes the length of x. For $n \in \mathbb{N}$, Σ^n is the set of all words $x \in \Sigma^*$ such that with $|x| = n$. For a word $x = x_1 \ldots x_n \in \Sigma^*$ any word $x_1 \ldots x_k$ such that $k \leq n$ is called a prefix of x. We use regular expressions to describe subsets of Σ^* (see, e.g., [77]).

2.2 Graph theory

A graph $G = (V, E)$ consists of a set V of vertices and a set E of edges joining pairs of vertices. The vertex set and edge set of a graph G are denoted by $V(G)$ and $E(G)$, respectively. The cardinality of V is usually denoted by n, the cardinality of E by m. If two vertices are joined by an edge, they are adjacent and we call them *neighbors*. Graphs can be undirected und directed. In undirected graphs, the order in which vertices are joined is irrelevant. An undirected edge joining vertices $u, v \in V$ is denoted by $\{u, v\}$. In directed graphs, each directed edge has an *origin* and a *destination*. An edge with origin $u \in V$ and destination $v \in V$ is represented by an ordered pair (u, v). For a directed graph $G = (V, E)$, the *underlying undirected graph* is the undirected graph with vertex set V that has an undirected edge between two vertices $u, v \in V$ if (u, v) or (v, u) is in E.

Multigraphs

In both undirected and directed graphs, we may allow the edge set E to contain the same edge several times, i.e., E can be a multiset. If an edge occurs several times in E, the copies of that edge are called *parallel edges*. Graphs with parallel edges are also called *multigraphs*. A graph is called *simple*, if each of its edges in contained in E only once, i.e., if the graph does not have parallel edges. An edge joining a vertex to itself, is called a *loop*. A graph is called *loopless* if it has no loops. In general, we assume all graphs to be loopless unless specified otherwise.

Degrees

The *degree* of a vertex v in an undirected graph $G = (V, E)$, denoted by d_v, is the number of edges in E joining v. If G is a multigraph, parallel edges are counted according to their multiplicity in E. The set of neighbors of v is denoted by $N(v)$. $N^0(v)$ denotes the vertex set $N(v) \cup \{v\}$. If the graph under consideration is not clear from the context, these notations can be augmented by specifying the graph as an index. For example, $N_G(v)$ denotes the neighborhood of v in G.

Subgraphs

A graph $G' = (V', E')$ is a *subgraph* of the graph $G = (V, E)$ if $V' \subseteq V$ and $E' \subseteq E$. Sometimes we denote this by $G' \subseteq G$. It is a *(vertex-)induced subgraph* if E' contains all edges $e \in E$ that join vertices in V'. The induced subgraph of $G = (V, E)$ with vertex set $V' \subseteq V$ is denoted by $G[V']$. The *(edge-)induced subgraph* with edge set $E' \subseteq E$, denoted by $G[E']$, is the subgraph $G' = (V', E')$ of G, where V' is the set of all vertices in V that are joined by at least one edge in E'.

Walks, paths, and cycles

A *walk* from x_0 to x_k in a graph $G = (V, E)$ is a sequence $x_0, e_1, x_1, e_2, x_2, \ldots, x_{k-1}, e_k, x_k$ alternating between vertices and edges of G, where $e_i = \{x_{i-1}, x_i\}$ in the undirected case and

$e_i = (x_{i-1}, x_i)$ in the directed case. The length of a walk is the number of edges on the walk. As shorthands we use $(x_0, x_1, \ldots, x_k)$ and $(e_1, e_2, \ldots, e_k)$ to denote a walk. The walk is called a *path* if $x_i \neq x_j$ for $i \neq j$. A walk with $x_0 = x_k$ is called a *cycle* if $e_i \neq e_j$ for $i \neq j$. A cycle is a *simple cycle* if $x_i \neq x_k$ for $0 \leq i < j \leq k-1$.

Special graphs

A *tree* is a connected (for a definition see below) undirected graph not containing a cycle. An undirected graph $G = (V, E)$ is called *complete* if it contains all possible pairs of vertices as edges. A complete graph with n vertices is denoted by K^n. A K^n is called a *clique*. A K^2 is a graph of two vertices with one edge joining them. A K^3 is also called a *triangle* or *triad*. A graph without edges is called *empty*. An *independent set* within a graph $G = (V, E)$ is a vertex set $U \subseteq V$ such that $G[U]$ is empty. A graph $G = (V, E)$ is called *bipartite* if there are independent vertex sets $V_1, V_2 \subseteq V$ such that V_1 and V_2 are disjoint and $V_1 \cup V_2 = V$. We denote by $E(V_1, V_2)$ the set of edges joining vertices from V_1 with vertices from V_2. If $E(V_1, V_2) = V_1 \times V_2$ then G is called a *complete bipartite graph*. Such a graph is denoted by K_{n_1,n_2} if V_1 consists of n_1 vertices and V_2 of n_2 vertices. A $K_{1,n}$ is also called a *star*. For two graphs $G = (V, E)$ and $G' = (V', E')$ we denote by $G \oplus G'$ the graph consisting of the disjoint union of the graphs G and G'.

Graph classes

Two graphs $G = (V, E)$ and $G' = (V', E')$ are *isomorphic*, denoted by $G \simeq G'$, if there is a bijective mapping $\varphi : V \to V'$ such that for all vertices $u, v \in V$ the following is true: in the case that G and G' are directed graphs it holds that $(u, v) \in E \Leftrightarrow (\varphi(u), \varphi(v)) \in E'$, and in the case that G and G' are undirected graphs it holds that $\{u, v\} \in E \Leftrightarrow \{\varphi(u), \varphi(v)\} \in E'$. A set of graphs is called a *graph class* if for each graph G in the class all graphs isomorphic to G belong to the class as well.

2.3 Algorithmics

Most results of this work relate to algorithms. In the following we mention essential problems and concepts which are needed more than once.

For two functions $f : \mathbb{N} \to \mathbb{N}$ and $g : \mathbb{N} \to \mathbb{N}$ we say that f is in $O(g)$ if there are constant $n_0, c \in \mathbb{N}_+$ such that for all $n \geq n_0$, $f(n) \leq c \cdot g(n)$. We say that f is in $\Omega(g)$ if g is in $O(f)$. We say that f is in $\Theta(g)$ if f is in $O(g) \cap \Omega(g)$.

Connected components

An undirected graph $G = (V, E)$ is *connected* if every vertex can be reached from every other vertex, i.e., if there is a path from every vertex to every other vertex. A graph consisting of a single vertex is also taken to be connected. Graphs that are not connected are called *disconnected*. For a given undirected graph $G = (V, E)$, a *connected component* of G is an induced subgraphs $G' = (V', E')$ that is connected and maximal, i.e., there is no connected subgraph $G'' = (V'', E'')$ such that $V'' \supset V'$. Checking whether a graph is connected and

finding all its connected components can be done in time $O(n+m)$ using depth-first search or breadth-first search.

A directed graph $G = (V, E)$ is *strongly connected* if there is a directed path from every vertex to every other vertex. A *strongly connected component* of a directed graph G is an induced subgraph that is strongly connected and maximal. The strongly connected components of a directed graph can be computed in time $O(n+m)$ using a depth-first search.

NP-*completeness*

It is important to consider the running-time of an algorithm for a given problem. Usually, one wants to give an upper bound on the running time of the algorithm for inputs of a certain size. If the running-time of an algorithm is $O(n^k)$ for some $k \in \mathbb{N}$ and for inputs of size n, we say that the algorithm runs in polynomial time. For graph problems, the running-time is usually specified as a function of n and m, the number of vertices and edges of the graph, respectively. For many problems, however, no polynomial-time algorithm has been discovered. Although one cannot rule out the possible existence of polynomial-time algorithms for such problems, the theory of NP-completeness provides means to give evidence for the computational intractability of a problem.

A decision problem is in the complexity class NP if there is a nondeterministic Turing machine that solves the problem in polynomial time. That is to say that the answer to a problem instance is "yes" if there exists a solution in the set of all possible solutions to the instance which is of polynomial size. Moreover, the test whether a potential solution is an actual solution must be performed in polynomial time. Note that a decision problem is usually considered to consist of the set of the "yes"-instances. A decision problem is NP-hard if every problem in NP can be reduced to it via a polynomial-time many-one reduction. (A polynomial-time many-one reduction from a set A to a set B is a function computable in polynomial time such that for all instances x, $x \in A \Leftrightarrow f(x) \in B$.) Problems that are NP-hard and belong to NP are called NP-*complete*. A polynomial-time algorithm for an NP-hard problem would imply polynomial-time algorithms for all problems NP—something that is considered very unlikely. Therefore, the NP-hardness of a problem is considered substantial evidence for the computational difficulty of the problem.

A standard example of an NP-complete problem is 3SAT, i.e., checking whether a given propositional formula given as a 3CNF has a satisfying assignment. To be more precise, a kCNF is a formula $H = C_1 \wedge \cdots \wedge C_m$ consisting of clauses C_i each of which has the form $C_i = l_{i1} \vee l_{i2} \vee \cdots \vee l_{ik}$ where l_{ij} is either a positive or a negative literal. A positive literal is some variable, say x_k, and a negative literal is the negation of some variable, say $\overline{x_k}$.

The class of complements of NP sets is denoted by coNP, i.e., coNP $= \{\overline{A} | A \in \text{NP}\}$.

For optimization problems (where the goal is to compute a feasible solution that maximizes or minimizes some objective function), we say that the problem is NP-*hard* if the corresponding decision problem (checking whether a solution with objective value better than a given value k exists) is NP-hard.

#P-*completeness*

A complexity class closely related to NP is the class #P which has been introduced in [153, 152] to provide evidence for the computational intractability of counting problems. The class

#P consists of all problems of the form "compute $f(x)$" where $f(x)$ is the number of accepting paths of a nondeterministic Turing machine running in polynomial time. Equivalently, a #P-functions counts the number of solutions to instances of an NP-problem. We say that a function f is #P-*complete* if it belongs to #P and every function $g \in$ #P is polynomial-time Turing reducible to f, i.e., g can be computed by a deterministic polynomial-time Turing machines which is allowed to make queries to f and answering these queries is done within one step (see, e.g., [77, 76]). The canonical example of a #P-complete problem is #3SAT, i.e., counting the number of satisfying assignments of a propositional formula given as a 3CNF. One of the most prominent #P-complete problem is counting the number of perfect matchings in a bipartite graph [152]. As in the case of NP, if there is a polynomial-time algorithm for computing some #P-complete function from #P then there are polynomial-time algorithms for all #P-functions—which is equally considered unlikely. In particular, such a polynomial-time algorithm would imply that P = NP.

Part I

Discrete, complex dynamical systems

3. Components

3.1 Background

A complex system, in a mathematical sense, can be viewed as a collection of highly interdependent variables. A discrete dynamical system is a complex system where variables update their values in discrete time. Though the interdependencies among the variables might have quite simple descriptions on a local level, the overall global behavior of the systems can be as complicated as unpredictable or undecidable (see, e.g., [28, 112, 113]). This phenomenon has been widely studied in the theory of cellular automata [158, 162, 163, 64, 164] and its applications (see, e.g., [64, 119, 60]).

Finite discrete dynamical systems are characterized by finite sets of variables which can take values from a finite domain. In essence (see, e.g., [19, 16]), a finite discrete dynamical system (over a finite domain) consists of

- a finite undirected graph, where vertices correspond to variables and edges correspond to an interdependence between the two connected variables,
- a set of local transition functions, one for each vertex, that map values of variables depending on the values of all connected variables to new variable values, and
- an update schedule that governs which variables are allowed to update their values in a certain time step.

Due to their structural simplicity and modelling flexibility, finite discrete dynamical systems are suitable for analyzing the behavior of real-world complex systems. In fact, the conception is motivated by large-scale simulations of traffic flow and their analysis (see, e.g, [8, 10]).

3.2 Terminology

In this section we describe a fairly general model of dynamical systems. We follow the approach given by [16] with a marginal generalization regarding update schedules.

3.2.1 Dynamical systems

The underlying network structure of a dynamical system is given by an undirected graph $G = (V, E)$. We suppose that the set V of vertices is ordered. So, without loss of generality, we assume $V = \{1, 2, \ldots, n\}$.

Definition 3.1. A *dynamical system S over a domain $\mathcal{D}$* is a pair (G, F) where

- $G = (V, E)$ is an undirected graph (the *network*) and
- $F = \{f_i \mid i \in V\}$ is a set of *local transition functions* $f_i : \mathcal{D}^{d_i+1} \to \mathcal{D}$.

The intuition of the definition is that each vertex i corresponds to an active element (entity, agent, actor etc.) which is always in some state x_i and which is capable to change its state, if necessary. The domain of S formalizes the set of possible states of all vertices of the network, i.e., for all $i \in V$, it always holds that $x_i \in \mathcal{D}$. A vector $\boldsymbol{x} = (x_i)_{i \in V}$ such that $x_i \in \mathcal{D}$ for all $i \in V$ is called a *configuration of* S. If it is more convenient, then we also say that a mapping $I : V \to \mathcal{D}$ is a configuration. A *subconfiguration* of I with respect to $A \subseteq V$ is a mapping $I[A] : A \to \mathcal{D}$ such that $I[A](i) = I(i)$ for all $i \in A$. The local transition function f_i for some vertex i describes how i changes its state depending on the states of its neighbors $N_G(i)$ in the network and its own state.

3.2.2 Discrete dynamical systems

We are particularly interested in dynamical system operating on a discrete time-scale. A *discrete dynamical system* $\mathcal{S} = (S, \alpha)$ consists of a dynamical system S and a mapping $\alpha : \{1, \ldots, T\} \to \mathcal{P}(V)$ where V is the set of vertices of the network of S and $T \in \mathbb{N}$. The mapping α is called the *update schedule* and specifies which states updates are realized at certain time-steps: for $t \in \{1, \ldots, T\}$, $\alpha(t)$ specifies those vertices that simultaneously update their states in step t. The *order* of an update schedule α is T. We denote this by $\mathrm{ord}(\alpha) = T$.

Depending on the structure of α, there two distinctive processing modes for dynamical systems:

- in a synchronous dynamical system, the image of α is $\{V\}$,
- in a sequential dynamical system, for each $i \in \{1, \ldots, \mathrm{ord}(\alpha)\}$, the value of $\alpha(i)$ is a singleton set.

The original approach in [19, 16] defines sequential dynamical systems to have update schedules $\alpha : \{1, \ldots, n\} \to \{1, \ldots, n\}$ of order n which are permutations. In [98] this has been generalized to the more flexible sequentiality definition above. A schedule of a sequential dynamical system is called a *sequential update schedule*. A schedule is *fair* if and only if for all $v \in V$ there is an i such that $v \in \alpha(i)$. Synchronous systems are always fair.

3.2.3 The global map

A discrete dynamical system $\mathcal{S} = (S, \alpha)$ over domain $\mathcal{D}$ induces a global map $\mathbf{F}_{\mathcal{S}} : \mathcal{D}^n \to \mathcal{D}^n$ where n is the number of vertices of S. For each vertex $i \in V$, define an *activity function* φ_i for a set $U \subseteq V$ and $\boldsymbol{x} = (x_1, \ldots, x_n) \in \mathcal{D}^n$ by

$$\varphi_i[U](\boldsymbol{x}) =_{\mathrm{def}} \begin{cases} f_i(x_{i_1}, \ldots, x_{i_{d_i+1}}) & \text{if } i \in U \\ x_i & \text{if } i \notin U \end{cases}$$

where $\{i_1, i_2, \ldots, i_{d_i+1}\} = N_G^0(i)$. For a set $U \subseteq V$, define the *global transition function* $\mathbf{F}_S[U] : \mathcal{D}^n \to \mathcal{D}^n$ for all $\boldsymbol{x} \in \mathcal{D}^n$ by

$$\mathbf{F}_S[U](\boldsymbol{x}) =_{\mathrm{def}} (\varphi_1[U](\boldsymbol{x}), \ldots, \varphi_n[U](\boldsymbol{x})).$$

Note that the global transition function does not refer to the update schedule, i.e., it only depends on the dynamical system S and not on $\mathcal{S}$. The function $F_{\mathcal{S}} : \mathcal{D}^n \to \mathcal{D}^n$ computed by the discrete dynamical system $\mathcal{S}$, the *global map* of $\mathcal{S}$, is defined by

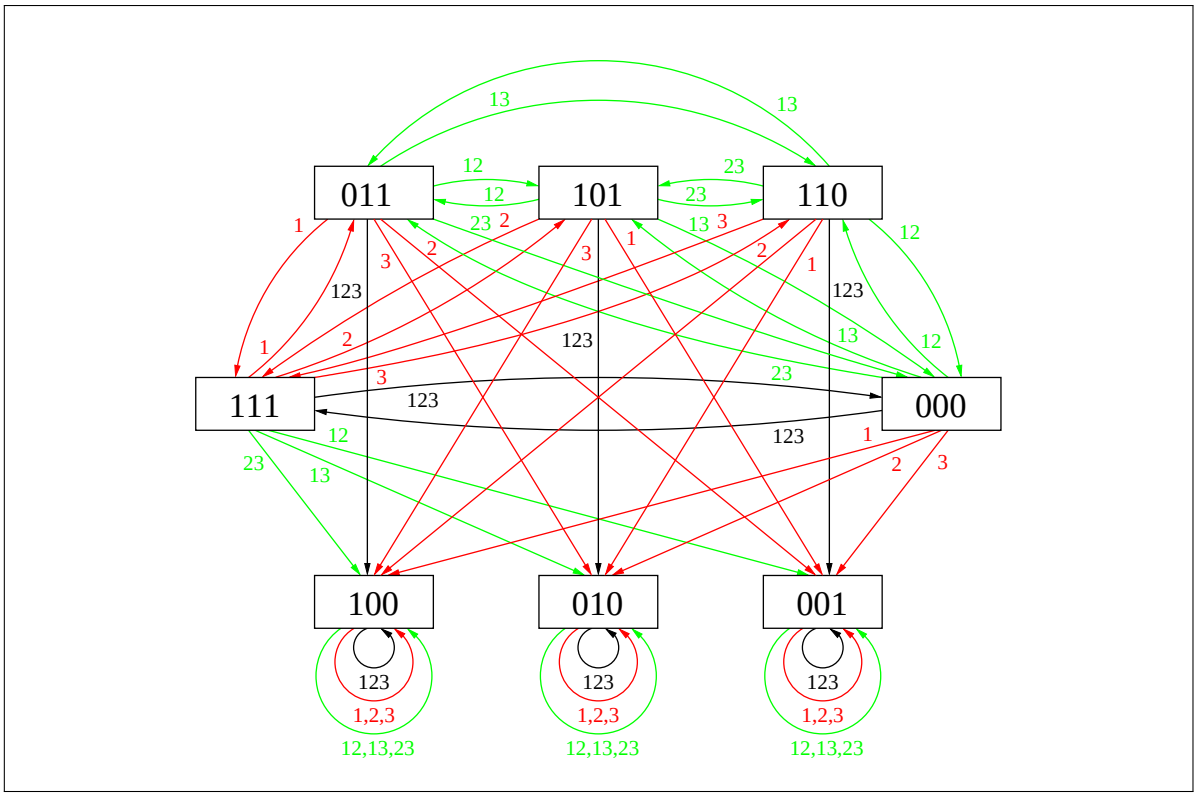


Figure 3.1. Global transition function

$$\mathbf{F}_{\mathcal{S}} =_{\text{def}} \prod_{k=1}^{T} \mathbf{F}_S[\alpha(k)],$$

i.e., by the composition of global transition functions specified by the update schedule. Recall that our usage of $f \cdot g$ is $(f \cdot g)(x) = g(f(x))$.

Example 3.2. Figure 3.1 visualizes the global transition functions of a dynamical system $S = (G, \{f_1, f_2, f_3\})$ where $G = (V, E)$ is a K^3 and for each $i \in \{1, 2, 3\}$,

$$f_i(x_1, x_2, x_3) = \begin{cases} x_i & \text{if } x_1 + x_2 + x_3 = 1, \\ 1 - x_i & \text{otherwise.} \end{cases}$$

For instance, if we assume that all elements update their states simultaneously then

$$\mathbf{F}_S[\{1, 2, 3\}](1, 1, 1) = (0, 0, 0) \quad \text{and} \quad \mathbf{F}_S[\{1, 2, 3\}](0, 0, 0) = (1, 1, 1).$$

Thus, in a synchronous dynamical systems these two configurations oscillate. In contrast, if the update schedule α is given by $\alpha(1) = 2$, $\alpha(2) = 1$, and $\alpha(3) = (3)$, the global map looks as follows:

$$\mathbf{F}_{(S,\alpha)}: \begin{array}{rcl} (0,0,0) & \mapsto & (0,1,0) \\ (0,0,1) & \mapsto & (0,0,1) \\ (0,1,0) & \mapsto & (0,1,0) \\ (0,1,1) & \mapsto & (0,0,1) \\ (1,0,0) & \mapsto & (1,0,0) \\ (1,0,1) & \mapsto & (0,1,0) \\ (1,1,0) & \mapsto & (1,0,0) \\ (1,1,1) & \mapsto & (0,0,1) \end{array}$$

Any further update does not change the resulting configurations. Thus, the discrete dynamical system is highly stable as it evolves from any configuration into a fixed point. Moreover, it is easily seen that the dynamical system has for each permutation this stabilizing property.

3.3 Networks

A characteristic of complex systems is the operational closedness of subsystems, i.e., their modularization. In an abstract sense we reflect this on the side of networks by considering all network classes closed under a fixed set of graph transformation rules. The set of transformation rules expresses certain modularization processes. For instance, stable complex-systems classes should be closed under taking subnetworks. Moreover, this approach has the advantage of providing great flexibility for system classifications and analyses when compared to the usual inductive approaches, such as Cayley graphs [64, 36] or graph grammars, which are more appropriate for modelling special complex systems.

Graph minors are certainly the best-understood non-trivial set of graph transformation rules. Consequently, we consider graph minors as a reference model for classifying the networks of complex systems. We are thus interested in graph classes which are closed under the following operations:

– vertex deletion
– edge deletion
– edge contraction

Recall that we implicitly require that a graph class is closed under isomorphisms. Modularization based on graph minors can be observed in the Internet routing hierarchy where the AS graph appears as a minor of the router graph.

We should note that some graph properties frequently observed in complex networks, such as bipartitness, power laws, or small-world properties, are not closed under the operations above. Though there are some graph evolution models explaining such behavior in concrete settings (see, e.g., [121, 27], there is not yet a theory available which characterizes network classes closed under the corresponding evolution rules. The sensitivity to the network samples restricts these approaches for usage in analysis issues.

In this section we define the relevant notions, give important alternative characterizations of graph classes closed under the above-mentioned operations, and consider the membership problem from a computational point of view. In the following we adopt notation from [43].

3.3.1 Closure properties

Let $G = (V, E)$ be an undirected graph and let $e = \{x, y\}$ be an edge in G. Then, the graph $G/e = (V', E')$ obtained by *contracting the edge e in G* is defined to consist of

$$\begin{aligned} V' &=_{\text{def}} \big(V \setminus \{x, y\}\big) \cup \{v_e\} && \text{(where } v_e \notin V \cup E\text{)} \\ E' &=_{\text{def}} \big\{\ \{v, w\} \mid \{v, w\} \in E \text{ and } \{v, w\} \cap \{x, y\} = \emptyset\ \big\} \\ &\qquad \cup\ \big\{\ \{v_e, w\} \mid \{x, w\} \in E \setminus \{e\} \text{ or } \{y, w\} \in E \setminus \{e\}\ \big\} \end{aligned}$$

Let X be another graph and let $\{\ V_x \mid x \in V(X)\ \}$ be a partition of the vertex set V of G into connected subgraphs such that for all $x, y \in V(X)$ there is an edge in G connecting some vertex from V_x and some vertex from V_y if and only if $\{x, y\} \in E(X)$. Then, we say that $G = MX$ (more precisely, $G \in MX$). The sets V_x are called *branch sets* (and will be contracted to obtain X). An easy inductive argument shows that $G = MX$ if and only if X is obtained from G by a sequence of edge contractions, i.e., there exist a sequence of graphs $G_0, \ldots, G_r$ and edges $e_i \in E(G_i)$ such that $G_0 = G$, $G_r \simeq X$, and for all $0 \leq i < r$, $G_{i+1} = G_i/e_i$. A graph X is said to be a *minor* of a graph Y (in symbols, $X \preceq Y$) if and only if there is a subgraph $G \subseteq Y$ such that $G = MX$.

A class $\mathcal{G}$ of graphs is said to be *closed under taking minors* if and only if for all graphs G and G', if $G \in \mathcal{G}$ and $G' \preceq G$, then $G' \in \mathcal{G}$. Obviously, for a graph class to be closed under taking minors is equivalent to being closed under vertex deletion, edge deletion, and edge contraction.

3.3.2 Forbidden minors

In this subsection we describe an alternative characterization of graph classes closed under taking minors.

Let $\mathcal{X}$ be any set of graphs. Let $\text{Forb}_{\preceq}(\mathcal{X})$ denote the class of all graphs without a minor in $\mathcal{X}$ (and which is closed under isomorphisms). More specifically, we define

$$\text{Forb}_{\preceq}(\mathcal{X}) =_{\text{def}} \{G \mid G \not\succeq X \text{ for all } X \in \mathcal{X}\ \}.$$

The set $\mathcal{X}$ is called the set of *forbidden minors*. Note that $\text{Forb}_{\preceq}(\emptyset)$ is the class of all graphs. As usual, we write $\text{Forb}_{\preceq}(X_1, \ldots, X_n)$ instead of $\text{Forb}_{\preceq}(\{X_1, \ldots, X_n\})$. A useful property of the forbidden-minor classes is the monotonicity with respect to $\preceq$.

Proposition 3.3. *Let X and Y be graphs. If $X \preceq Y$ then* $\text{Forb}_{\preceq}(X) \subseteq \text{Forb}_{\preceq}(Y)$.

Proof. Let G be a graph in $\text{Forb}_{\preceq}(X)$. Assume that $G \notin \text{Forb}_{\preceq}(Y)$, i.e., $Y \preceq G$. Since X is a minor of Y, it follows that $X \preceq G$. A contradiction. Hence, $G \in \text{Forb}_{\preceq}(Y)$. ❑

It is easily seen that a class $\mathcal{G}$ of graphs is closed under taking minors if and only if there is a set $\mathcal{X}$ such that $\mathcal{G} = \text{Forb}_{\preceq}(\mathcal{X})$. (Note that for "if", $\mathcal{X}$ can be chosen as the set of all complement graphs for graphs in $\mathcal{G}$.) In this characterization the set of forbidden minors is allowed to have an infinite cardinality. Much harder is it to prove that there is always a finite number of forbidden graphs. In fact, the corresponding graph minor theorem is one of deepest results in graph theory.

Theorem 3.4 (Robertson & Seymour [134]). *A class $\mathcal{G}$ of graphs is closed under taking minors if and only if there exists a finite set $\{X_1, \ldots, X_r\}$ of graphs such that $\mathcal{G} = \mathrm{Forb}_{\preceq}(X_1, \ldots, X_r)$.*

The most prominent examples of forbidden-minor classes are the following:

– planar graphs having $K_{3,3}$ and K^5 as forbidden minors [92, 160]
– series-parallel graphs having K^4 as a forbidden minor [47]
– outerplanar graphs having $K_{2,3}$ and K^4 as forbidden minors [33]

We use the characterizations by forbidden minors as definitions for the corresponding graph classes.

Bounded treewidth

As they characterize bounded treewidth, planar graphs are also important when using as forbidden minors. Let $G = (V, E)$ be a graph, let T be a tree, and let $\mathcal{V} = \{V_t\}_{t \in V(T)}$ be a family of vertex sets $V_t \subseteq V(G)$ indexed by the vertices in the tree T. The pair $(T, \mathcal{V})$ is said to be a *tree decomposition* of G if and only if the following conditions are satisfied:

1. $V = \bigcup_{t \in V(T)} V_t$,
2. for all $e \in E$ there is a $t \in V(T)$ such that $e \subseteq V_t$,
3. for all $t_1, t_2, t_3 \in V(T)$ such that t_2 lies on a path from t_1 to t_3 in T, it holds that $V_{t_1} \cap V_{t_3} \subseteq V_{t_2}$.

The *width* of a tree decomposition $(T, \mathcal{V})$ is defined to be

$$\max\{\|V_t\| - 1 \mid t \in V(T)\}.$$

The *treewidth* $\mathrm{tw}(G)$ of graph G is defined to be the minimum width of a tree decomposition of G. A class $\mathcal{G}$ of graphs is said to have *bounded treewidth* if and only if there is a $k \in \mathbb{N}$ such that all graphs in the class have treewidth at most k.

Theorem 3.5 (Robertson & Seymour [132]). *Let X be a graph. Then, X is planar if and only if $\mathrm{Forb}_{\preceq}(X)$ has bounded treewidth.*

Note that computing the treewidth of a given graph is NP-hard [5]. In contrast, for a fixed treewidth computing a tree decomposition can be done in linear time [21].

Bounded degree

A similar but much less subtle behavior to planar graphs show graphs with a vertex cover of size one. Let $G = (V, E)$ be a graph. We say that a subset $U \subseteq V$ is a *vertex cover* of G if and only if for all edges $\{u, v\} \in E$, it holds that $\{u, v\} \cap U \neq \emptyset$. It is known that the class of graphs having a vertex cover of size at most k is closed under taking minors [30]. Moreover, G has a vertex cover of size one if and only if G belongs to $\mathrm{Forb}_{\preceq}(K^3, K^2 \oplus K^2)$ [30]. A class of graphs is said to have *bounded degree* if and only if there is a $k \in \mathbb{N}$ such that all graphs in the class have a maximum vertex-degree of at most k.

Proposition 3.6. *Let X be a graph. Then, X has a vertex cover of size one if and only if* $\mathrm{Forb}_{\preceq}(X)$ *has bounded degree.*

Proof. For ($\Rightarrow$), suppose that X has a vertex cover of size one. Then, X consists of some star graph $K_{1,k}$ and some isolated vertices $u_1, \ldots, u_r$. Assume that $\mathrm{Forb}_{\preceq}(X)$ does not have bounded degree. Thus, there exists a graph $G \in \mathrm{Forb}_{\preceq}(X)$ of maximum vertex-degree at least $k + r$. So, G contains a subgraph $K_{1,k+r}$. Hence, $X \preceq G$, contradicting $G \in \mathrm{Forb}_{\preceq}(X)$. Therefore, $\mathrm{Forb}_{\preceq}(X)$ has bounded degree.

For ($\Leftarrow$), suppose that X does not have any vertex cover of size one. It is easily seen that in this case, X contains a triangle or two non-incident edges. First, suppose X contains a triangle K^3. As $\mathrm{Forb}_{\preceq}(K^3)$ contains the class of all trees, which certainly does not have bounded degree, we easily obtain from $K^3 \preceq X$, that $\mathrm{Forb}_{\preceq}(X)$ does not have bounded degree as well. Second, suppose X contains at least two non-incident edges, i.e., $K^2 \oplus K^2 \preceq X$. It follows that $\mathrm{Forb}_{\preceq}(X)$ contains for all $k \in \mathbb{N}$, the star graph $K_{1,k}$. Thus, $\mathrm{Forb}_{\preceq}(X)$ does not have bounded degree. This completes the proof of the direction from left to right. ❑

3.3.3 The Membership problem

From a computational point of view we are interested in the complexity of deciding whether a given graph is contained in some relevant graph class. For a fixed class $\mathcal{G}$ of graphs we consider the following problem:

Problem:	Membership($\mathcal{G}$)
Input:	A graph $G = (V, E)$
Question:	Does G belong to $\mathcal{G}$?

A consequence of Theorem 3.4 is that for deciding the membership of a graph in a graph class closed under taking minors it is enough to check whether none of the finite and fixed number of forbidden minors of the class occurs as minor of the graph in question. There is an $O(n^3)$ algorithm for this minor-containment test [133]. This gives the following theorem.

Theorem 3.7 (Robertson & Seymour [133]). *Let $\mathcal{G}$ be a graph class closed under taking minors. Then,* Membership($\mathcal{G}$) *is solvable in time $O(n^3)$.*

The efficiency of the minor-containment test relies on an $O(n^3)$ algorithm for the Disjoint Paths problem: given a graph $G = (V, E)$ and pairs $(s_1, t_1), \ldots, (s_k, t_k)$ of vertices $s_i, t_i \in V$, are there vertex-disjoint paths $p_1, \ldots, p_k$ in G such that p_i joins s_i and t_i? In general, the problem is known to be NP-complete [85]. This implies that the variable Membership problem, i.e., when the forbidden minors are part of the input, is NP-complete as well. In contrast, if the number of k is fixed as it is the case for the Membership problem, then the result in [133] shows that the Disjoint Path problem is fixed-parameter tractable. Note that the problem for directed graphs is NP-complete even for two pairs [57].

The algorithm used for Theorem 3.7 is practically infeasible, "since it involves the manipulation of enormous constants" [133, p. 66]. This is due to the fact that the set of forbidden minors can be tremendously large. As an example, the number of forbidden minors to graphs

of pathwidth three is at least 60,000,000 [87, 46]. Moreover, we do not know any general method to determine sets of forbidden minors of graph classes.

For some special classes there exist faster (and easier-to-implement) algorithms. For instance, if X is planar graph then MEMBERSHIP(Forb$_{\preceq}(X)$) can be solved in time $O(n^2)$ [133]. There are also classical linear-time algorithms for MEMBERSHIP$(K_{3,3}, K^5)$ [99, 78].

3.4 Local transition functions

As with networks we consider classes closed under certain operations reflecting a certain modularity of complex systems. Operations typically considered in the context of discrete dynamical systems are projections, permutations, addition, and various threshold-operations (see, e.g., [16, 17, 114, 15, 18, 12, 13, 11, 159, 9, 14]). The operation *par excellence* [106] on functions is the superposition (composition) generalizing most of the interesting operations on finite functions. We adopt notation from [24].

3.4.1 Closure properties

In the following we introduce our notion of being superpositionally closed.

Let $\mathcal{D}$ be any finite domain. An n-ary $\mathcal{D}$-function f is a mapping $f : \mathcal{D}^n \to \mathcal{D}$. Let $\mathcal{F}$ be any class of $\mathcal{D}$-functions. We define the following closure properties for $\mathcal{F}$:

- $\mathcal{F}$ is closed under *introduction of fictive variables* if and only if for all $n \in \mathbb{N}$ and for all n-ary functions $f \in \mathcal{F}$, the function $\varphi : \mathcal{D}^{n+1} \to \mathcal{D}$ defined by $\varphi(x_1, \ldots, x_{n+1}) =_{\text{def}} f(x_1, \ldots, x_n)$ belongs to $\mathcal{F}$.
- $\mathcal{F}$ is closed under *permutations of variables* if and only if for all $n \in \mathbb{N}$, for all n-ary functions $f \in \mathcal{F}$, and for all permutations $\pi : \{1, \ldots, n\} \to \{1, \ldots, n\}$, the function $\varphi : \mathcal{D}^n \to \mathcal{D}$ defined by $\varphi(x_1, \ldots, x_n) =_{\text{def}} f(x_{\pi^{-1}(1)}, \ldots, x_{\pi^{-1}(n)})$ belongs to $\mathcal{F}$.
- $\mathcal{F}$ is closed under *identification of variables* if and only if for all $n \in \mathbb{N}$, for all n-ary functions $f \in \mathcal{F}$, the function $\varphi : \mathcal{D}^{n-1} \to \mathcal{D}$ defined by $\varphi(x_1, \ldots, x_{n-1}) =_{\text{def}} f(x_1, \ldots, x_{n-1}, x_{n-1})$ belongs to $\mathcal{F}$.
- $\mathcal{F}$ is closed under *substitution* if and only if for all $n, m \in \mathbb{N}$, for all n-ary functions $f \in \mathcal{F}$, and for all m-ary functions $g \in \mathcal{F}$, the function $\varphi : \mathcal{D}^{n+m-1} \to \mathcal{D}$ defined by $\varphi(x_1, \ldots, x_{n-1}, x_n, \ldots, x_{n+m-1}) =_{\text{def}} f(x_1, \ldots, x_{n-1}, g(x_n, \ldots, x_{n+m-1}))$ belongs to $\mathcal{F}$.

For a class $\mathcal{F}$ of $\mathcal{D}$-functions, the class $[\mathcal{F}]$ is defined to be the minimal class containing $\mathcal{F} \cup \{\text{id}\}$ and which is closed under introduction of fictive variables, permutations of variables, identification of variables, and substitution. Note that substitution as defined here only allows to insert another function into the last position. Clearly, a more general definition of substitution allowing insertions at each position does not change the class $[\mathcal{F}]$ since permutation of variables is allowed. Actually, there are many more equivalent definitions (see, e.g., [157]). In general, a class $[\mathcal{F}]$ for some class $\mathcal{F}$ of $\mathcal{D}$-functions is called a *clone*.

3.4.2 Boolean clones

In the case of the boolean domain $\mathcal{D} = \{0, 1\}$ we are in the fortunate situation to have a complete description of the family of all clones [123]. In this subsection we present all clones of boolean functions.

Let BF denote the class of all boolean functions. There are two 0-ary boolean functions:

$$c_0 =_{\text{def}} 0 \quad \text{(denoted in formulas by 0)}$$
$$c_1 =_{\text{def}} 1 \quad \text{(denoted in formulas by 1)}$$

There are two 1-ary boolean functions (without fictive variables):

$$\text{id}(x) = 1 \iff_{\text{def}} x = 1 \quad \text{(denoted in formulas by } x\text{)}$$
$$\text{not}(x) = 1 \iff_{\text{def}} x \neq 1 \quad \text{(denoted in formulas by } \overline{x} \text{ or } \neg\text{)}$$

There are nine 2-ary boolean functions (without fictive variables). The most prominent are:

$$\begin{aligned}
\text{and}(x,y) = 1 &\iff_{\text{def}} \min(x,y) = 1 && \text{(denoted in formulas by } \wedge\text{)}\\
\text{or}(x,y) = 1 &\iff_{\text{def}} \max(x,y) = 1 && \text{(denoted in formulas by } \vee\text{)}\\
\text{imp}(x,y) = 1 &\iff_{\text{def}} x \leq y && \text{(denoted in formulas by } \rightarrow\text{)}\\
\text{eq}(x,y) = 1 &\iff_{\text{def}} x = y && \text{(denoted in formulas by } \leftrightarrow\text{)}\\
\text{xor}(x,y) = 1 &\iff_{\text{def}} x \neq y && \text{(denoted in formulas by } \oplus\text{)}
\end{aligned}$$

The following properties of boolean functions are important to describe clones.

- For $b \in \{0,1\}$, a boolean function f is said to be *b-reproducing* if and only if $f(b,\ldots,b) = b$.
- For binary n-tuples $\boldsymbol{a} = (a_1,\ldots,a_n)$ and $\boldsymbol{b} = (b_1,\ldots,b_n)$, we say that $(a_1,\ldots,a_n) \leq (b_1,\ldots,b_n)$ if and only if for all $i \in \{1,\ldots,n\}$, it holds that $a_i \leq b_i$. An n-ary boolean function f is said to be *monotone* if and only if for all $\boldsymbol{x},\boldsymbol{y} \in \{0,1\}^n$, $\boldsymbol{x} \leq \boldsymbol{y}$ implies $f(\boldsymbol{x}) \leq f(\boldsymbol{y})$.
- An n-ary boolean function f is said to be *selfdual* if and only if for all $(x_1,\ldots,x_n) \in \{0,1\}^n$, it holds that $f(x_1,\ldots,x_n) = \text{not}(f(\text{not}(x_1),\ldots,\text{not}(x_n)))$.
- A boolean function f is *linear* if and only if there exists constants $a_1,\ldots,a_n \in \{0,1\}$ such that $f(x_1,\ldots,x_n) = a_0 \oplus a_1x_1 \oplus \cdots \oplus a_nx_n$. Note that $\oplus$ is understood as addition modulo 2 and xy is understood as multiplication modulo 2.
- For $b \in \{0,1\}$, a tuple set $T \subseteq \{0,1\}^n$ is said to be b-separating if and only if there is an $i \in \{1,\ldots,n\}$ such that for $(t_1,\ldots,t_n) \in T$ holds $t_i = b$. A boolean function f is *b-separating* if and only if $f^{-1}(b)$ is b-separating. A function f is called *b-separating of level* k if and only if every $T \subseteq f^{-1}(b)$ such that $\|T\| = k$ is b-separating.
- An n-ary boolean function f is *minimizing* if and only if for all $x_1,\ldots,x_n \in \{0,1\}$, it holds that $f(x_1,\ldots,x_n) = \min(x_1,\ldots,x_n)$. Dually, an n-ary boolean function f is *maximizing* if and only if for all $x_1,\ldots,x_n \in \{0,1\}$, it holds that $f(x_1,\ldots,x_n) = \max(x_1,\ldots,x_n)$. A boolean function f is *constant* if and only if there exists a $b \in \{0,1\}$ such that for all $x_1,\ldots,x_n \in \{0,1\}$, it holds that $f(x_1,\ldots,x_n) = b$.

Table 3.1 contains a list of all boolean clones. Each class has a finite logical basis, i.e., a set of functions from which all functions in the class can be built using the introduction of fictive variables, permutations of variables, identification of variables, and superposition. Note that the identity functions is always added to the basis. We thus omit them in the basis.

It is a famous result by Post [123] that the family of all boolean clones classes is a countable lattice with respect to set inclusion. Countability is seen from Table 3.1. We give a short reproof of the lattice property.

Class	Definition	Logical Basis
BF	all boolean functions	$\{\wedge, \neg\}$
R_0	$\{f \mid f \text{ is 0-reproducing}\}$	$\{\wedge, \oplus\}$
R_1	$\{f \mid f \text{ is 1-reproducing}\}$	$\{\vee, x \oplus y \oplus 1\}$
R_2	$\mathrm{R}_0 \cap \mathrm{R}_1$	$\{\vee, x \wedge (y \oplus z \oplus 1)\}$
M	$\{f \mid f \text{ is monotone}\}$	$\{\wedge, \vee, 0, 1\}$
M_0	$\mathrm{M} \cap \mathrm{R}_0$	$\{\wedge, \vee, 0\}$
M_1	$\mathrm{M} \cap \mathrm{R}_1$	$\{\wedge, \vee, 1\}$
M_2	$\mathrm{M} \cap \mathrm{R}_2$	$\{\wedge, \vee\}$
S_0^k	$\{f \mid f \text{ is 0-separating of degree } k\}$	$\{\rightarrow, \bigwedge_{i=1}^{k+1}(x_1 \vee \cdots \vee x_{i-1} \vee x_{i+1} \vee \cdots \vee x_{k+1})\}$
S_0	$\{f \mid f \text{ is 0-separating}\}$	$\{\rightarrow\}$
S_1^k	$\{f \mid f \text{ is 1-separating of degree } k\}$	$\{x \wedge \overline{y}, \bigvee_{i=1}^{k+1} x_1 \wedge \cdots \wedge x_{i-1} \wedge x_{i+1} \wedge \cdots \wedge x_{k+1}\}$
S_1	$\{f \mid f \text{ is 1-separating}\}$	$\{x \wedge \overline{y}\}$
S_{02}^k	$S_0^k \cap \mathrm{R}_2$	$\{x \vee (y \wedge \overline{z}), \bigwedge_{i=1}^{k+1}(x_1 \vee \cdots \vee x_{i-1} \vee x_{i+1} \vee \cdots \vee x_{k+1})\}$
S_{02}	$S_0 \cap \mathrm{R}_2$	$\{x \vee (y \wedge \overline{z})\}$
S_{01}^k	$S_0^k \cap \mathrm{M}$	$\{\bigwedge_{i=1}^{k+1}(x_1 \vee \cdots \vee x_{i-1} \vee x_{i+1} \vee \cdots \vee x_{k+1}), 1\}$
S_{01}	$S_0 \cap \mathrm{M}$	$\{x \vee (y \wedge z), 1\}$
S_{00}^k	$S_0^k \cap \mathrm{R}_2 \cap \mathrm{M}$	$\{x \vee (y \wedge z), \bigwedge_{i=1}^{k+1}(x_1 \vee \cdots \vee x_{i-1} \vee x_{i+1} \vee \cdots \vee x_{k+1})\}$
S_{00}	$S_0 \cap \mathrm{R}_2 \cap \mathrm{M}$	$\{x \vee (y \wedge z)\}$
S_{12}^k	$S_1^k \cap \mathrm{R}_2$	$\{x \wedge (y \vee \overline{z}), \bigvee_{i=1}^{k+1} x_1 \wedge \cdots \wedge x_{i-1} \wedge x_{i+1} \wedge \cdots \wedge x_{k+1})\}$
S_{12}	$S_1 \cap \mathrm{R}_2$	$\{x \wedge (y \vee \overline{z})\}$
S_{11}^k	$S_1^k \cap \mathrm{M}$	$\{\bigvee_{i=1}^{k+1} x_1 \wedge \cdots \wedge x_{i-1} \wedge x_{i+1} \wedge \cdots \wedge x_{k+1}), 0\}$
S_{11}	$S_1 \cap \mathrm{M}$	$\{x \wedge (y \vee z), 0\}$
S_{10}^k	$S_1^k \cap \mathrm{R}_2 \cap \mathrm{M}$	$\{x \wedge (y \vee z), \bigvee_{i=1}^{k+1} x_1 \wedge \cdots \wedge x_{i-1} \wedge x_{i+1} \wedge \cdots \wedge x_{k+1}), 0\}$
S_{10}	$S_1 \cap \mathrm{R}_2 \cap \mathrm{M}$	$\{x \wedge (y \vee z)\}$
D	$\{f \mid f \text{ is selfdual}\}$	$\{(x \wedge \overline{y}) \vee (x \wedge \overline{z}) \vee (\overline{y} \wedge \overline{z})\}$
D_1	$\mathrm{D} \cap \mathrm{R}_2$	$\{(x \wedge y) \vee (x \wedge \overline{z}) \vee (y \wedge \overline{z})\}$
D_2	$\mathrm{D} \cap \mathrm{M}$	$\{(x \wedge y) \vee (x \wedge z) \vee (y \wedge z)\}$
L	$\{f \mid f \text{ is linear}\}$	$\{\oplus, 1\}$
L_0	$\mathrm{L} \cap \mathrm{R}_0$	$\{\oplus\}$
L_1	$\mathrm{L} \cap \mathrm{R}_1$	$\{\leftrightarrow\}$
L_2	$\mathrm{L} \cap \mathrm{R}_2$	$\{x \oplus y \oplus z\}$
L_3	$\mathrm{L} \cap \mathrm{D}$	$\{x \oplus y \oplus z \oplus 1\}$
V	$\{f \mid f \text{ is maximizing or constant}\}$	$\{\vee, 0, 1\}$
V_0	$[\{\text{or}\}] \cup [\{c_0\}]$	$\{\vee, 0\}$
V_1	$[\{\text{or}\}] \cup [\{c_1\}]$	$\{\vee, 1\}$
V_2	$[\{\text{or}\}]$	$\{\vee\}$
E	$\{f \mid f \text{ is minimizing or constant}\}$	$\{\wedge, 0, 1\}$
E_0	$[\{\text{and}\}] \cup [\{c_0\}]$	$\{\wedge, 0\}$
E_1	$[\{\text{and}\}] \cup [\{c_1\}]$	$\{\wedge, 1\}$
E_2	$[\{\text{and}\}]$	$\{\wedge\}$
N	$[\{\text{not}\}] \cup [\{c_0\}] \cup [\{c_1\}]$	$\{\neg, 0\}$
N_2	$[\{\text{not}\}]$	$\{\neg\}$
I	$[\{c_0\}] \cup [\{c_1\}]$	$\{0, 1\}$
I_0	$[\{c_0\}]$	$\{0\}$
I_1	$[\{c_1\}]$	$\{1\}$
I_2	$[\emptyset]$	$\emptyset$

Table 3.1. Boolean clones.

Proposition 3.8 (e.g., [122]). *The family of all boolean clones is a lattice with respect to set inclusion.*

Proof. It is clear that all boolean clones are contained in BF and contain I_2. So there is a maximum and a minimum element in the family of all clones. It remains to prove that for two clones $\mathcal{F}$ and $\mathcal{F}'$

1. there is exactly one clone contained in both $\mathcal{F}$ and $\mathcal{F}'$.
2. there is exactly one clone containing both $\mathcal{F}$ and $\mathcal{F}'$,

It is easily seen that the intersection of two clones is again a clone. Thus, $\mathcal{F} \cap \mathcal{F}'$ is a unique, maximal clone contained in $\mathcal{F}$ and $\mathcal{F}'$. This implies the first statement. To prove the second statement, let $\mathcal{H}$ be any clone such that $\mathcal{F} \cup \mathcal{F}' \subseteq \mathcal{H}$. Hence, $[\mathcal{F} \cup \mathcal{F}'] \subseteq \mathcal{H}$. Thus, $[\mathcal{F} \cup \mathcal{F}']$ is a unique, minimal clone containing $\mathcal{F}$ and $\mathcal{F}'$. This implies the second statement. ❑

Note that Proposition 3.8 can be readily extended to arbitrary domains.

The inclusion graph of Post's lattice is shown in Figure 3.2. Important classes within Post's lattice are the maximal classes, i.e., maximal clones different to BF. These classes are R_0, R_1, M, D, and L. Further relevant classes are the irreducible classes. A class is meet-irreducible in Post's lattice if the class cannot be expressed as the intersection of at least two other clones. In the inclusion graph these are the classes which only have one upper neighbor. Clearly, all maximal clones are meet-irreducible. Dually, the join-irreducible classes are those that cannot be expressed as the union of at least two other classes. In the inclusion graph these are the classes which only have one lower neighbor. Each class in Post's lattice is the intersection of a finite set of meet-irreducible classes or the union of a finite set of join-irreducible classes.

3.4.3 Polymorphisms

There is an alternative approach to define clones which is more appropriate for a description of clones over larger domains.

Let $\mathcal{D}$ be any domain. Let R be an n-ary relation over $\mathcal{D}$. An m-ary $\mathcal{D}$-function f is said to be a *polymorphism* of R if for all tuples $x_1, \ldots, x_m \in R$ where $x_i = (x_i[1], \ldots, x_i[n])$ it holds that

$$\big(f(x_1[1], \ldots, x_m[1]), f(x_1[2], \ldots, x_m[2]), \ldots, f(x_1[n], \ldots, x_m[n])\big) \in R.$$

Let $\mathrm{Pol}(R)$ denote the set of all polymorphisms of R. We extend this notion to relation sets. For any set $\mathcal{Q}$ of relations define $\mathrm{Pol}(\mathcal{Q}) =_{\mathrm{def}} \bigcap_{R \in \mathcal{Q}} \mathrm{Pol}(R)$. It is easily seen from the definition that for each relation set $\mathcal{Q}$ the function class $\mathrm{Pol}(\mathcal{Q})$ is a clone. The following theorem states that clones are characterized by sets of polymorphisms.

Theorem 3.9 (Geiger [66]; Bodnarchuk et al. [22, 23]). *Let $\mathcal{D}$ be any finite domain and let $\mathcal{F}$ be a class of $\mathcal{D}$-functions. Then, $\mathcal{F}$ is a clone if and only if there exists a set $\mathcal{Q}$ of relations over $\mathcal{D}$ such that $\mathcal{F} = \mathrm{Pol}(\mathcal{Q})$.*

Example 3.10. Let R_0 denote the relation $\{(0,0),(1,1),(0,1)\}$ on the boolean domain $\mathcal{D} = \{0,1\}$, i.e., the standard total order on $\{0,1\}$. Note that for $x \in \{0,1\} \times \{0,1\}$ it holds $x \in R_0$ if and only if $x[1] \leq x[2]$. Let f be any polymorphism of R_0. Then, for all $x_1, \ldots, x_n \in R_0$ it holds $(x_1[1], \ldots, x_n[1]) \leq (x_1[2], \ldots, x_n[2])$ with respect to the vector-ordering and

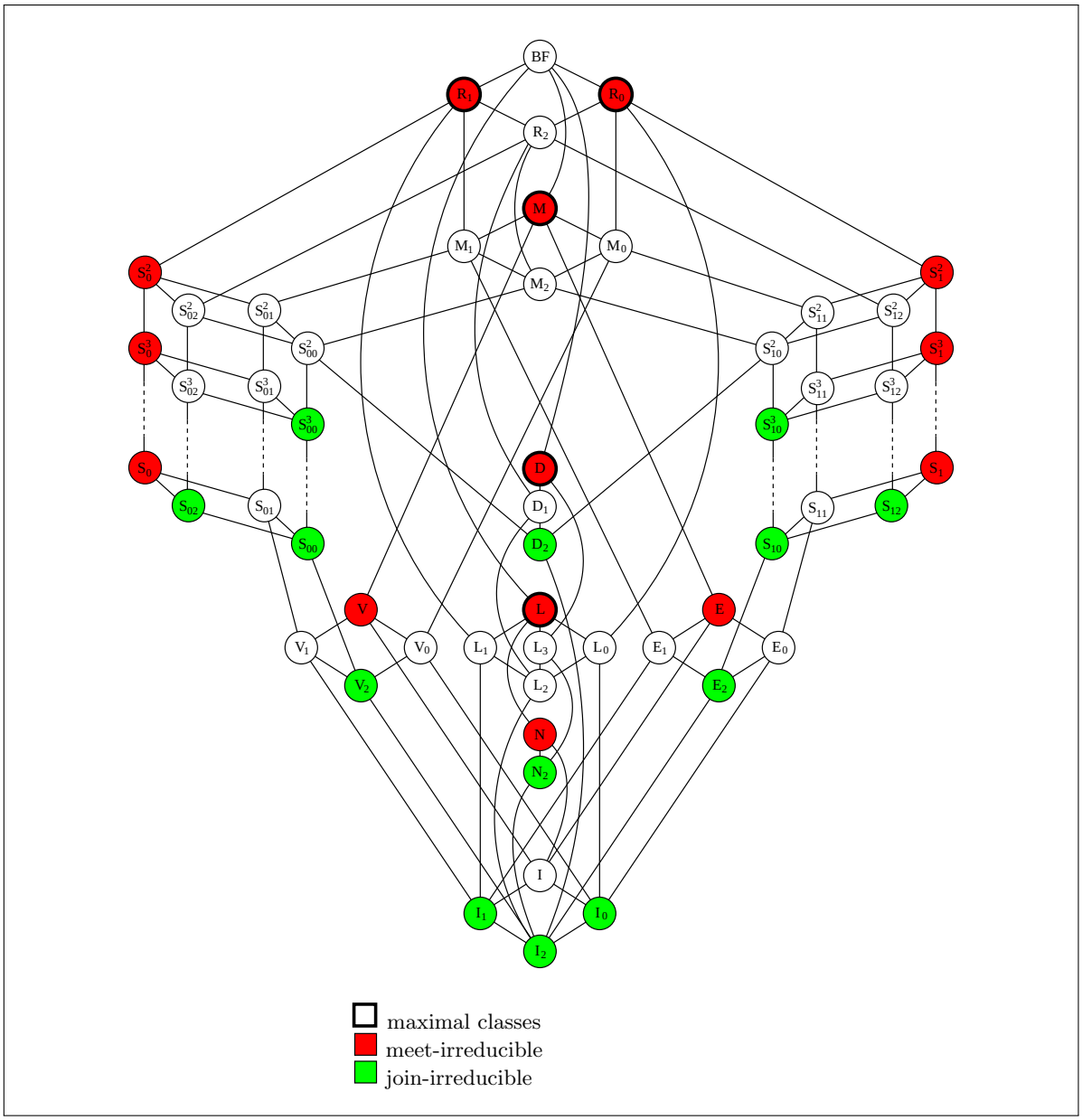


Figure 3.2. Post's lattice

$f(x_1[1],\ldots,x_n[1]) \leq f(x_1[2],\ldots,x_n[2])$. Thus, f is monotone. Hence, $\mathrm{Pol}(R_0) = \mathrm{M}$. Note that for the relation $R_1 = \{(0,0),(1,1),(1,0)\}$, which is the second order on $\{0,1\}$, we have $\mathrm{Pol}(R_1) = \mathrm{M}$ as well.

As a side remark we mention that there is a dual theorem stating that each class of relations is closed under a certain set of operations if and only if it is a class of invariants of some clone (see, e.g., [122, Theorem 1.2.3]). A relation R is an invariant of a function f if and only if f is polymorphism of R. The polymorphisms and the invariants form a Galois correspondence between clones and the so-called co-clones (see, e.g., [122, 25]).

Algorithm 1: A generic algorithm for polymorphism testing given an m-ary relation R

Input: A function $f : \mathcal{D}^n \to \mathcal{D}$ given as a lookup table, i.e., a list of $\|\mathcal{D}\|^n$ many $(n+1)$-tuples
Output: An indication whether f is a polymorphism of R

```
label := true
foreach x_1, ..., x_n ∈ R do
    for i := 1 to m do
        y_i := f(x_1[i], ..., x_n[i])
    end
    if (y_1, ..., y_m) ∉ R then
        label := false
    end
end
if label then
    return "f ∈ Pol(R)"
else
    return "f ∉ Pol(R)"
end
```

3.4.4 The Membership problem

Given any $\mathcal{D}$-function we want to decide whether the function belongs to a certain clone. We are thus interested in the complexity of the following problem:

Problem:	MEMBERSHIP($\mathcal{F}$)
Input:	An n-ary function $f : \mathcal{D}^n \to \mathcal{D}$
Question:	Does f belong to $\mathcal{F}$?

The complexity of the problem depends on the representation of the function. The standard representation in our setting is the lookup table. In this case it is easily seen that the membership test can be done in polynomial time for any clone given by some relation.

Proposition 3.11. *Let R be any m-ary relation over a domain $\mathcal{D}$ such that $\|\mathcal{D}\| \geq 2$. Then,* MEMBERSHIP(Pol(R)) *is solvable in polynomial time.*

Proof. Consider the generic Algorithm 1 which tests for a given function f in lookup-table representation whether f is a polymorphism of a fixed m-ary relation R, i.e., whether f belongs to Pol(R) or not. Since the algorithm simply examines the definition of a polymorphism, the correctness is obvious. Regarding the run-time analysis we obtain for the number of accesses to f and R:

$$O\big(\|R\|^m(nm+m)\big) = O\left(\left(\|\mathcal{D}\|^{\log_{\|\mathcal{D}\|}\|R\|}\right)^n\right) = O\left((\|\mathcal{D}\|^n)^{\log_{\|\mathcal{D}\|}\|R\|}\right) = O\left(|f|^{\log_{\|\mathcal{D}\|}\|R\|}\right)$$

The latter term is polynomial in $|f|$ since $\|\mathcal{D}\|$ and $\|R\|$ are fixed. This proves the proposition. ❑

Corollary 3.12. *For each fixed $m \geq 1$ the variable Membership problem for clones where the m-ary relation is also part of the input is solvable in polynomial time.*

Proof. We use the Algorithm 1 and the estimation of the proof of Proposition 3.11. Since $\|R\| \leq \|\mathcal{D}\|^m$ we proceed with the estimation and obtain as an upper bound for the accesses to f and R:

$$O\left(|f|^{\log_{\|\mathcal{D}\|}\|\mathcal{D}\|^m}\right) = O(|f|^m) = O\left((|f| + |R|)^m\right).$$

Since m is constant this is a bound polynomial in the input size. ❑

Let R be an m-ary relation over $\mathcal{D}$. Then, we say that R has *order* m. We define $\mathrm{ord}(R)$ to be the order of R. Let $\mathcal{Q}$ be a set of relations. The order of $\mathcal{Q}$ is defined by

$$\mathrm{ord}(\mathcal{Q}) =_{\mathrm{def}} \begin{cases} \max\{\mathrm{ord}(R) \mid R \in \mathcal{Q}\} & \text{if the maximum exists} \\ \infty & \text{otherwise} \end{cases}$$

We set $\mathrm{ord}(\emptyset) =_{\mathrm{def}} 0$. Let $\mathcal{F}$ be a clone of $\mathcal{D}$-functions. The *relation degree* $\deg(\mathcal{F})$ of $\mathcal{F}$ is the minimum order of a set of invariants of $\mathcal{F}$, i.e.,

$$\deg(\mathcal{F}) =_{\mathrm{def}} \min\{\mathrm{ord}(\mathcal{Q}) \mid \mathcal{F} = \mathrm{Pol}(\mathcal{Q})\}.$$

It is clear that if $\deg(\mathcal{F})$ is finite then there are finitely many $R_1, \ldots, R_k$ such that $\mathcal{F} = \mathrm{Pol}(R_1, \ldots, R_k)$.

Corollary 3.13. *Let $\mathcal{D}$ be a finite domain and let $\mathcal{F}$ be a clone of $\mathcal{D}$-functions. If $\deg(\mathcal{F}) < \infty$ then* Membership$(\mathcal{F})$ *is solvable in polynomial time.*

Corollary 3.14. *Let $\mathcal{F}$ be any boolean clone. Then,* Membership$(\mathcal{F})$ *is solvable in polynomial time.*

Proof. All boolean clones except $\mathrm{S}_0, \mathrm{S}_1, \mathrm{S}_{00}, \mathrm{S}_{01}, \mathrm{S}_{02}, \mathrm{S}_{10}, \mathrm{S}_{11}, \mathrm{S}_{12}$ have finite relation degrees (see, e.g., [122, Example 4.1.14f]). Corollary 3.13 implies the polynomial-time solvability of the Membership problem for these clones. For the remaining clones it is enough to show that Membership$_{\mathrm{T}}(\mathrm{S}_0)$ and Membership$_{\mathrm{T}}(\mathrm{S}_1)$ can be solved in polynomial time. Note that the other clones are intersections of S_0 or S_1 with clones having finite relation degrees. However, testing a function for being b-separating for $b \in \{0, 1\}$ is certainly polynomial time when the function is given by a lookup table. ❑

Note that a result similar to Corollary 3.14 for larger domains is in principle out of reach: already for a ternary domain the number of clones is continuum (see, e.g., [122, Theorem 3.1.4]); but there are only countably many polynomial-time algorithms for membership tests. However, the scope of Proposition 3.11 covers a significant part of clones for any domain. For instance, all maximal clones are given by single relations (see, e.g., [122]).

Independently of the simple analysis presented here, in [157] the Membership problem has been investigated in more detail. The results are that for any fixed relation R, Membership$(\mathrm{Pol}(R))$ is in AC^0 whereas the uniform Membership problem, i.e., the problem to decide, given a function f and a set of functions $g_1, \ldots, g_k$, does f belong to $[\{g_1, \ldots, g_k\}]$, is in qAC^0 for boolean functions. Note that the uniform Membership problem over arbitrary domains is complete for the complexity class EXPTIME [20].

In case of boolean clones the membership test becomes computationally hard in many cases when the function is given by circuits.

Theorem 3.15 (Böhler & Schnoor [26]). *Let $\mathcal{F}$ be an arbitrary boolean clone. Suppose functions are represented by boolean circuits. If $\mathcal{F} \supseteq \mathrm{R}_0$ then* Membership($\mathcal{F}$) *is solvable in polynomial time. Otherwise,* Membership($\mathcal{F}$) *is* coNP*-complete.*

In fact, in [26] many more problems have been investigated which have the following form: suppose we are given a boolean circuit using gates from the logical basis of a fixed clone $\mathcal{B}$, decide whether the circuit actually computes a function from another fixed clone $\mathcal{A}$. As it has turned out all these problem are either decidable in polynomial time or coNP-complete (or have a still unknown complexity).

3.5 Update schedules

Update schedules are typically exogenous to a dynamical system under study. Except synchronicity which is very much an idealization there is no mathematical theory explaining the meaning of modularity for update schedules. Initial attempts have been made in the study of hierarchically organized dynamical systems where the update process is governed by a superior dynamical system (see, e.g., [7]).

From the viewpoint of analyzability and simulatability of complex systems a promising approach is given in [98]. The approach is motivated by black-box simulations of sequential systems. A black-box simulation of a discrete dynamical system is any discrete dynamical system which provides a certain behavioral equivalence to the given system for all instantiations of local transition functions.

The central notion with respect to black-box simulations is the update graph. A pair (G, α) is called an *update graph* if $G = (V, E)$ is an undirected graph and $\alpha : \{1, \ldots, T\} \to \mathcal{P}(V)$. An update graph is called *sequential* if and only if α is a sequential update schedule, i.e., $\alpha : \{1, \ldots, T\} \to V$.

In the following we present an alternative characterization of update graphs and discuss black-box simulation equivalences of discrete dynamical systems

3.5.1 The poset model for graphs

The poset model for graphs has been introduced in [98]. A triple (G, P, β) is called a *poset model for the graph G* (or *pograph*) if $G = (V, E)$ is an undirected graph, $P = (P, \leq)$ is a finite poset, and $\beta : P \to V$ is a mapping satisfying for all $i, j \in P$:

– if $i \prec_P j$ then $\{\beta(i), \beta(j)\} \in E$,
– if $\{\beta(i), \beta(j)\} \in E$ then $i \leq j \vee j \leq i$.

The pograph approach is motivated by the following fact [98, Theorem 3.1].

Theorem 3.16 (Laubenbacher & Pareigis [98]). *Let (G, α) be a sequential update graph with graph $G = (V, E)$ and update schedule $\alpha : \{1, \ldots, T\} \to V$. Then there exist a finite poset P with $\|P\| = T$, a mapping $\beta : P \to V$, and a bijective and monotone mapping $\gamma : P \to \{1, \ldots, T\}$ such that (G, P, β) is a pograph and $\alpha = \beta \circ \gamma^{-1}$.*

The proof of theorem provides the following canonical representation of sequential update graphs. Let (G, α) be a sequential update graph such that $G = (V, E)$ and α is of order T. We define a poset $(P(G, \alpha), \leq_\alpha)$ as follows:

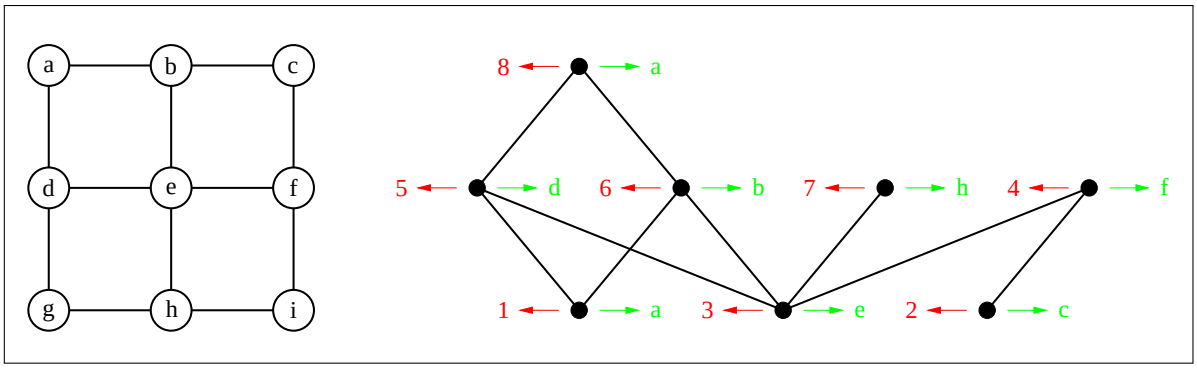


Figure 3.3. A pograph

$$P(G,\alpha) =_{\text{def}} \{1,\dots,T\}$$

and $\leq_\alpha$ is the reflexive and transitive closure of the precedence relation $\prec_\alpha$ on $P(G,\alpha)$ defined for all $i,j \in P(G,\alpha)$ by

$$i \prec_\alpha j \iff_{\text{def}} i \leq j \text{ and } \{\alpha(i),\alpha(j)\} \in E.$$

Theorem 3.16 states that for any sequential update graph (G,α) the relation $\leq_\alpha$ is a partial order.

Example 3.17. Figure 3.3 shows an example of the pograph construction from [98]. On the left-hand side, an undirected graph G with vertex set $\{a,b,c,d,e,f,g,h,i\}$ is drawn. The update schedule α is given by a tuple (a,c,e,f,d,b,h,a), i.e., $\alpha(1)=a, \alpha(2)=c,\dots,\alpha(8)=a$. On the right-hand side, the poset $P(G,\alpha)$ is shown. The red arrows indicate the mapping γ and the green arrows indicate the mapping β mentioned in Theorem 3.16. Notice that $\beta\circ\gamma^{-1}=\alpha$.

Proposition 3.18 (Laubenbacher & Pareigis [98]). *Suppose that* $\big((G,P,\beta),\gamma\big)$ *and* $\big((G,P',\beta'),\gamma'\big)$ *are pairs of pographs on the same graph* G *and mappings* $\gamma : P \to \{1,\dots,T\}$ *and* $\gamma' : P' \to \{1,\dots,T\}$ *such that* γ *and* γ' *are bijective and preserve orders. Assume that* $\beta\circ\gamma^{-1} = \beta'\circ\gamma'^{-1}$. *Then, there exists a unique isomorphism* $\delta : P \to P'$ *such that* $\gamma = \gamma'\circ\delta$ *and* $\beta = \beta'\circ\delta$.

3.5.2 Black-box simulation of sequential systems

The pograph construction gives rise to the following update schedule. Let P be any finite poset. As usual the *height* $h_P(i)$ of an element i in P is defined to be the maximum length of a decreasing chain with i as its maximal element. More specifically, we define

$$h_P(i) =_{\text{def}} \max\ \Big\{\ r\ \Big|\ (\exists i_1,\dots,i_r \in P)\big[i_r = i\ \wedge\ (\forall 1 \leq \nu < r)[\ i_\nu < i_{\nu+1}\ ]\ \big]\ \Big\}.$$

Note that, since P is finite, h_P is well-defined. Furthermore, $h_P(i) \geq 1$. The height $h(P)$ of P is defined to be the length of the longest chain in P, i.e.,

$$h(P) =_{\text{def}} \max_{i\in P} h_P(i).$$

Using these definitions, we define for a pograph (G, P, β) the *level-wise schedule* λ_P for all $i \in P$:

$$\lambda_P(i) =_{\text{def}} \{\ \beta(j) \mid j \in P \text{ and } h_P(j) = i\ \}.$$

Note that in general, λ_P is a non-sequential update schedule.

Example 3.19. The level-wise schedule for the pograph (G, P, β) in Figure 3.3 is $\lambda_P(1) = \{a, c, e\}$, $\lambda_P(2) = \{b, d, f, h\}$, and $\lambda_P(3) = \{a\}$.

In a certain sense level-wise schedules are best update schedules for sequential update graphs. This is based on a notion of (computational) equivalence for update schedules. Let $G = (V, E)$ be an undirected graph. Let $\alpha : \{1, \ldots, T\} \to \mathcal{P}(V)$ and $\alpha' : \{1, \ldots, T'\} \to \mathcal{P}(V)$ be two update schedules. We say that α and α' are *equivalent on* G if and only if for all domains $\mathcal{D}$ and for all sets $\{f_1, \ldots, f_n\}$ of $\mathcal{D}$-functions,

$$\mathbf{F}_{(G,\{f_1,\ldots,f_n\},\alpha)} = \mathbf{F}_{(G,\{f_1,\ldots,f_n\},\alpha')}.$$

Lemma 3.20 (Laubenbacher & Pareigis [98]). *Let (G, P, β) be a pograph, $\|P\| = T$. Let $\gamma, \gamma' : P \to \{1, \ldots, T\}$ be bijective and monotone mappings. Then, $\beta \circ \gamma^{-1}$ and $\beta \circ \gamma'^{-1}$ are equivalent on G.*

Note that by definition both update schedules are sequential schedules.

We aim at proving that the level-wise schedule is a standard way to obtain a non-sequential schedule equivalent to a given sequential update schedule. The next proposition is useful to sequentialize non-sequential parts of an update schedule.

Proposition 3.21. *Let $S = (G, \{f_1, \ldots, f_n\})$ be a dynamical system having the vertex set V. Let $U \subseteq V$ be an independent set in G. Then, for all $\alpha : \{1, \ldots, T\} \to \mathcal{P}_+(U)$ such that $2 \leq T \leq \|U\|$, $\alpha(i) \cap \alpha(j) = \emptyset$ if $i \neq j$, and $\alpha(1) \cup \cdots \cup \alpha(T) = U$,*

$$\mathbf{F}_S[U] = \prod_{i=1}^{T} \mathbf{F}_S[\alpha(i)].$$

Proof. Follows easily from the definition of global transition functions by noting that the arguments to the local transition function of a vertex in an independent set do not depend on the values of the other vertices in the independent set. ❑

Proposition 3.22. *Let (G, α) be a sequential update graph. Then, α and $\lambda_{P(G,\alpha)}$ are equivalent on G.*

Proof. Let (G, α) be an update graph with a graph $G = (V, E)$ and an update schedule $\alpha : \{1, \ldots, T\} \to V$. Suppose the vertices of V are totally ordered. Then, as remarked above $(G, P(G, \alpha), \alpha)$ is a pograph. Let γ be the identity function which is obviously bijective and monotone. Define $\gamma' : P(G, \alpha) \to \{1, \ldots, T\}$ for all $j \in P(G, \alpha)$ by

$$\gamma'(j) \quad =_{\text{def}} \quad \left\|\left\{k \;\middle|\; h_{P(G,\alpha)}(k) < h_{P(G,\alpha)}(j) \text{ or } h_{P(G,\alpha)}(k) = h_{P(G,\alpha)}(j) \ \wedge\ \alpha(k) \leq \alpha(j)\right\}\right\|$$

In other words, γ' is built by sorting each level of $P(G, a)$ according to the total ordering of V and concatenating these totally ordered pieces ascendingly with respect to the heights.

Clearly, $\gamma' : P(G,\alpha) \to \{1,\ldots,T\}$ is bijective and monotone on $P(G,\alpha)$. By Proposition 3.20, $\alpha = \alpha \circ \gamma^{-1}$ and $\alpha \circ \gamma'^{-1}$ are equivalent on G. Furthermore, note that by construction of $\lambda_{P(G,\alpha)}$ each set $\lambda_{P(G,\alpha)}(i)$ is an independent set in G. Hence, by iteratively applying Proposition 3.21 we conclude that $\alpha \circ \gamma'^{-1}$ and $\lambda_{P(G,\alpha)}$ are equivalent on G. Consequently, α and $\lambda_{P(G,\alpha)}$ are equivalent on G. ❑

An easy observation shows that in fact, the level-wise schedule has the least order among all equivalent update schedules.

Proposition 3.23. *Let (G,α) be an update graph. For any update schedule α' which is equivalent to α on G it holds that*

$$\mathrm{ord}(\alpha') \geq \mathrm{ord}(\lambda_{P(G,\alpha)}).$$

Proof. Let (G,α) be an update graph such that $G = (V,E)$ is an undirected graph on n vertices and α is of order T. Consider the domain $\mathcal{D} = \{1,\ldots,T\}$. Assume $\mathcal{D}$ is totally ordered with respect to the standard ordering. For each vertex $i \in V$ define the local transition function f_i by

$$f_i(x_{i_1},\ldots,x_{i_k}) =_{\mathrm{def}} 1 + \max_{1\leq j\leq k} x_{i_j},$$

where $\{i_1,\ldots,i_k\} = N_G^0(i)$. It is easily seen that the maximal component of the configuration $\mathbf{F}_{(G,\{f_1,\ldots,f_n\},\lambda_{P(G,\alpha)})}(0,\ldots,0)$ is equal to the height of $P(G,\alpha)$. On the other hand, for any other update schedule α' the components of the configuration $\mathbf{F}_{(G,\{f_1,\ldots,f_n\},\lambda_{P(G,\alpha)})}(0,\ldots,0)$ have values at most $\mathrm{ord}(\alpha')$. Hence, for α' to be equivalent to $\lambda_{P(G,\alpha)}$ (and thus, equivalent to α by Proposition 3.22) it must hold that $\mathrm{ord}(\alpha') \geq \mathrm{ord}(\lambda_{P(G,\alpha)})$. ❑

To summarize, level-wise schedules provide the most efficient parallel black-box simulations of sequential dynamical systems.

3.5.3 Black-box sequentializability

We mention that, not surprisingly, there are non-sequential update schedules which cannot be sequentialized by equivalent sequential update schedules.

Proposition 3.24. *Let (G,α) be an update graph with a graph $G = (V,E)$ and an update schedule α of order T such that there is an edge $e \in E$ with $e \subseteq \alpha(i)$ for some $i \in \{1,\ldots,T\}$. Then, there is no sequential update schedule α' equivalent to α on G.*

Proof. First consider the following simple dynamical system $S = (H,\{f_1,f_2\})$ where $H = (\{1,2\},\{\{1,2\}\})$ is a graph consisting of a single edge and the local transition functions f_j, $j \in \{1,2\}$, are given for $x_1,x_2 \in \{0,1\}$ by:

$$f_j(x_1,x_2) =_{\mathrm{def}} \begin{cases} 1 & \text{if } x_1 = x_2 \\ 0 & \text{otherwise} \end{cases}$$

The global transition functions are shown in Figure 3.4. Now suppose α_1 is an update schedule on H of order T_1 such that $\{1,2\} = \alpha_1(i)$ for some $i \in \{1,\ldots,T_1\}$. Then, it is easily seen from Figure 3.4 that $\|\mathbf{F}^{-1}_{(S,\alpha_1)}(1,1)\| \geq 2$: Indeed, $(1,1) \in F^{-1}_{(S,\alpha_1)}(1,1)$ and let i be minimal such that $\{1,2\} = \alpha_1(i)$. That is, the sequence $(\alpha_1(1),\ldots,\alpha_1(i-1))$ can be viewed as a word over

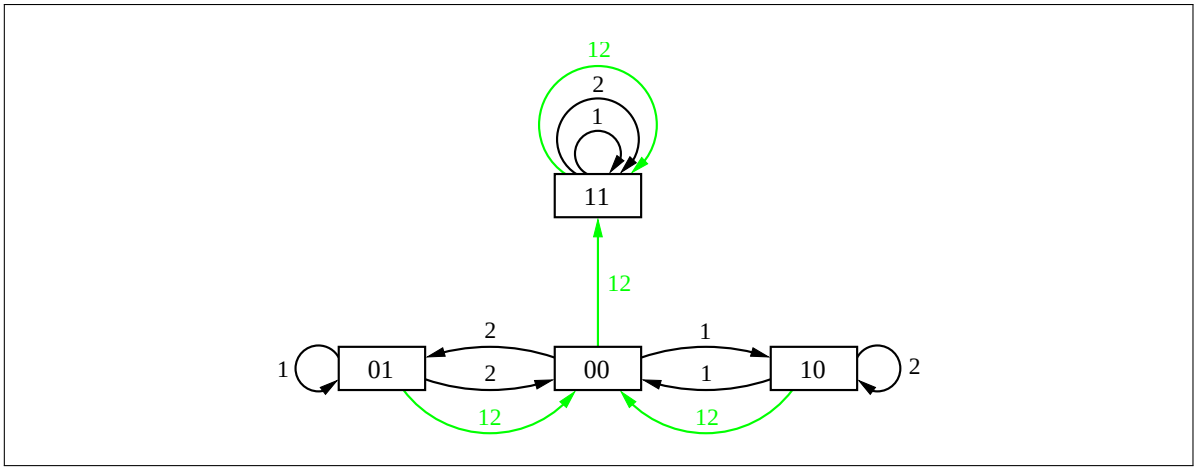


Figure 3.4. Global transition functions without sequential schedules

$\{1,2\}$. We define φ to be the mapping given for $t_1 \dots t_k \in \{1,2\}^*$ by the following recursion: for $k > 1$,

$$\varphi(t_1 \dots t_k) \;=_{\text{def}}\; \varphi(t_1 \dots t_{k'-1})$$

where $k' < k$, $t_{k'} = t_k$, and $t_\ell \neq t_k$ if $k' < \ell < k$; and

$$\begin{aligned} \varphi(\varepsilon) &=_{\text{def}} (0,0), \\ \varphi(1) &=_{\text{def}} (1,0), \\ \varphi(2) &=_{\text{def}} (0,1). \end{aligned}$$

Clearly, for $\varphi(\alpha_1(1) \dots \alpha_1(i-1)) = (a,b) \in \{(0,0),(0,1),(1,0)\}$ we have $\mathbf{F}_{(S,\alpha_1)}(a,b) = (1,1)$. Thus, (a,b) is a second element in the pre-image of $\mathbf{F}_{(S,\alpha_1)}$. On the other hand, it is easy to see that for each sequential update schedule α_2 on H, $F^{-1}_{(S,\alpha_2)}(1,1) = \{(1,1)\}$. Hence, there is no sequential update schedule α_2 equivalent to α_1 on H.

Now consider the update graph (G,α) with $G = (V,E)$ and α of order T. Let $e = \{u,v\} \in E$ be the edge such that $\{u,v\} \subseteq \alpha(i)$ for some $i \in \{1,\dots,T\}$. Setting $f_w = c_0$ for $w \notin \{u,v\}$ we can easily simulate the dynamical system S from above at the edge $\{u,v\}$. It follows that there is no sequential update schedule equivalent to α on G. ❑

We conclude with a characterization of sequentializability of dynamical systems.

Corollary 3.25. *Let (G,α) be an update graph. Then, there is a sequential update schedule α' equivalent to α on G if and only if for each $i \in \{1,\dots,\mathrm{ord}(\alpha)\}$, $\alpha(i)$ is an independent set in G.*

3.6 Summary

In this chapter we presented a mathematical framework for analyzing discrete dynamical systems. Graph minors were introduced as a graph-theoretical concept to express neighborhood dependencies between elements. Functional interdependence is classified according to clones. Certainly, other schemes can be devised for system classifications. For instance, network

classes closed under contraction of maximal cliques would be a reasonable scheme. However, we do not know any theorem similar to Theorem 3.4. The proposed framework based on forbidden minors and and clone theory is capable to serve as a reference model for complex-systems analysis. In the next chapter we will see that our dynamical systems framework admits precise and complete complexity characterizations for fixed-point analyses.

4. Fixed-point analysis

4.1 Background

A fundamental behavioral pattern for discrete dynamical systems are fixed points (a.k.a. attractors, equilibria, homogeneous states). A configuration of a system is a fixed point if the states of the system's elements left unchanged if the system updates the states. Fixed points are the most stable configurations of complex systems and are typically associated with a meaningful system behavior. For instance, in distributed systems, such as communication networks, a fixed point indicates that a certain information spreading process, such as route propagation, has converged. Many ex post analysis problem for complex systems are in large part fixed-point oriented, when a stable configuration has broken down or some stable configuration should be reached.

Accordingly, we are interested in the two most basic computational analysis problems involving fixed points:

– Fixed-Point Existence: *given a system is there a fixed point for the system, and if the answer is "yes", how can we find a fixed point?*
– Fixed-Point Counting: *given a system how many fixed points are there for the system?*

A series of recent papers has been devoted to a partial identification of finite systems with tractable/intractable fixed-point analyses (see, e.g., [15, 148, 145, 147]). We summarize the most significant results from the literature.

As shown in [15] tractable cases for the Fixed-Point Existence problem are constituted by systems having local transition functions from the following sets:

– linear functions (over arbitrary domains)
– monotone functions (over arbitrary finite domains)
– functions computed by gates (of unbounded fan-in) from {AND, OR, NAND, NOR}

In contrast, finding fixed points is intractable for dynamical systems with local transition functions in one of the following sets [15]:

– symmetric boolean functions
– boolean functions computed by gates (of unbounded fan-in) from {AND, XNOR}
– boolean functions computed by gates (of unbounded fan-in) from {NAND, XOR}
– boolean functions computed by gates (of unbounded fan-in) from {XOR, XNOR}
– boolean functions computed by gates (of unbounded fan-in) from {NOR, XOR}

All these results refer to dynamical systems over unrestricted network classes. Note that except the linear and monotone functions none of these function classes is a clone. If we take the closure under superposition of those classes defined by gates then the intractability results still hold for the larger classes obtained. Actually, these classes contain all boolean functions.

The Fixed-Point Counting problem is apparently much harder from a computational point of view. No tractability results have been reported in the literature so far. On the other side, intractability has been proven for dynamical systems having:

– symmetric boolean local transition functions and networks of maximum degree at most five [148]
– monotone boolean local transition functions and star networks [145, 147]
– monotone boolean local transition functions and networks of maximum degree at most three [146]

The corresponding intractability proofs rely on succinct representations of boolean functions by formulas and circuits. This equally applies to the intractability results for the Fixed-Point Existence problem listed above. The proofs do not directly transfer validity to the case of lookup-table representations.

In this chapter we present a precise characterization of the *islands of tractability*, with respect to the framework introduced in the last chapter, for finding and counting fixed points in dynamical systems over the boolean domain $\{0, 1\}$ for lookup-table, formula, and circuit representations. Further, we briefly discuss the possibility of proving a dichotomy theorem for the Fixed-Point Existence problem over larger domains as well.

4.2 Terminology

The central notion for our study of dynamical systems is the concept of a fixed point, i.e., a configuration which does not change under any global behavior of the system.

Definition 4.1. Let $S = (G, \{f_i \mid i \in V\})$ be a dynamical system over domain $\mathcal{D}$.

1. A configuration $\boldsymbol{x} \in \mathcal{D}^n$ is said to be a *local fixed point of S for $U \subseteq V$* if and only if $\mathbf{F}_S[U](\boldsymbol{x}) = \boldsymbol{x}$.
2. A configuration $\boldsymbol{x} \in \mathcal{D}^n$ is said to be a *fixed point of S* if and only if $\boldsymbol{x}$ is a local fixed point of S for V.

The following useful proposition is easily seen.

Proposition 4.2. *Let $S = (G, \{f_i \mid i \in V\})$ be a dynamical system over domain $\mathcal{D}$.*

1. *If the configuration $\boldsymbol{x} \in \mathcal{D}^n$ is a local fixed point of S for $U' \subseteq V$ and $\boldsymbol{x}$ is a local fixed point of S for $U'' \subseteq V$, then $\boldsymbol{x}$ is a local fixed point of S for $U' \cup U''$.*
2. *A configuration $\boldsymbol{x} \in \mathcal{D}^n$ is a local fixed point of S for $U \subseteq V$ if and only if $\boldsymbol{x}$ is a local fixed point for all $U' \subseteq U$.*

Notice that the second item of Proposition 4.2 shows that the concept of fixed points is independent of update schedules.

Corollary 4.3. *Let $S = (G, \{f_i \mid i \in V\})$ be a dynamical system over domain $\mathcal{D}$. A configuration $\boldsymbol{x} \in \mathcal{D}^n$ is a fixed point of S if and only if for all update schedules $\alpha : \{1, \ldots, T\} \to \mathcal{P}(V)$, it holds that $\mathbf{F}_{(S,\alpha)}(\boldsymbol{x}) = \boldsymbol{x}$.*

4.3 The Fixed-Point Existence problem

4.3.1 Definition

In this section we are interested in the computational complexity of the following problem. Let $\mathcal{F}$ be a class of finite functions and let $\mathcal{G}$ be a class of graphs.

Problem:	FIXED POINTS$(\mathcal{F}, \mathcal{G})$
Input:	A dynamical system $S = (G, \{f_1, \ldots, f_n\})$ such that $G \in \mathcal{G}$ and for all $i \in \{1, \ldots, n\}$, $f_i \in \mathcal{F}$
Question:	Does S have a fixed point? (And if the answer is "Yes", find a fixed point.)

The complexity of the problem depends on how transition functions are represented. We consider the following three cases:

- FIXED POINTS$_{\mathrm{T}}$ is the problem where local transition functions are given by lookup tables.
- FIXED POINTS$_{\mathrm{F}}$ is the problem where local transition functions are given by formulas (over certain bases), and circuit representations.
- FIXED POINTS$_{\mathrm{C}}$ is the problem where local transition functions are given by circuits (over certain bases).

Note that the size of a dynamical systems $S = (G, F)$ is defined to be $|S| =_{\mathrm{def}} \|V\| + \|E\| + \sum_{i \in V} |f_i|$. The size $|S|$ thus depends on the sizes $|f_i|$ of function representations. In case of lookup-table representations we obtain that

$$|S| = \Theta\left(\sum_{i \in V} (1 + d_i) \cdot 2^{1+d_i}\right).$$

Here, we assume that for each vertex there is a table with 2^{1+d_i} rows of length $1 + d_i$. Of course, more compact representations exist, e.g., bit-vectors. However, this is not critical for complexities up to polynomial factors. The size of a formula over a certain finite logical basis is defined to be the number of symbols from the basis used to encode the formula plus the number of variables. The size of (unbounded fan-in) circuits over a certain finite logical basis is defined to be the number of wires connecting the gates of the circuit. All circuits appearing in the proofs have bases with maximum fan-in 3. In these cases the size of the circuit can be taken as the number of gates from the basis plus the number of input gates.

It is clear that all problem versions belong to NP. We say that a problem is *intractable* if it is NP-complete, and it is *tractable* if it is solvable in polynomial time.

4.3.2 Boolean systems with local transitions given by lookup tables

The smallest non-trivial domain for finite functions is the boolean domain $\{0, 1\}$. A dynamical systems is called *boolean* if the domain of all its local transition functions is $\{0, 1\}$. The main result of this subsection is the following dichotomy result for boolean dynamical systems.

Theorem 4.4. *Let $\mathcal{F}$ be a boolean clone and let $\mathcal{G}$ be a class of graphs closed under taking minors. If $\mathcal{F} \supseteq \mathrm{D}$ and $\mathcal{G} \supseteq \mathrm{Forb}_{\preceq}(K_{3,3}, K^5)$ then* FIXED POINTS$_{\mathrm{T}}(\mathcal{F}, \mathcal{G})$ *is intractable. Otherwise,* FIXED POINTS$_{\mathrm{T}}(\mathcal{F}, \mathcal{G})$ *is tractable.*

We postpone the proof to the end of this subsection when we have proved a number of complexity results for several classes of systems.

Dynamical systems with a tractable fixed-point analysis

We first summarize the tractable classes of local transition functions of systems on arbitrary underlying networks. The results presented next are well-known or follow easily from the definitions.

Proposition 4.5 (Barrett et al. [15]). FIXED POINTS$_{\mathrm{T}}(\mathrm{M}, \mathrm{Forb}_{\preceq}(\emptyset))$ *is solvable in polynomial time.*

Proposition 4.6 (Barrett et al. [15]). FIXED POINTS$_{\mathrm{T}}(\mathrm{L}, \mathrm{Forb}_{\preceq}(\emptyset))$ *is solvable in polynomial time.*

Proposition 4.7. FIXED POINTS$_{\mathrm{T}}(\mathrm{R}_1, \mathrm{Forb}_{\preceq}(\emptyset))$ *is solvable in polynomial time.*

Proposition 4.8. FIXED POINTS$_{\mathrm{T}}(\mathrm{R}_0, \mathrm{Forb}_{\preceq}(\emptyset))$ *is solvable in polynomial time.*

Notice that D is the only remaining class which does not contain all boolean functions.

Next we restrict network classes. To identify tractable cases, we express the fixed-point existence problem as a constraint satisfaction problem.

A *constraint satisfaction problem* (CSP) [111] consists of triples $(X, \mathcal{D}, \mathcal{C})$, where

– $X = \{x_1, \ldots, x_n\}$ is the set of variables,
– $\mathcal{D}$ is the domain of the variables,
– $\mathcal{C}$ is a set of constraints $Rx_{i_1}, \ldots, x_{i_k}$ having associated corresponding relations $R_{i_1,\ldots,i_k}$.

The set $\mathcal{C}$ of constraints is listed by pairs $\langle Rx_{i_1}, \ldots, x_{i_k}, R_{i_1,\ldots,i_k}\rangle$. A solution for $(X, \mathcal{D}, \mathcal{C})$ is an assignment $I : X \to \mathcal{D}$ such that $(I(x_{i_1}), \ldots, I(x_{i_k})) \in R_{i_1,\ldots,i_k}$ for all constraints $Rx_{i_1}, \ldots, x_{i_k} \in \mathcal{C}$. The (primal) *constraint graph* for $(X, \mathcal{D}, \mathcal{C})$ consists of the vertex set X and the edge set $\{\{x_i, x_j\} \mid x_i \text{ and } x_j \text{ occur in the same constraint of } \mathcal{C}\}$.

Theorem 4.9. *Let X be a planar graph. Then,* FIXED POINTS$_{\mathrm{T}}(\mathrm{BF}, \mathrm{Forb}_{\preceq}(X))$ *is solvable in polynomial time.*

Proof. We describe a general reduction from FIXED POINTS$(\mathcal{F}, \mathcal{G})$ to constraint satisfaction problems. Suppose we are given a dynamical system $S = (G, \{f_1, \ldots, f_n\})$. Let $G = (V, E)$ be the underlying network. Without loss of generality, we may assume that G does not have isolated vertices, i.e., $d_i \geq 1$ for all $i \in V$. Define $\mathrm{CSP}(S) = (X, \mathcal{D}, \mathcal{C})$ to be the constraint satisfaction problem specified as follows:

$$
\begin{aligned}
X &=_{\mathrm{def}} \{x_1, \ldots, x_n\} \\
\mathcal{D} &=_{\mathrm{def}} \textstyle\bigcup_{i \in V} \mathcal{D}_i \text{ where for all } i \in V, \\
&\qquad \mathcal{D}_i =_{\mathrm{def}} \{(I, i) \mid I : N_G^0(i) \to \{0,1\} \text{ such that } f_i(I(i_0), \ldots, I(i_k)) = I(i) \\
&\qquad\qquad \text{where } \{i_0, \ldots, i_k\} = N_G^0(i)\} \\
\mathcal{C} &=_{\mathrm{def}} \{Ex_ix_j \mid \{i,j\} \in E(G) \text{ and } i \leq j\} \text{ where for all } i \leq j, \\
&\qquad E_{ij} =_{\mathrm{def}} \{\ \{(I_i, i), (I_j, j)\} \mid\ I_i(k) = I_j(k) \text{ for all } k \in N_G^0(i) \cap N_G^0(j)\ \}
\end{aligned}
$$

Let $n = \|V\|$ and $m = \|E\|$. By construction of $\mathrm{CSP}(S)$ we obtain that $\|X\| = n$ and that the number of constrains in $\mathcal{C}$ is just m. The size of the domain $\mathcal{D}$ is at most proportional to $\sum_{i \in V}(1 + d_i) \cdot 2^{1+d_i}$, and the size of the set of constraint relations can bounded by

$$c \cdot \textstyle\sum_{\{i,j\} \in E}(2 + d_i + d_j) \cdot 2^{1+d_i} \cdot 2^{1+d_j} \leq c \cdot \left(\sum_{i \in V}(1 + d_i) \cdot 2^{1+d_i}\right)^2$$

for some constant $c > 0$. The latter holds because $d_i \geq 1$ for all $i \in V$. All in all, this easily implies that $|\mathrm{CSP}(S)| = O(|S|^2)$. Hence, $\mathrm{CSP}(S)$ is computable in polynomial time in the size of S. We have to show that

$$S \text{ has a fixed point} \quad \Longleftrightarrow \quad \mathrm{CSP}(S) \text{ has a satisfying assignment.}$$

We prove both directions of the equivalence separately.

For ($\Rightarrow$), suppose the configuration $I : V \to \{0, 1\}$ is a fixed point of S. Define an assigment $I' : \{x_1, \ldots, x_n\} \to \mathcal{D}$ by

$$I'(x_i) =_{\mathrm{def}} I[N_G^0(i)].$$

We show that I' satisfies all constraints. For all $i \in V$, let I_i denote $I[N_G^0(i)]$. Let Ex_ix_j be any constraint of $\mathrm{CSP}(S)$ and let E_{ij} be the relation associated with Ex_ix_j. By definition, $\{i, j\} \in E$. Since I is a fixed point, it holds that (I_i, i) and (I_j, j) belong to $\mathcal{D}$. For all $k \in N_G^0(i) \cap N_G^0(j)$ we obtain

$$I_i(k) = I'(x_i)(k) = I'(x_j)(k) = I_j(k).$$

Thus, $\{(I_i, i), (I_j, j)\}$ lies in E_{ij}. This proves the direction from left to right.

For ($\Leftarrow$), suppose I is a satisfying assignment for $\mathrm{CSP}(S)$. Let $(I_i, i) \in \mathcal{D}$ be the pair which I assigns to the variable $x_i \in V$. Define a configuration $I' : V \to \{0, 1\}$ by

$$I'(i) =_{\mathrm{def}} I_i(i).$$

We are done if we are able to show that the following is true for all $i \in V$:

$$I'[N_G^0(i)] = I_i \tag{4.1}$$

Since for all $i \in V$, $f_i(I_i(i_0), \ldots, I_i(i_k)) = I_i(i)$ where $\{i_0, \ldots, i_k\} = N_G^0(i)$, Eq. (4.1) implies that I' is a fixed point of S. To verify the equation we consider i and all neighbors of i individually. Let $j \in N_G^0(i)$. If $j = i$, there is nothing to show. So let $j \neq i$. Since $\{i, j\} \in E(G)$ and since I is a satisfying assignment, $I_i(k) = I_j(k)$ holds for all $k \in N_G^0(i) \cap N_G^0(j)$. Thus, we obtain $I'(j) = I_j(j) = I_i(j)$. This shows the correctness of Eq. (4.1) for all $i \in V$. Hence, the direction from right to left is proved.

We thus have established a polynomial-time many-one reduction between the Fixed-Point Existence problem and constraint satisfaction problems. Moreover, it is easy to see that the graph G of a given dynamical system S is isomorphic to $\mathrm{CSP}(S)$'s constraint graph. Thus, if X is planar then $\mathrm{Forb}_{\preceq}(X)$ has bounded treewidth, and so the constraint graph of $\mathrm{CSP}(S)$ has bounded treewidth for each dynamical system $S = (G, \{f_1, \ldots, f_n\})$ such that $G \in \mathrm{Forb}_{\preceq}(X)$. The theorem follows from the well-known polynomial-time algorithms for constraint satisfaction problems having constraint graphs of bounded treewidth [58]. ❑

Dynamical systems with an intractable fixed-point analysis

We turn to the intractable cases of the fixed-point existence problem. As a first step, FIXED POINTS$(\mathcal{F}, \mathcal{G})$ is shown to be NP-complete for arbitrary boolean local transition functions and planar networks. Recall that similar results have only been shown for formula respresentation.

We use PLANAR 3SAT to prove NP-hardness. Let $H = C_1 \wedge \cdots \wedge C_m$ be a 3CNF having variables $x_1, \ldots, x_n$. The graph representation $\Gamma(H)$ of H is a bipartite graph consisting of the vertex set $\{x_1, \ldots, x_n, C_1, \ldots, C_m\}$ and all edges $\{x_i, C_j\}$ such that variable x_i appears as a literal in the clause C_j. A planar 3CNF is a 3CNF such that its graph representation is planar. PLANAR 3SAT is the problem to decide whether a given planar 3CNF is satisfiable. It is well known that PLANAR 3SAT is an NP-complete problem [100].

Theorem 4.10. FIXED POINTS$(\mathrm{BF}, \mathrm{Forb}_{\preceq}(K_{3,3}, K^5))$ *is* NP*-complete, even restricted to underlying networks having maximum vertex degree three.*

Note that for graphs having maximum degree at most two the problem is tractable.

Proof. Let $H = C_1 \wedge C_2 \wedge \cdots \wedge C_m$ be a 3CNF having variables $x_1, \ldots, x_n$ and a planar graph representation $\Gamma(H)$ where $V(\Gamma(H)) = U \cup W$, $U = \{x_1, \ldots, x_n\}$, and $W = \{C_1, \ldots, C_m\}$. Suppose that $V(\Gamma(H))$ is totally ordered such that the clause vertices of W come before the variable vertices of U.

We construct the following system. For the underlying network $G = (V, E)$, compute an embedding of $\Gamma(H)$ in the plane (in linear time), e.g., using the Lempel-Even-Cederbaum method [99]. Now replace the vertices of U in the following way. Let $x_i \in U$ and suppose that x_i's neighbors $C_{j_1}, \ldots, C_{j_r}$ in $\Gamma(H)$ are clockwise ordered with respect to the planar embedding, such that j_1 is minimal in $\{j \mid x_i \text{ appears as a literal in } C_j\}$. Replace the vertex $x_i \in U$ by a cycle $\{x_{i,j_1}, \ldots, x_{i,j_r}\}$ which is connected to the neighbors of x_i in $\Gamma(H)$ by having an edge $\{x_{i,j}, C_j\}$ whenever x_i appears as a literal in C_j. Let $G = (V, E)$ be the graph obtained after all replacements are made in this way. Clearly, G is planar, i.e., $G \in \mathrm{Forb}_{\preceq}(K_{3,3}, K^5)$, and can be computed in time polynomial in the size of H. Note that the maximum degree of a vertex in G is three. To complete the construction, we specify the local transition functions. Let $C_i \in W$ be a clause vertex. Let x_{i_1,j_1}, x_{i_2,j_2}, and x_{i_3,j_3} be the neighbors of C_i in G. Define the local transition function f_{C_i} by

$$f_{C_i}(I(C_i), I(x_{i_1,j_1}), I(x_{i_2,j_2}), I(x_{i_3,j_3})) =_{\mathrm{def}} \begin{cases} 1 & \text{if } I(x_{i_1,j_1}), I(x_{i_2,j_2}), I(x_{i_3,j_3}) \text{ is a satisfying} \\ & \text{assignment of } C_i \\ \mathrm{not}(I(C_i)) & \text{otherwise} \end{cases}$$

Let $x_{i,j} \in V$ be a variable vertex. Suppose that $\{C_j, x_{i,k_0}, \ldots, x_{i,k_r}\} = N^0_G(x_{i,j})$ where r is the degree of $x_{i,j}$ in G. Define the local transition function $f_{x_{i,j}}$ by

$$f_{x_{i,j}}(I(C_j), I(x_{i,k_0}), \ldots, I(x_{i,k_r})) =_{\mathrm{def}} \begin{cases} I(x_{i,j}) & \text{if } I(x_{i,k_0}) = \cdots = I(x_{i,k_r}) \\ \mathrm{not}(I(x_{i,j})) & \text{otherwise} \end{cases}$$

Let S_H denote the dynamical system $(G, \{f_v | v \in V\})$ constructed from any planar 3CNF H in the way just specified. Note that, since the maximum degree of any vertex in G is three, we can compute S_H in time polynomial in the size of H.

By construction of S_H, a configuration I is a fixed point of S_H if and only if I satisfies $I(x_{i,j_1}) = \cdots = I(x_{i,j_r})$ for all $i \in \{1,\ldots,n\}$ and, furthermore, $I(C_j) = 1$ for all $j \in \{1,\ldots,m\}$. Hence, it is easily seen that H has a satisfying assignment if and only if S_H has a fixed point. ❑

We extend the NP-completeness of the fixed-point existence problem to systems with selfdual transition functions and planar graphs. First, observe that the transition functions for clause variables are not selfdual. This implies that the construction used in the last theorem is not fully appropriate. However, arbitrary boolean functions can easily be embedded into a selfdual function of larger arity.

Proposition 4.11. *Let $n \in \mathbb{N}_+$. For each k-ary boolean function $f : \{0,1\}^k \to \{0,1\}$, the function $\mathrm{sd}_n(f) : \{0,1\}^{k+n+1} \to \{0,1\}$ defined for all $x_1,\ldots,x_k,y_1,\ldots,y_n,z \in \{0,1\}$ by*

$$\mathrm{sd}_n(f)(x_1\ldots,x_k,y_1,\ldots y_n,z) =_{\mathrm{def}} \begin{cases} f(x_1,\ldots,x_k) & \text{if } y_1 = \cdots = y_n = 0 \\ \mathrm{not}(f(\mathrm{not}(x_1),\ldots,\mathrm{not}(x_k))) & \text{if } y_1 = \cdots = y_n = 1 \\ \mathrm{not}(z) & \text{otherwise} \end{cases}$$

is selfdual.

Proof. Let f be any k-ary boolean function and let $n \in \mathbb{N}_+$. Let $x_1,\ldots,x_k,y_1,\ldots,y_n,z \in \{0,1\}$ be arbitrary arguments for $\mathrm{sd}_n(f)$. Assume there are i,j such that $1 \le i,j \le n$ and $y_i \neq y_j$. So, $\mathrm{not}(y_i) \neq \mathrm{not}(y_j)$ and we conclude that

$$\begin{aligned} &\mathrm{sd}_n(f)(x_1,\ldots,x_k,y_1,\ldots,y_n,z) \\ &\quad = \mathrm{not}(z) = \mathrm{not}\big(\mathrm{sd}_n(f)(\mathrm{not}(x_1),\ldots,\mathrm{not}(x_k),\mathrm{not}(y_1),\ldots,\mathrm{not}(y_n),\mathrm{not}(z))\big). \end{aligned}$$

Now assume that $y_1 = \cdots = y_n$. Without loss of generality, suppose $y_n = 0$. That is

$$\begin{aligned} \mathrm{sd}_n(f)(x_1,\ldots,x_k,y_1,\ldots,y_n,z) &= f(x_1,\ldots,x_k), \\ \mathrm{sd}_n(f)(\mathrm{not}(x_1),\ldots,\mathrm{not}(x_k),\mathrm{not}(y_1),\ldots,\mathrm{not}(y_n),\mathrm{not}(z)) &= \mathrm{not}(f(x_1,\ldots,x_k)). \end{aligned}$$

It follows that $\mathrm{sd}_n(f)(x_1,\ldots,x_k,y_1,\ldots,y_n,z)$ is selfdual. ❑

The usage of Proposition 4.11 introduces ambiguity to the set of fixed points.

Proposition 4.12. *Let $S = (G, \{f_i \mid i \in V\})$ be a dynamical system over $\{0,1\}$ so that all local transition functions f_i are selfdual. Let $U \subseteq V$. Then, a configuration $I : V \to \{0,1\}$ is a local fixed point of S for U if and only if the configuration $\overline{I} : V \to \{0,1\}$ defined by $\overline{I}(i) = \mathrm{not}(I(i))$ is a local fixed point of S for U.*

Proof. Immediate from the definitions. ❑

Theorem 4.13. Fixed Points$(\mathrm{D}, \mathrm{Forb}_{\preceq}(K_{3,3}, K^5))$ *is* NP*-complete.*

Proof. We reduce from Fixed Points$(\mathrm{BF}, \mathrm{Forb}_{\preceq}(K_{3,3}, K^5))$. Let $S = (G, \{f_i \mid i \in V\})$ be a dynamical system such that the underlying network G is planar. We construct another system $S' = (G', \{f'_i \mid i \in V'\})$ as follows. The network $G' = (V', E')$ is defined by:

$$\begin{aligned} V' &=_{\mathrm{def}} V \cup E \\ E' &=_{\mathrm{def}} E \cup \{\, \{i,e\} \mid i \in V, e \in E, i \in e \,\} \end{aligned}$$

It is easily seen that G' is planar as well. Suppose V' is ordered such that V is in the same ordering as for S, E is arbitrarily ordered, and E comes completely after V. The local transition functions of the vertices of V' are specified as follows. Suppose $i \in V$. Let $\{i_0, \ldots, i_k\} = N^0_{G'}(i) \cap V$ and let $\{e_1, \ldots, e_k\} = N^0_{G'}(i) \cap E$. We define the local transition function f'_i by

$$f'_i(x_{i_0}, \ldots, x_{i_k}, x_{e_1}, \ldots, x_{e_k}) =_{\text{def}} \mathrm{sd}_k(f_i)(x_{i_0}, \ldots, x_{i_k}, x_{e_1}, \ldots, x_{e_k}, x_i).$$

By Proposition 4.11 and since D is closed under identification of variables, f'_i is selfdual. Moreover, as the degree of i in V' is $2k$ where k is the degree of i in G, the function table can be computed in polynomial time. Now consider a vertex $e = \{i, j\} \in E$. Define the local transition function f'_e by

$$f'_{\{i,j\}}(x_i, x_j, x_{\{i,j\}}) =_{\text{def}} x_{\{i,j\}}.$$

Clearly, f'_e is selfdual. As the degree of $e \in E$ is two, the function table is trivially computable in polynomial time. It remains to show that

$$S \text{ has a fixed point} \iff S' \text{ has a fixed point.}$$

We prove both direction individually.

For ($\Rightarrow$), suppose the configuration $I : V \to \{0, 1\}$ is a fixed point of S, i.e., for all $i \in V$, it holds that $f_i(I(i_0), \ldots, I(i_k)) = I(i)$ where $\{i_0, \ldots, i_k\} = N^0_G(i)$. Define a configuration $I' : V' \to \{0, 1\}$ for $i \in V$ by

$$I'(i) =_{\text{def}} I(i)$$

and for $e \in E$ by

$$I'(e) =_{\text{def}} 0.$$

Consider $i \in V$. Assume $\{i_0, \ldots, i_k\} = N^0_{G'}(i) \cap V$ and $\{e_1, \ldots, e_k\} = N^0_{G'}(i) \cap E$. We obtain

$$\begin{aligned} &f'_i(I'(i_0), \ldots, I'(i_k), I'(e_1), \ldots, I'(e_k)) \\ &\quad = \mathrm{sd}_k(f_i)(I(i_0), \ldots, I(i_k), 0, \ldots, 0, I(i)) = f_i(I(i_1), \ldots, I(i_k)) = I(i) = I'(i). \end{aligned}$$

Thus, I' is a local fixed point for $i \in V$. Suppose $e = \{i, j\} \in E$. By definition, $f'_{\{i,j\}}(I'(i), I'(j), I'(\{i, j\})) = I'(\{i, j\})$. Thus, I' is a local fixed point for $e \in E$. Proposition 4.2 implies that I' is a fixed point of S'.

For ($\Leftarrow$), suppose the configuration $I' : V' \to \{0, 1\}$ is a fixed point of S'. Observe that for all $e, e' \in E \subseteq V'$, there is a walk in G' from e to e' alternating between vertices in V and E, i.e., there are vertices $p_0, \ldots, p_{2\ell} \in V'$ such that $\{p_i, p_{i+1}\} \in E'$ for all $0 \leq i < 2\ell$, $p_0 = e$, $p_{2\ell} = e'$, and for all $0 \leq j < \ell$, it holds that $p_{2j} \in E$ and $p_{2j+1} \in V$. Consider a vertex $p_{2j+1} \in V$. Let $\{i_0, \ldots, i_k\} = N^0_{G'}(p_{2j+1}) \cap V$ and $\{e_1, \ldots, e_k\} = N^0_{G'}(p_{2j+1}) \cap E$. Since I' is a fixed point of S, we have

$$\begin{aligned} I'(p_{2j+1}) &= f'_{p_{2j+1}}(I'(i_0), \ldots, I'(i_k), I'(e_1), \ldots, I'(e_k)) \\ &= \mathrm{sd}_k(f_{p_{2j+1}}(I'(i_0), \ldots, I'(i_k), I'(e_1), \ldots, I'(e_k), I'(i)) \end{aligned}$$

By definition of $\mathrm{sd}_k(f_{p_{2j+1}})$, it follows that $I'(e_1) = \cdots = I'(e_k)$. In particular, $I'(p_{2j}) = I'(p_{2j+2})$. This implies that for all $e, e' \in E$, it holds that $I'(e) = I'(e')$. By Proposition 4.12, we may assume that $I'(e) = 0$ for all $e \in E$. Define a configuration $I : V \to \{0, 1\}$ of S for all $i \in V$ by

$$I(i) = I'(i).$$

It is easily seen that I is a fixed point of S. ❑

Composing the big picture

We come back to the proof of the main result of the subsection. For convenience we state it once more.

Theorem 4.4. *Let $\mathcal{F}$ be a boolean clone and let $\mathcal{G}$ be a class of graphs closed under taking minors. If $\mathcal{F} \supseteq$ D and $\mathcal{G} \supseteq \mathrm{Forb}_{\preceq}(K_{3,3}, K^5)$ then* Fixed Points$_{\mathrm{T}}(\mathcal{F}, \mathcal{G})$ *is intractable. Otherwise,* Fixed Points$_{\mathrm{T}}(\mathcal{F}, \mathcal{G})$ *is tractable.*

Proof. If $\mathcal{F} \supseteq$ D and $\mathcal{G} \supseteq \mathrm{Forb}_{\preceq}(K_{3,3}, K^5)$ then Fixed Points$(\mathcal{F}, \mathcal{G})$ is NP-complete by Theorem 4.13. Suppose $\mathcal{F} \not\supseteq$ D or $\mathcal{G} \not\supseteq \mathrm{Forb}_{\preceq}(K_{3,3}, K^5)$. The maximal classes $\mathcal{F}$ that do not contain D are M, L, R_1, and R_0. For all these classes, by Propositions 4.5–4.8, the Fixed-Point Existence problem is solvable in polynomial time. It remains to consider the case that $\mathcal{G} \not\supseteq \mathrm{Forb}_{\preceq}(K_{3,3}, K^5)$. Suppose $\mathcal{G} = \mathrm{Forb}_{\preceq}(X_1, \ldots, X_n)$. Since $\mathcal{G} \not\supseteq \mathrm{Forb}_{\preceq}(K_{3,3}, K^5)$, there is an i such that X_i is planar. Since $\mathcal{G} \subseteq \mathrm{Forb}_{\preceq}(X_i)$, Theorem 4.9 shows that Fixed Points$_{\mathrm{T}}(\mathrm{BF}, \mathcal{G})$ is solvable in polynomial time. ❑

Figure 4.1 shows the islands of tractability according to Theorem 4.4 in Post's lattice.

4.3.3 Boolean systems with succinctly represented local transitions

In this subsection we prove a dichotomy theorem for the fixed-point existence problem when transitions are given by formulas or circuits. Recall that the size of a formula is the number of symbols from the basis used to encode the formula, the size of a circuit is the number of gates it consists of (including the input gates). Both succinct representations of functions lead to the same result.

Theorem 4.14. *Let $\mathcal{F}$ be a boolean clone and let $\mathcal{G}$ be a class of graphs closed under taking minors. If $\mathcal{F} \supseteq$ D and $\mathcal{G} \supseteq \mathrm{Forb}_{\preceq}(K^3, K^2 \oplus K^2)$ then* Fixed Points$_{\mathrm{F}}(\mathcal{F}, \mathcal{G})$ *is intractable. Otherwise,* Fixed Points$_{\mathrm{F}}(\mathcal{F}, \mathcal{G})$ *is tractable. Moreover, the same statement holds for* Fixed Points$_{\mathrm{C}}$.

Again we postpone the proof until we have proved a number of special results.

On the side of tractable cases, first note that Propositions 4.5–4.8 still hold for formulas and circuits. Actually, the results in [15] were stated for formulas. Furthermore, notice that circuits over the basis $\{\oplus, 0, 1\}$ can easily be transformed in polynomial time into equivalent formulas over the same basis. The following result provides the tractability limit for restricted network classes.

Theorem 4.15. *Let X be a graph having a vertex cover of size one. Then,* Fixed Points$_{\mathrm{F}}$ $(\mathrm{BF}, \mathrm{Forb}_{\preceq}(X))$ *and* Fixed Points$_{\mathrm{C}}(\mathrm{BF}, \mathrm{Forb}_{\preceq}(X))$ *are solvable in polynomial time.*

Proof. Suppose X has a vertex cover of size one. Then, there is an $r \in \mathbb{N}$ such that all graphs in $\mathrm{Forb}_{\preceq}(X)$ have maximum vertex degree r. Thus, if we compute lookup tables from formulas or circuits then each lookup table has at most 2^{r+1} entries. Hence, in polynomial time, we can transform each dynamical system S with a network in $\mathrm{Forb}_{\preceq}(X)$ and local transitions functions given by formulas or circuits into a dynamical system S' with the same networks and local transition functions given by lookup tables such that S and S' have the same fixed-point configurations. Moreover, since X is planar (note that $K^3 \preceq K_{3,3}$ and $K^2 \oplus K^2 \preceq K_{3,3}$

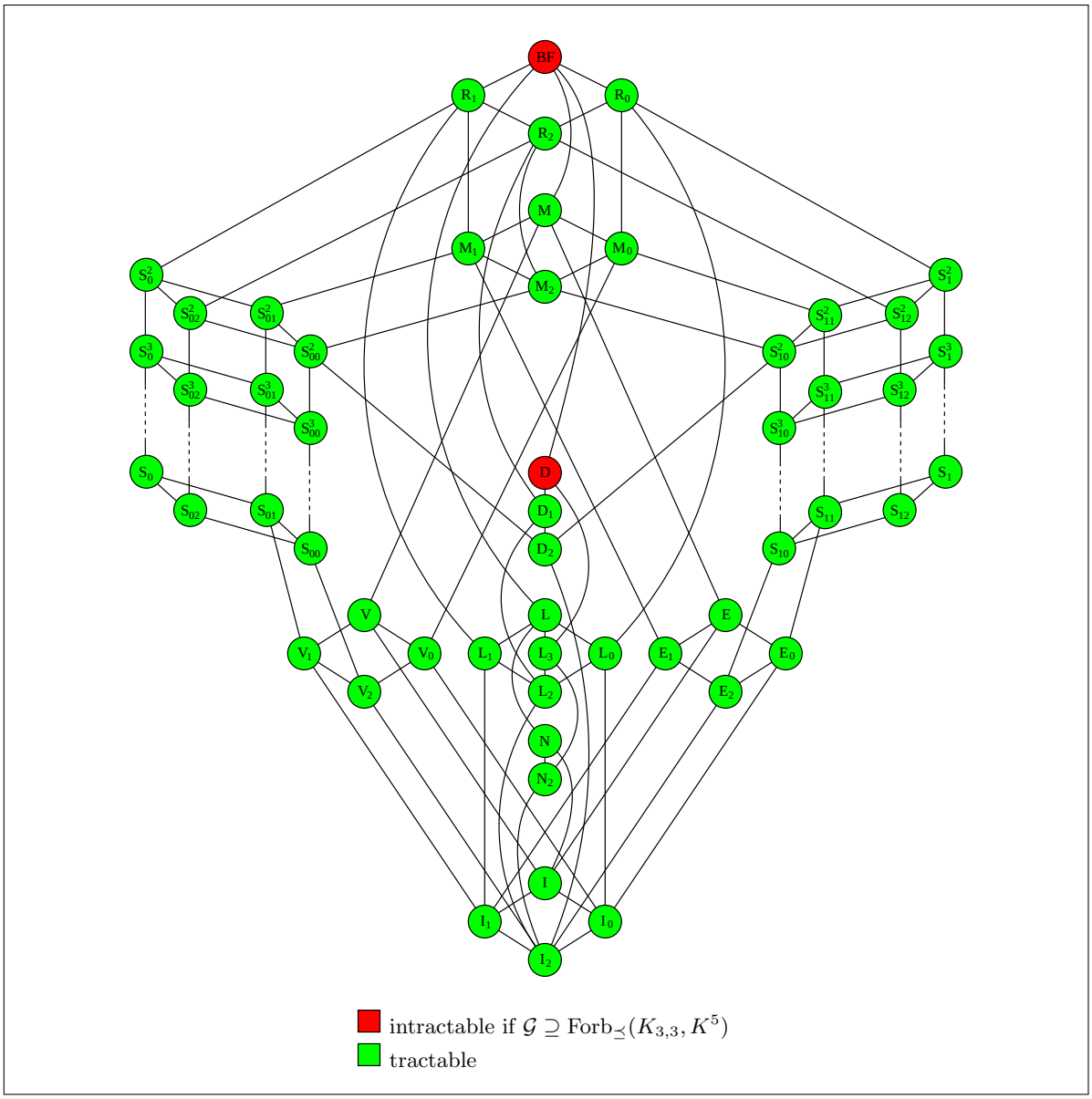


Figure 4.1. Islands of tractability for finding fixed points in boolean systems with local transition functions given by lookup tables

as well as $K^3 \preceq K^5$ and $K^2 \oplus K^2 \preceq K^5$), Theorem 4.9 implies polynomial-time solvability of Fixed Points$_{\mathrm{F}}(\mathrm{BF}, \mathrm{Forb}_{\preceq}(X))$ and Fixed Points$_{\mathrm{C}}(\mathrm{BF}, \mathrm{Forb}_{\preceq}(X))$. ❑

Theorem 4.16. Fixed Points$_{\mathrm{F}}(\mathrm{D}, \mathrm{Forb}_{\preceq}(K^3, K^2 \oplus K^2))$ *is* NP*-complete.*

Proof. The reduction is from 3SAT. We start with a description of a reduction to dynamical systems where each transition function is computed by a 3CNF. Suppose we are given a 3CNF $H = C_1 \wedge \cdots \wedge C_m$ having variables $x_1, \ldots, x_n$. Note that the formula $H' =_{\mathrm{def}} H \vee \overline{x_0}$, where x_0 is a new variable, satisfies for any assignment $I : \{x_0, x_1, \ldots, x_n\} \to \{0, 1\}$ that $f_{H'}(I(x_0), I(x_1), \ldots, I(x_n)) = I(x_0)$ if and only if $I(H) = 1$ and $I(x_0) = 1$. More-

over, H' is equivalent to a 4CNF which can be transformed in a 3CNF $\hat{H}$ with variables $x_0, x_1, \ldots, x_n, x_{n+1}, \ldots, x_{n+m}$ such that $\hat{H}$ is satisfiable with 1 assigned to x_0 if and only if H is satisfiable with 1 assigned to x_0. Define S_H to be the dynamical system $(G, \{f_0, \ldots, f_{n+m}\})$ consisting of the network $G = (V, E)$ where $V =_{\text{def}} \{0, 1, \ldots, n+m\}$ and $E =_{\text{def}} \{ \{0, i\} \mid i \in \{1, \ldots, n+m\} \}$. Thus, G is a star $K_{1,n+m}$, i.e., $G \in \text{Forb}_{\preceq}(K^3, K^2 \oplus K^2)$. The local transition functions are given as follows. For a vertex $i \in \{1, \ldots, n+m\}$, the local transition function f_i is defined to be computed by the formula $H(x_0, x_i) =_{\text{def}} x_i$. For the central vertex, the local transition function f_0 is given by the formula $\hat{H}(x_0, x_1, \ldots, x_{n+m})$ where the variable x_i stands for a vertex $i \in V$. Clearly, S_H can be computed in time polynomial in the size of H and we have that H is satisfiable if and only if S_H has a fixed point.

We now transform the dynamical system S_H into another system S'_H with selfdual local transition functions given by formulas over the corresponding basis. Note that D has a single basis function of arity three. Let D denote the corresponding ternary function symbol, i.e., the semantics of D is defined by

$$D(x, y, z) \equiv_{\text{def}} (x \wedge \overline{y}) \vee (x \wedge \overline{z}) \vee (\overline{y} \wedge \overline{z}).$$

Note that $D(x, x, y) \equiv \overline{y}$. We embed a 3CNF into a selfdual function, similarly to Proposition 4.11. That is, for an arbitrary 3CNF $H = C_1 \wedge \cdots \wedge C_m$ having variables $x_1, \ldots, x_n$, define

$$\text{sd}(H)(x_1, \ldots, x_n, z) =_{\text{def}} (H \wedge z) \vee \big(\neg H(\overline{x_1}, \ldots, \overline{x_n}) \wedge \overline{z}\big).$$

By induction over the number of clauses, we show that $\text{sd}(H)$ is equivalent to a formula built using D which is of polynomial size:

1. For the base of induction, suppose $m = 1$. Since we know how to express negation using D, without loss of generality, we assume that $H = (x_1 \vee x_2 \vee x_3)$. So,

   $$\text{sd}(H) \equiv \big((x_1 \vee x_2 \vee x_3) \wedge z\big) \vee \big((x_1 \wedge x_2 \wedge x_3) \wedge \overline{z}\big).$$

   By truth-table inspection we obtain that

   $$\text{sd}(H) \equiv \overline{D(\overline{z}, D(z, \overline{x_1}, \overline{x_2}), D(z, \overline{x_1}, \overline{x_3}))}.$$

2. For the induction step, suppose $m > 1$. Let $H' = C_1 \wedge \cdots \wedge C_m$ be a 3CNF over the variables $x_1, \ldots, x_n$. Define $H' =_{\text{def}} C_1 \wedge \cdots \wedge C_{\lfloor m/2 \rfloor}$ and $H'' =_{\text{def}} C_{\lfloor m/2 \rfloor + 1} \wedge \cdots \wedge C_m$. Some equivalent transformations show that

   $$\text{sd}(H) \equiv D(\overline{z}, \overline{\text{sd}(H')}, \overline{\text{sd}(H'')}).$$

   By induction hypothesis, $\text{sd}(H')$ and $\text{sd}(H'')$ can be expressed using D. Replacing $\text{sd}(H')$ and $\text{sd}(H'')$ (and the negation) gives the appropriate formula for $\text{sd}(H)$. Note that the recursion depth for formula replacement is logarithmic. It follows that the size of the formula for $\text{sd}(H)$ is $O(|H|^2)$.

Finally, we define the dynamical system S'_H to be specified as follows. Let the network $G' = (V', E')$ consist of

$$\begin{aligned} V' &=_{\text{def}} \{\underline{0}, 0, \ldots, n+m\}, \\ E' &=_{\text{def}} \{\{\underline{0}, i\} \mid i \in \{0, \ldots, n+m\} \}. \end{aligned}$$

Thus, G' is a star $K_{1,n+m+1}$, i.e., $G' \in \mathrm{Forb}_{\preceq}(K^3, K^2 \oplus K^2)$. The local transition function f_i for a vertex $i \in \{0, \ldots, n+m\}$ is given by

$$H(x_{\underline{0}}, x_i) =_{\mathrm{def}} D(x_{\underline{0}}, x_{\underline{0}}, D(x_{\underline{0}}, x_{\underline{0}}, x_i)).$$

Notice that $H(x_{\underline{0}}, x_i) \equiv x_i$. The local transition function $f_{\underline{0}}$ for the vertex $\underline{0}$ is given as follows. Recall that $\hat{H}$ is the 3CNF associated with the local transition function of vertex 0 in the system S_H. Then $f_{\underline{0}}$ is represented by the D-formula equivalent to $\mathrm{sd}(\hat{H})(x_0, \ldots, x_{n+m}, x_{\underline{0}})$. Clearly, S'_H can be computed in time polynomial in the size of H. Moreover, it is easy to verify that H is satisfiable if and only if S'_H has a fixed point. ❑

We combine Theorem 4.15 and Theorem 4.16 to prove Theorem 4.14.

Proof. (Theorem 4.14) If $\mathcal{F} \supseteq \mathrm{D}$ and $\mathcal{G} \supseteq \mathrm{Forb}_{\preceq}(K^3, K^2 \oplus K^2)$ then Fixed Points$(\mathcal{F}, \mathcal{G})$ is NP-complete by Theorem 4.16. Suppose $\mathcal{F} \not\supseteq \mathrm{D}$ or $\mathcal{G} \not\supseteq \mathrm{Forb}_{\preceq}(K^3, K^2 \oplus K^2)$. The maximal classes $\mathcal{F}$ that do not contain D are M, L, R_1, and R_0. For all these classes, by Propositions 4.5–4.8, the Fixed-Point Existence problem is solvable in polynomial time. It remains to consider the case that $\mathcal{G} \not\supseteq \mathrm{Forb}_{\preceq}(K^3, K^2 \oplus K^2)$. Suppose $\mathcal{G} = \mathrm{Forb}_{\preceq}(X_1, \ldots, X_n)$. Since $\mathcal{G} \not\supseteq \mathrm{Forb}_{\preceq}(K^3, K^2 \oplus K^2)$, there is an i such that X_i has a vertex cover of size one. Since $\mathcal{G} \subseteq \mathrm{Forb}_{\preceq}(X_i)$, Fixed Points$_{\mathrm{T}}(\mathrm{BF}, \mathcal{G})$ is solvable in polynomial time by Theorem 4.15. ❑

In Figure 4.2 the islands of tractability in Post's lattice are shown when local transitions functions are succinctly represented.

4.3.4 Extensions

We briefly discuss the possibility of proving a dichotomy result for larger domains. As mentioned already, the number of clones for ternary domains is continuum. This indicates that dichotomy theorems for finite domains of cardinality at least three are much harder to prove than in the case of the boolean domain where we have complete information on all clones.

In general, a clone of $\mathcal{D}$-functions is called *maximal* if it is maximal with respect to set inclusion among all clones not containing all $\mathcal{D}$-functions. In the last subsections we have seen that the boundary of the islands of tractability for the boolean domain goes along the maximal clones. In particular, no non-maximal boolean clone induces an NP-complete fixed-point existence problem. In this subsection we see that this is not the case for larger domains. Note that due to a theorem of Rosenberg [135] we have complete knowledge of the finitely many maximal clones for each domain. In the following we present complexity results for a subset of maximal clones. We focus on lookup-table representations.

Let $\mathbf{O}(\mathcal{D})$ denote the set of partial orders on $\mathcal{D}$ having a maximum and a minimum element. As we have seen in the boolean case polymorphisms of such relations are monotone functions (see Example 3.10). It is known that for each $R \in \mathbf{O}(\mathcal{D})$, $\mathrm{Pol}(R)$ is a maximal clone [105]. The following proposition is well-known from the literature.

Proposition 4.17 (Barrett et al. [15]). *Let $\mathcal{D}$ be a finite domain and let R be any relation in $\mathbf{O}(\mathcal{D})$. Then,* Fixed Points$_{\mathrm{T}}(\mathrm{Pol}(R), \mathrm{Forb}_{\preceq}(\emptyset))$ *is solvable in polynomial time.*

Suppose $\|\mathcal{D}\| = p^k$ for some $k \geq 1$ where p is a prime number. Let $G = (\mathcal{D}; 0, +, -)$ be an abelian p-group, i.e., $pa = \underbrace{a + \cdots + a}_{p} = 0$ where 0 is the null element of G. Define a relation R_G as follows:

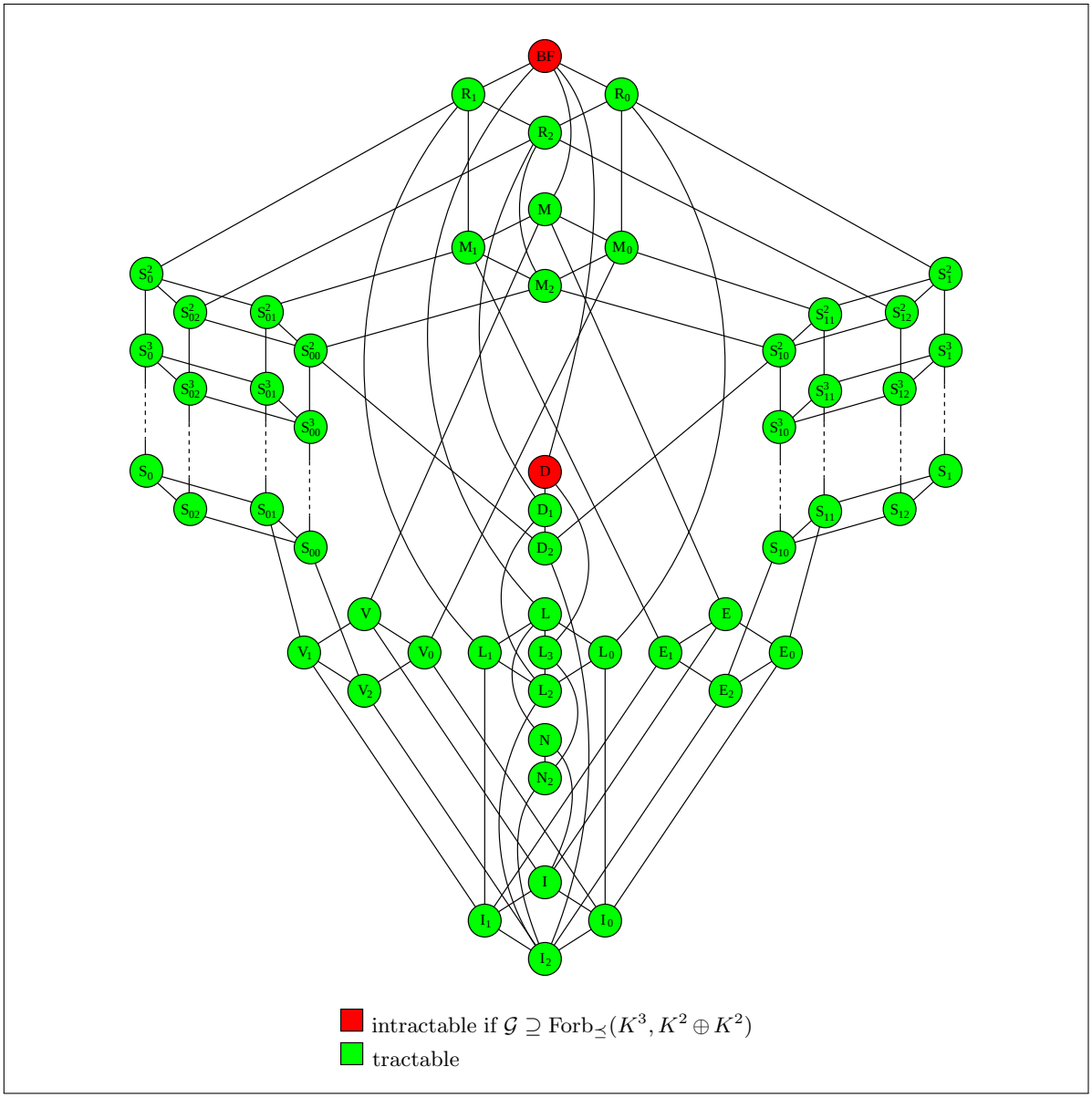


Figure 4.2. Islands of tractability for finding fixed points in boolean systems with succinctly represented local transition functions

$$R_G =_{\text{def}} \{(x, y, u, v) \mid x, y, u, v \in \mathcal{D} \text{ and } x + y = u + v\}.$$

Let $\mathbf{L}(\mathcal{D})$ denote the set of all relations R_G such that G is an abelian p-group on $\mathcal{D}$. If $\mathcal{D}$ is not a prime power then we set $\mathbf{L}(\mathcal{D}) = \emptyset$. Let R_G be a relation in $\mathbf{L}(\mathcal{D})$. Then, $\mathrm{Pol}(R)$ is the clone of all functions quasilinear with respect to G, i.e., an n-ary function f belongs to $\mathrm{Pol}(R)$ if and only if f satisfies for all $x_1, \ldots, x_n, y_1, \ldots, y_n \in \mathcal{D}$,

$$f(x_1 + y_1, \ldots, x_n + y_n) = f(x_1, \ldots, x_n) + f(y_1, \ldots, y_n) - f(0, \ldots, 0),$$

where addition is in G [135]. It has been proven in [137, 135] that for each $R \in \mathbf{L}(\mathcal{D})$, $\mathrm{Pol}(R)$ is a maximal clone. Fixed points of systems having quasilinear, local transition functions can be easily computed by Gaussian elimination.

Proposition 4.18 (Barrett et al. [15]). *Let $\mathcal{D}$ be a domain having a prime-power cardinality. Let R be any relation in $\mathbf{L}(\mathcal{D})$. Then,* Fixed Points$_{\mathrm{T}}(\mathrm{Pol}(R), \mathrm{Forb}_{\preceq}(\emptyset))$ *is solvable in polynomial time.*

In the boolean case the selfdual functions constitute an intractable fixed-point existence problem even in planar graphs. In larger domains we have a slightly more differentiated picture. Let $\mathcal{D}$ be any finite domain. Let $\pi : \mathcal{D} \to \mathcal{D}$ be a bijective mapping, i.e., π is a permutation. An n-ary function $f : \mathcal{D}^n \to \mathcal{D}$ is said to be *π-selfdual* if and only if for all $x_1, \ldots, x_n \in \mathcal{D}$,

$$\pi(f(x_1, \ldots, x_n)) = f(\pi(x_1), \ldots, \pi(x_n)).$$

Let $\mathrm{D}^\pi(\mathcal{D})$ be the class of all π-selfdual $\mathcal{D}$-functions. It is easily seen that for all permutations $\pi : \mathcal{D} \to \mathcal{D}$, the class $\mathrm{D}^\pi(\mathcal{D})$ is a clone. If the domain is clear from the context then we use D^π instead of $\mathrm{D}^\pi(\mathcal{D})$. The next theorem establishes a dichotomy for $\mathrm{D}^\pi(\mathcal{D})$ classes.

Theorem 4.19. *Let $\mathcal{D}$ be a finite domain and let $\pi : \mathcal{D} \to \mathcal{D}$ be any permutation. If there is exactly one $a \in \mathcal{D}$ such that $\pi(a) = a$ then* Fixed Points$_{\mathrm{T}}(\mathrm{D}^\pi, \mathrm{Forb}_{\preceq}(\emptyset))$ *is tractable. Otherwise,* Fixed Points$_{\mathrm{T}}(\mathrm{D}^\pi, \mathrm{Forb}_{\preceq}(K_{3,3}, K^5))$ *is intractable.*

Proof. We start with the tractable case. Suppose $a = \pi(a)$ for some $a \in \mathcal{D}$ and $b \neq \pi(b)$ for all $b \neq a$. If $f \in \mathrm{D}^\pi$ then $f(a, \ldots, a) = a$. This can be seen as follows:

$$\begin{aligned} b &= f(a, \ldots, a) \\ &= f(\pi(a), \ldots, \pi(a)) && (\text{since } a = \pi(a)) \\ &= \pi(f(a, \ldots, a)) && (\text{since } f \text{ is } \pi\text{-selfdual}) \\ &= \pi(b). \end{aligned}$$

Hence, $b = a$. Consequently, each dynamical system with local transition functions from D^π has a fixed point at the configuration where the state of each vertex is a. Thus, Fixed Points$_{\mathrm{T}}(\mathrm{D}^\pi, \mathrm{Forb}_{\preceq}(\emptyset))$ is solvable in polynomial time.

Now we turn to the intractable cases. Suppose that either there is no $a \in \mathcal{D}$ such that $a = \pi(a)$ or there are at least two different one's having this property. First, assume that $a \neq \pi(a)$ for all $a \in \mathcal{D}$. That is all cycles of the permutation π have length at least two. The proof of the NP-completeness is an easy extension of the proof of Theorem 4.13. We also reduce from Fixed Points$_{\mathrm{T}}(\mathrm{BF}, \mathrm{Forb}_{\preceq}(K_{3,3}, K^5))$. However, we simulate boolean functions by a π-selfdual function in a cycle of the permutation π which contains some fixed $b \in \mathcal{D}$. Define the following embedding of an m-ary $\mathcal{D}$-function into an $(m+n+1)$-ary π-selfdual $\mathcal{D}$-function:

$$\mathrm{sd}_n^\pi(f)(x_1, \ldots, x_m, y_1, \ldots, y_n, z) =_{\mathrm{def}} \begin{cases} \pi^{-j}(f(\pi^j(x_1), \ldots, \pi^j(x_m))) & \text{if } y_1 = \cdots = y_n = \pi^j(b) \\ \pi^{-1}(z) & \text{otherwise} \end{cases}$$

It is clear that for all $\mathcal{D}$-functions f, $\mathrm{sd}_n^\pi(f) \in \mathrm{D}^\pi$. Using the embedding sd_n^π the proof of Theorem 4.13 can be easily rewritten in order to prove the reduction from Fixed Points$_{\mathrm{T}}$(BF,

$\mathrm{Forb}_{\preceq}(K_{3,3}, K^5))$ to $\textsc{Fixed Points}_{\mathrm{T}}(\mathrm{D}^{\pi}, \mathrm{Forb}_{\preceq}(K_{3,3}, K^5))$. Now assume that there is a set $\{a_1, \ldots, a_r\} \subseteq \mathcal{D}$ such that $r \geq 2$, $b \notin \{a_1, \ldots, a_r\}$, and for all $1 \leq i \leq r$, $a_i = \pi(a_i)$. Suppose these are all elements having this property. In order to get the above simulation work in this case we have to exclude fixed points involving states from $\{a_1, \ldots, a_r\}$. Define the following auxiliary function $h : \mathcal{D} \to \mathcal{D}$ for $x \in \mathcal{D}$

$$h(x) =_{\mathrm{def}} \begin{cases} x & \text{if } x \notin \{a_1, \ldots, a_r\} \\ a_{i+1} & \text{if } x = a_i \text{ for } 1 \leq i \leq r_1 \\ a_1 & \text{if } x = a_r \end{cases}$$

Certainly, h belongs to D^{π}. Note that if $a \neq \pi(a)$ for all $a \in \mathcal{D}$ then h is the identity function. Replacing each variable x_i by $h(x_i)$ in all local transition functions of the system constructed as in the reduction of the first case we again obtain a polynomial-time many-one reduction from $\textsc{Fixed Points}_{\mathrm{T}}(\mathrm{BF}, \mathrm{Forb}_{\preceq}(K_{3,3}, K^5))$ to $\textsc{Fixed Points}_{\mathrm{T}}(\mathrm{D}^{\pi}, \mathrm{Forb}_{\preceq}(K_{3,3}, K^5))$. This proves the intractable cases. ❑

Let $\mathbf{P}(\mathcal{D})$ denote the set of all relations on $\mathcal{D}$ which are permutations π on $\mathcal{D}$ such that there is a prime number p such that all cycles have length p and p divides $\|\mathcal{D}\|$. Note that for each prime divisor of $\|\mathcal{D}\|$ there is such a permutation. If R is an relation in $\mathbf{P}(\mathcal{D})$ which corresponds to a permutation π_R then we have $\mathrm{Pol}(R) = \mathrm{D}^{\pi_R}$. For all $R \in \mathbf{P}(\mathcal{D})$, the clones $\mathrm{Pol}(R)$ are maximal [166].

Corollary 4.20. *Let $\mathcal{D}$ be any finite domain and let R be a relation in $\mathbf{P}(\mathcal{D})$. Then,* $\textsc{Fixed Points}(\mathrm{Pol}(R), \mathrm{Forb}_{\preceq}(K_{3,3}, K^5))$ *is* NP*-complete.*

For $\|\mathcal{D}\| = 3$ there is no relation $R \in \mathbf{P}(\mathcal{D})$ such that $\mathrm{Pol}(R)$ is not maximal and the fixed-point existence problem is intractable. This appears in the case that $\|\mathcal{D}\| \geq 4$, e.g., for each permutation with one cycle of length two and at least two cycle of length one.

As the proof of the tractable case in Theorem 4.19 clarifies, the π-selfdual functions for a permutation π with exactly one cycle of length one are b-reproducing for some $b \in \mathcal{D}$. For $\|\mathcal{D}\| \geq 3$ the b-reproducing $\mathcal{D}$-functions are not covered by the relations in $\mathbf{P}(\mathcal{D})$, yet they are a maximal clone for each b. The according relations belong to the set of central relations. Since a complexity classification of the fixed-point existence problem for the clones of central relations is not yet available, we only mention them and two more relation sets for the sake of completeness (see, e.g., [122, Chapter 4.3] for formal definitions):

- $\mathbf{C}(\mathcal{D})$ is the set of all non-diagonal central relations on $\mathcal{D}$,
- $\mathbf{E}(\mathcal{D})$ is the set of all non-diagonal equivalence relations on $\mathcal{D}$ without the coarsest and without the finest equivalence relation,
- $\mathbf{B}(\mathcal{D})$ is the set of all non-diagonal k-universal relations on $\mathcal{D}$ for $k \geq 3$.

The polymorphisms of the relations in these three and the three above-defined classes exhaust the set of maximal clones for each domain [135]. We notice that $\mathbf{E}(\{0,1\})$ and $\mathbf{B}(\{0,1\}$ are empty.

Table 4.1 which is taken from [122, Theorem 4.3.24] gives an impression on the growth of the number of maximal clones depending on the size of the domain. The other columns contain the number of different maximal clones which are polymorphisms of relations in $\mathbf{O}(\mathcal{D})$ (monotone clones), of relations in $\mathbf{L}(\mathcal{D})$ (quasilinear clones), and of relations in $\mathbf{P}(\mathcal{D})$ (selfdual clones).

Domain Size	Maximal Clones	Monotone Clones	Quasilinear Clones	Selfdual Clones
2	5	1	1	1
3	18	3	1	1
4	82	18	1	3
5	643	190	6	6
6	15,182	3,285	0	35
7	7,848,984	88,851	120	120

Table 4.1. The numbers of maximal clones.

4.4 The Fixed-Point Counting problem

4.4.1 Definition

In this section we are interested in the computational complexity of the following counting problem. Let $\mathcal{F}$ be a class of boolean functions and let $\mathcal{G}$ be a class of graphs.

Problem:	#Fixed Points$(\mathcal{F}, \mathcal{G})$
Input:	A dynamical system $S = (G, \{f_1, \ldots, f_n\})$ such that $G \in \mathcal{G}$ and for all $i \in \{1, \ldots, n\}$, $f_i \in \mathcal{F}$
Output:	The number of fixed points of S

Again we consider the three cases of lookup table, formula, and circuit representations of local transition functions. The corresponding problems are denoted by #Fixed Points$_{\mathrm{T}}$, #Fixed Points$_{\mathrm{F}}$, and #Fixed Points$_{\mathrm{C}}$. It is obvious that all problem versions belong to #P. We say that a problem is *intractable* if it is #P-complete and it is *tractable* if it is solvable in polynomial time.

4.4.2 Boolean systems with local transitions given by lookup tables

Again we are interested in boolean system. We start by identifying tractable counting problems.

Lemma 4.21. #Fixed Points$_{\mathrm{T}}(\mathrm{L}, \mathrm{Forb}_{\preceq}(\emptyset))$ *is solvable in polynomial time.*

Proof. Notice that for a linear function $f(x_1, \ldots, x_n) = a_0 \oplus a_1 x_2 \oplus a_2 x_2 \oplus \cdots \oplus a_n x_n$, the proposition $x_i \leftrightarrow [a_0 \oplus a_1 x_2 \oplus a_2 x_2 \oplus \cdots \oplus a_n x_n]$ is true if and only if $a_0 \oplus a_1 x_2 \oplus a_2 x_2 \oplus \cdots \oplus a_n x_n \oplus x_i \oplus 1$ is satisfiable. So, each dynamical system with linear, boolean local transition functions constitutes a system of linear equations over $\mathbb{Z}_2$, for which the number of solutions can be computed in polynomial time using Gaussian elimination (cf. [37]). ❑

Theorem 4.9 shows that the decision version of #Fixed Points$_{\mathrm{T}}(\mathrm{BF}, \mathrm{Forb}_{\preceq}(X))$ for planar graphs X can be solved in polynomial time. To obtain the result we constructed a reduction to constraint satisfaction problems. Actually, the reduction establishes injections between the fixed points of a dynamical system and the satisfying assignments of the corresponding constraint satisfaction problem. Consequently, the numbers of fixed points and the numbers of satisfying assignments are equal.

Lemma 4.22. *Let X be a planar graph. Then,* #Fixed Points$(\mathrm{BF}, \mathrm{Forb}_{\preceq}(X))$ *is solvable in polynomial time.*

Proof. By inspecting the proof of Theorem 4.9 and noting that counting satisfying assignments for constraint satisfaction problems having constraint graphs of bounded treewidth can be done in polynomial time (cf. [56]). ❑

We turn to intractable Fixed-Point Counting problems. Let H be a 2CNF such that each clause consists of exactly one positive and one negative literal. H is called a Horn-2CNF formula. Moreover, suppose H has a planar graph representation. Then, H is called a planar Horn 2-CNF formula. #Planar Horn-2SAT is the problem of counting all satisfying assignments of a given planar Horn-2CNF formula.

Proposition 4.23. #Planar Horn-2SAT *is* #P*-complete even if each variable is allowed to occur in four clauses only.*

Proof. The following problem has been shown to be #P-complete in [151]: #4Δ-Planar Bipartite Independent Set, i.e., compute, on a given bipartite graph $G = (V, E)$ with maximum vertex-degree at most four, the number of independent sets $U \subseteq V$. Let $G = (V, E)$ be a bipartite graph, $V = V_1 \cup V_2$ and $E \subseteq V_1 \times V_2$. Define H to be the 2CNF given by clauses $(x_u \vee \overline{x_v})$ for all $u \in V_1$, $v \in V_2$ such that $\{u, v\} \in E$. Clearly, H is a Horn-2CNF formula. Moreover, if G is planar and the maximum degree is at most four, the graph representation of H is planar and each variable occurs at most four times in H. Finally, it is easily seen that there is a bijection between the independent sets of G and the satisfying assignments for H (see, e.g., [126, 101]). Hence, #4Δ-Planar Bipartite Independent Set reduces to #Planar Horn-2SAT with each variable occuring in at most four clauses. ❑

Lemma 4.24. #Fixed Points$_{\mathrm{T}}(\mathrm{E}_2, \mathrm{Forb}_{\preceq}(K_{3,3}, K^5))$ *is* #P*-complete.*

Proof. We reduce from #Planar Horn-2SAT assuming that each variable occurs only four times in the formula. Let $H = C_1 \wedge \cdots \wedge C_m$ be a planar Horn-2CNF formula. Define a dynamical system $S = (G, F)$ as follows. Let $G = (V, E)$ be given by

$$\begin{aligned} V &=_{\mathrm{def}} \{1, \ldots, n\}, \\ E &=_{\mathrm{def}} \{\, \{i, j\} \mid (\overline{x_i} \vee x_j) = C_r \text{ for some } r \in \{1, \ldots, m\} \,\}. \end{aligned}$$

Since H has a planar graph representation, G is planar, i.e., $G \in \mathrm{Forb}_{\preceq}(K_{3,3}, K^5)$. The local transition functions are specified in the following way. For a vertex $i_0 \in V$ let $\{i_1, \ldots, i_r\}$ be the set of all vertices such that $(\overline{x_{i_j}} \vee x_{i_0})$ is a clause in H. Then, f_{i_0} is the function given by the formula

$$H_{i_0} = x_{i_0} \wedge x_{i_1} \wedge \cdots \wedge x_{i_r}.$$

Notice that all local transition functions belong to E_2 and also notice that the maximum degree of a vertex in G is four. Thus, we can compute the lookup tables in polynomial time depending on the size of H. Moreover, it is easily seen that

$$x_{i_0} \leftrightarrow \bigwedge_{j=1}^{r} x_{i_j} \;\equiv\; \bigwedge_{j=1}^{r} (\overline{x_{i_j}} \vee x_{i_0}).$$

Hence, the number of satisfying assignments of H is equal to the number of fixed-point configurations of S_H. This shows that #Planar Horn-2SAT reduces to #Fixed Points$_{\mathrm{T}}$(E_2, $\mathrm{Forb}_{\preceq}(K_{3,3}, K^5)$). ❑

Lemma 4.25. #Fixed Points$_{\mathrm{T}}$($\mathrm{V}_2, \mathrm{Forb}_{\preceq}(K_{3,3}, K^5)$) *is* #P*-complete.*

Proof. Again we reduce from #Planar Horn-2SAT assuming that each variable occurs only four times in the formula. Let $H = C_1 \wedge \cdots \wedge C_m$ be a planar Horn-2CNF formula. We construct the same network as in the proof of Lemma 4.24 on a given planar Horn-2CNF formula H having the variables $x_1, \ldots, x_n$. However, the local transition functions are specified as follows. For a vertex $i_0 \in V$, let $\{i_1, \ldots, i_r\}$ be the set of all vertices such that $(\overline{x_{i_0}} \vee x_{i_j})$ is a clause in H. Then, f_{i_0} is the function given by the formula

$$H_{i_0} =_{\mathrm{def}} x_{i_0} \vee x_{i_1} \vee \cdots \vee x_{i_r}$$

which clearly belongs to V_2. It remains to verify the number of satisfying assignments of H equals the number of fixed-point configuration of S_H. This follows from

$$x_{i_0} \leftrightarrow \bigvee_{j=1}^{r} x_{i_j} \;\equiv\; \bigwedge_{j=1}^{r} (\overline{x_{i_0}} \vee x_{i_j}).$$

So, #Planar Horn-2SAT reduces to #Fixed Points$_{\mathrm{T}}$($\mathrm{V}_2, \mathrm{Forb}_{\preceq}(K_{3,3}, K^5)$). ❑

Proposition 4.26. #Fixed Points$_{\mathrm{T}}$($\mathrm{D}_1, \mathrm{Forb}_{\preceq}(K_{3,3}, K^5)$) *is* #P*-complete.*

Proof. We reduce from the #Planar Horn-2SAT version where each variable occurs only up to four times. Let $H = C_1 \wedge \cdots \wedge C_m$ be a planar Horn 2CNF formula having variables $x_1, \ldots, x_n$. We define a dynamical system $S_H = (G, \{f_1, \ldots, f_n\})$ as follows. The network $G = (V, E)$ is given by

$$\begin{aligned} V &=_{\mathrm{def}} \{1, \ldots, n, n+1, \ldots n+m\} \\ E &=_{\mathrm{def}} \{\, \{i, j\,\} \mid (i \leq n) \text{ and } (j > n) \text{ and } (x_i \text{ is a variable in clause } C_{j-n})\}. \end{aligned}$$

That is, G is isomorphic to the graph representation of H. Hence, G is a planar graph. Moreover, the maximum degree of G is at most four. First, assume that G is connected. The local transition functions are specified as follows. For each $i \in \{1, \ldots, n\}$, let $\{j_1, \ldots, j_r\} = N_G(i)$. Define

$$f_i(x_i, x_{j_1}, \ldots, x_{j_r}) =_{\mathrm{def}} \begin{cases} x_i & \text{if } x_{j_1} = \cdots = x_{j_r} \\ \mathrm{not}(x_i) & \text{otherwise} \end{cases}$$

It is easily seen that for each $i \in \{1, \ldots, n\}$, $f_i \in \mathrm{D}_1$. Note that, since G is connected and because each x_{j_ℓ} is an argument to exactly two of these functions, the above functions require that all variables x_j for $j \in \{n+1, \ldots, n+m\}$, i.e., all clauses, be equal in order to have a fixed point. For each $i \in \{n+1, \ldots, n+m\}$, let (without loss of generality) $C_i = (\overline{x_{i_1}} \vee x_{i_2})$. Define a formula

$$H_i(x_i, x_{i_1}, x_{i_2}) =_{\mathrm{def}} \begin{cases} \overline{x_{i_1}} \wedge x_{i_2} & \text{if } x_i = 0 \\ \overline{x_{i_1}} \vee x_{i_2} & \text{if } x_i = 1 \end{cases}$$

Let f_i be the function represented by H_i. Truth-table inspection shows that for each $i \in \{n+1, \ldots, n+m\}$, $f_i \in \mathrm{D}_1$. Note that the lookup tables for the local transition functions

have at most 2^5 entries. Also, it is easy to check that for every satisfying assignment of H there are exactly two fixed points in S_H, i.e, the number of fixed-point configurations of S_H is equal to $2 \cdot \#_+(H)$, where $\#_+(H)$ is the number of satisfying assignments of H. To see this, note that a configuration $\boldsymbol{x}$ is a fixed point exactly when $x_{n+1} = \cdots = x_{n+m} = 1$ and the values $x_1, \ldots, x_n$ are a satisfying assignment to H, or when $x_{n+1} = \cdots = x_{n+m} = 0$ and $x_1, \ldots, x_n$ are a satisfying assignment to H', the formula that is the same as H except that each literal is negated (so, e.g., literals with negations in H no longer have negations in H'). Now, assume that G is not connected. Then, the arguments apply independently to each connected component of G, so the total number of fixed points becomes $2^s \cdot \#_+(H)$, where s is the number of connected components G has. All in all, this shows that #Planar Horn-2SAT reduces to #Fixed Points$_{\mathrm{T}}(\mathrm{D}_1, \mathrm{Forb}_{\preceq}(K_{3,3}, K^5))$. ❑

The remaining case of D_2 functions is special. We conjecture that the fixed-point counting problem for dynamical systems with local transition functions in D_2 and planar networks is intractable. This is based on the following weaker proposition.

Proposition 4.27. #Fixed Points$_{\mathrm{T}}(\mathrm{D}_2, \mathrm{Forb}_{\preceq}(\emptyset))$ *is* #P*-complete.*

Proof. We reduce from the #Planar Horn-2SAT version where each variable occurs only up to four times. Let $H = C_1 \wedge \cdots \wedge C_m$ be a planar Horn 2CNF formula having variables $x_1, \ldots, x_n$. We define a dynamical system $S_H = (G, \{f_1, \ldots, f_n\})$ as follows. The network $G = (V, E)$ is given by

$$\begin{aligned} V &=_{\mathrm{def}} \{1, \ldots, n, n+1, \ldots, 2n\} \\ E &=_{\mathrm{def}} \{\ \{i, j\ \} \mid (\overline{x_i} \vee x_j) \text{ or } (x_i \vee \overline{x_j}) \text{ is a clause of } H\ \} \\ &\qquad \cup\ \{\ \{i, i+n\} \mid i \in \{1, \ldots, n\}\ \}\ \cup\ \{\ \{j, j+1\} \mid j \in \{n, \ldots, 2n-1\} \end{aligned}$$

The local transition functions are specified as follows. Set

$$\begin{aligned} f_{2n}(x_n, x_{2n-1}, x_{2n}) &=_{\mathrm{def}} x_{2n}, \\ f_{n+1}(x_1, x_{n+1}, x_{n+2}) &=_{\mathrm{def}} x_{n+2}, \end{aligned}$$

and for $n+1 < i < 2n$, define

$$f_i(x_{i-n}, x_{i-1}, x_i, x_{i+1}) =_{\mathrm{def}} x_{i+1}.$$

For $i \in U$, let $\{0, i_1, \ldots, i_r\} = N_G(i) \cup \{i\}$ and let J_i denote the set of all vertices $j \in N_G(i) \cap \{1, \ldots, n\}$ such that $(\overline{x_i} \vee x_j)$ is a clause in H. We define a formula

$$H_i(x_{i_1}, \ldots, x_{i_r}, x_{i+n}) =_{\mathrm{def}} \begin{cases} x_i \wedge \bigwedge_{j \in J_i} x_j & \text{if } x_{i+n} = 0 \\ x_i \vee \bigvee_{j \in J_i} x_j & \text{if } x_{i+n} = 1 \end{cases}$$

Let f_i be the boolean function represented by H_i. It is easily seen that for all $i \in V$, we have $f_i \in \mathrm{D}_2$. Note that the lookup tables for the local transition functions have at most 2^6 entries. Thus, S_H is computable in time polynomial in the size of H. Moreover, a simple analysis shows that the number of fixed-point configurations of S_H is equal to $2 \cdot \#_+(H)$, where $\#_+(H)$ is the number of satisfying assignments of H. For this to obtain observe that any fixed-configuration $\boldsymbol{x}$ satisfies $x_{n+1} = \cdots = x_n$. Thus, #Planar Horn-2SAT reduces to #Fixed Points$_{\mathrm{T}}(\mathrm{D}_2, \mathrm{Forb}_{\preceq}(\emptyset))$. ❑

Finally, we combine the results to obtain the following conditional dichotomy theorem.

Theorem 4.28. *Let $\mathcal{F}$ be a boolean clone and let $\mathcal{G}$ be a graph class closed under taking minors. Under the assumption that #Fixed Points$(\mathrm{D}_2, \mathrm{Forb}_{\preceq}(K_{3,3}, K^5))$ is an intractable problem, the following holds: If ($\mathcal{F} \supseteq \mathrm{V}_2$ or $\mathcal{F} \supseteq \mathrm{E}_2$ or $\mathcal{F} \supseteq \mathrm{D}_2$) and $\mathcal{G} \supseteq \mathrm{Forb}_{\preceq}(K_{3,3}, K^5)$ then #Fixed Points$_{\mathrm{T}}(\mathcal{F}, \mathcal{G})$ is intractable, otherwise #Fixed Points$_{\mathrm{T}}(\mathcal{F}, \mathcal{G})$ is tractable.*

Proof. If ($\mathcal{F} \supseteq \mathrm{V}_2$ or $\mathcal{F} \supseteq \mathrm{E}_2$ or $\mathcal{F} \supseteq \mathrm{D}_2$) and $\mathcal{G} \supseteq \mathrm{Forb}_{\preceq}(K_{3,3}, K^5)$ then #Fixed Points$_{\mathrm{T}}$ $(\mathcal{F}, \mathcal{G})$ is #P-complete by Lemma 4.24, Lemma 4.25, and by the assumption made for D_2. Suppose the premise is not satisfied. First, assume that $\mathcal{F} \not\subseteq \mathrm{V}_2$, $\mathcal{F} \not\subseteq \mathrm{E}_2$, and $\mathcal{F} \not\subseteq \mathrm{D}_2$. The maximal clone having this property is L. By Lemma 4.21, #Fixed Points$_{\mathrm{T}}(\mathrm{L}, \mathrm{Forb}_{\preceq}(\emptyset))$ is tractable. It remains to consider the case $\mathcal{G} \not\supseteq \mathrm{Forb}_{\preceq}(K_{3,3}, K^5)$. That is $\mathcal{G} \subseteq \mathrm{Forb}_{\preceq}(X)$ for some planar graph X. Lemma 4.22 shows that #Fixed Points$_{\mathrm{T}}(\mathrm{BF}, \mathcal{G})$ is solvable in polynomial time. ❑

Figure 4.3 shows the islands of tractability for fixed-point counting according to Theorem 4.28. Note that the intractability for D_2 is only conjectured.

4.4.3 Boolean systems with succinctly represented local transitions

In this section we prove a dichotomy theorem for the fixed-point counting problem when transition are given by formulas or circuits. As for the decision problem both succinct representations of functions lead to the same result. We only prove special results for the case of formula representations. The corresponding results for circuit representations follow easily.

Again we start with gathering the tractable cases.

Lemma 4.29. #Fixed Points$_{\mathrm{F}}(\mathrm{L}, \mathrm{Forb}_{\preceq}(\emptyset))$ *is solvable in polynomial time.*

Proof. Similar to the proof of Lemma 4.21 by noting that each boolean circuit C over the base $\{\oplus, 1, 0\}$ can be easily transformed (in polynomial time in the number of gates of C) into the described system of linear equations over $\mathbb{Z}_2$. ❑

Lemma 4.30. *Let X be a planar graph. Then,* #Fixed Points$_{\mathrm{F}}(\mathrm{E}, \mathrm{Forb}_{\preceq}(X))$ *is solvable in polynomial time.*

Proof. Since X is planar, there exists a $k \in \mathbb{N}$ such that for all $G \in \mathrm{Forb}_{\preceq}(X)$, the treewidth of G is at most k. Let $S = (G, \{f_1, \ldots, f_n\})$ be a dynamical system such that $G = (V, E) \in \mathrm{Forb}_{\preceq}(X)$ and for all $i \in V$, the local transition function f_i is one of the constant functions c_0 or c_1, or is represented by a formula $H_i = \bigwedge_{j \in J_i} x_j$, where $J_i \subseteq N_G(i) \cup \{i\}$. Without loss of generality, we may assume that there is no $i \in V$ such that f_i is a constant function. (Otherwise, an obvious procedure exists to eliminate such vertices.) We define the directed graph $A(S)$ to consist of S's vertex set V and the edge set

$$E' =_{\mathrm{def}} \{\ (i, j) \mid i, j \in V, i \in J_j\ \}.$$

Note that $A(S)$ is allowed to have loops. Observe that for all vertices $i, j \in V$ and all fixed-point configurations $\boldsymbol{x}$ it holds that if $x_i = 0$ then $x_j = 0$. An easy consequence is that if $C = \{i_1, \ldots, i_r\}$ is a strongly connected component of $A(S)$ and $\boldsymbol{x}$ is a fixed-point configuration then $x_{i_1} = \cdots = x_{i_r}$. Let $\{C_1, \ldots, C_\ell\}$ be the set of all strongly connected

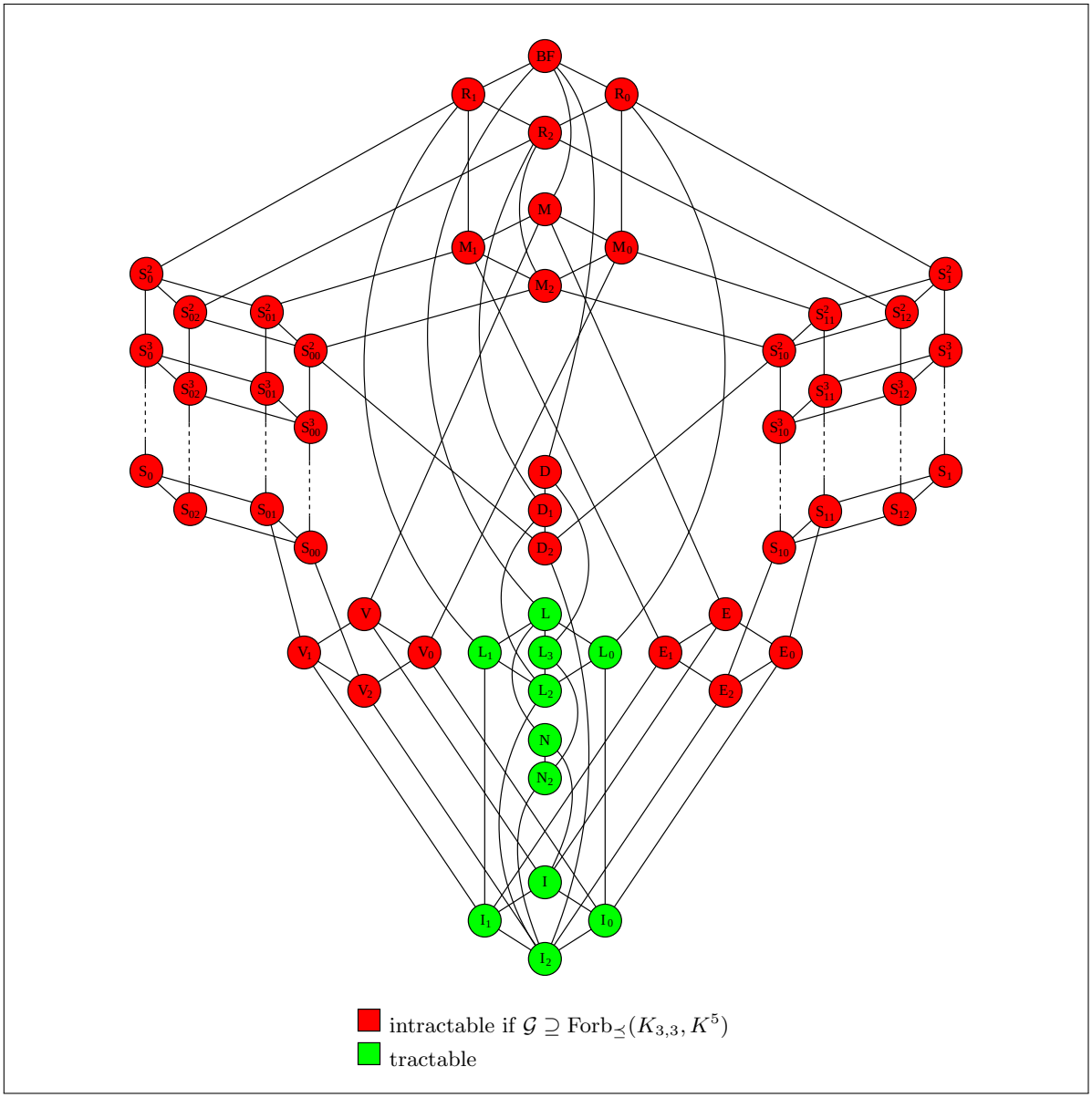


Figure 4.3. Islands of tractability for counting fixed points in boolean systems with local transition functions given by lookup tables

components of $A(S)$. Then, the number of fixed-point configurations of S is equal to the number of satisfying assignments of the constraint satisfaction problem $\mathrm{CSP}(S) = (W, \mathcal{D}, \mathcal{C})$ defined by:

$$
\begin{aligned}
W &=_{\text{def}} \{x_1, \ldots, x_\ell\} \\
\mathcal{D} &=_{\text{def}} \{0, 1\} \\
\mathcal{C} &=_{\text{def}} \{\ Ex_ix_j \mid \text{ there are } u \in C_i \text{ and } v \in C_j \text{ such that } (u, v) \in E'\ \} \\
&\qquad \text{where for all } i, j \text{ such that } Ex_ix_j \in \mathcal{C},\ E_{ij} =_{\text{def}} \{\ (0,0), (1,0), (1,1)\ \}
\end{aligned}
$$

Note that the constraint graph of $\mathrm{CSP}(S)$ (up to being oriented) is a minor of the network of S. It follows that the constraint graph has treewidth at most k. Hence, using the algorithms in [56], the number of fixed-point configurations can be computed in polynomial time. Consequently, $\#\textsc{Fixed Points}_{\mathrm{F}}(\mathrm{E}, \mathrm{Forb}_{\preceq}(X))$ can be solved in polynomial time. ❑

Lemma 4.31. *Let X be a planar graph. Then, $\#\textsc{Fixed Points}_{\mathrm{F}}(\mathrm{V}, \mathrm{Forb}_{\preceq}(X))$ is solvable in polynomial time.*

Proof. The case of V is dual to the case of E. Indeed, suppose we have a dynamical system $S = (G, \{f_1, \ldots, f_n\})$ such that $G = (V, E) \in \mathrm{Forb}_{\preceq}(X)$ and for all $i \in V$, f_i is constant or represented by a formula $H_i = \bigvee_{j \in J_i} x_j$ where $J_i \subseteq N_G(i) \cup \{i\}$. Replace each $\vee$ by $\wedge$, 0 by 1, and 1 by 0. Obviously, this gives a dynamical system having the same number of fixed-point configurations as S. Thus, $\#\textsc{Fixed Points}_{\mathrm{F}}(\mathrm{V}, \mathrm{Forb}_{\preceq}(X))$ reduces to $\#\textsc{Fixed Points}_{\mathrm{F}}(\mathrm{E}, \mathrm{Forb}_{\preceq}(X))$. Hence, by Lemma 4.30, $\#\textsc{Fixed Points}_{\mathrm{F}}(\mathrm{V}, \mathrm{Forb}_{\preceq}(X))$ can be solved in polynomial time. ❑

Lemma 4.32. *Let X be an arbitrary graph having a vertex cover of size one. Then, $\#\textsc{Fixed Points}_{\mathrm{F}}(\mathrm{BF}, \mathrm{Forb}_{\preceq}(X))$ is solvable in polynomial time.*

Proof. Let X have a vertex cover of size one, i.e., $\mathrm{Forb}_{\preceq}(X)$ has bounded degree. So, it is easily seen that for all classes $\mathcal{F}$ of boolean functions, $\#\textsc{Fixed Points}_{\mathrm{F}}(\mathcal{F}, \mathrm{Forb}_{\preceq}(X))$ reduces to $\#\textsc{Fixed Points}_{\mathrm{T}}(\mathcal{F}, \mathrm{Forb}_{\preceq}(X))$. As X is also a planar graph (note that $K^3 \preceq K_{3,3}$ and $K^2 \oplus K^2 \preceq K_{3,3}$ as well as $K^3 \preceq K^5$ and $K^2 \oplus K^2 \preceq K^5$), $\#\textsc{Fixed Points}_{\mathrm{F}}(\mathrm{BF}, \mathrm{Forb}_{\preceq}(X))$ is solvable in polynomial time using Lemma 4.22. ❑

We turn to the #P-complete cases.

Lemma 4.33. $\#\textsc{Fixed Points}_{\mathrm{F}}(\mathrm{E}_2, \mathrm{Forb}_{\preceq}(K_{3,3}, K^5))$ *is* #P*-complete.*

Proof. An inspection of the proof of Lemma 4.24 shows that the local transition functions specified there are in fact, represented by formulas. Thus, the proposition follows from the proof of Lemma 4.24. ❑

Lemma 4.34. $\#\textsc{Fixed Points}_{\mathrm{F}}(\mathrm{V}_2, \mathrm{Forb}_{\preceq}(K_{3,3}, K^5))$ *is* #P*-complete.*

Proof. Similar to Lemma 4.33 by inspecting the proof of Lemma 4.25. ❑

Let H be a 2CNF formula such that each clause consists of positive literals only. H is called a positive 2CNF. It is well known that the counting problem #Pos 2SAT, i.e., counting the satisfying assignments of positive 2CNF, is #P-complete [153].

Lemma 4.35. $\#\textsc{Fixed Points}_{\mathrm{F}}(\mathrm{S}_{10}, \mathrm{Forb}_{\preceq}(K^3, K^2 \oplus K^2))$ *is* #P*-complete.*

Proof. We reduce from #Pos 2SAT. Let $H = C_1 \wedge \cdots \wedge C_m$ be a positive 2CNF formula having variables $x_1, \ldots, x_n$. Let $\#_+(H)$ denote the number of satisfying assignments of H. Let

$$S_{10}(x, y, z) =_{\mathrm{def}} (x \wedge (y \vee z))$$

denote the only element in the logical basis of S_{10}. Define S_H to be the dynamical system consisting of the network $G = (V, E)$ given by

$$\begin{aligned} V &=_{\text{def}} \{1, \ldots, n, n+1\} \\ E &=_{\text{def}} \{\ \{i, n+1\} \mid i \in \{1, \ldots, n\}\ \}, \end{aligned}$$

and the local transition functions are specified as follows. For $i \in \{1, \ldots, n\}$ set

$$P_i(x_i, x_{n+1}) =_{\text{def}} S_{10}(x_i, x_i, x_i)$$

and let f_i be represented by P_i. For $i = n+1$, we first define auxiliary formulas A_j for $j \in \{1, \ldots, m\}$ by

$$A_1(x_1, \ldots, x_{n+1}) =_{\text{def}} S_{10}(x_{n+1}, x_{11}, x_{12})$$

and for $k > 1$ by

$$A_k(x_1, \ldots, x_{n+1}) =_{\text{def}} S_{10}(A_{k-1}(x_1, \ldots, x_{n+1}), x_{k1}, x_{k2})$$

where $C_k = (x_{k1} \vee x_{k2})$. Finally, set

$$P_{n+1}(x_1, \ldots, x_{n+1}) =_{\text{def}} A_m(x_1, \ldots, x_{n+1})$$

and let f_{n+1} be represented by P_{n+1}. Certainly, S_H is a dynamical system with local transition functions in S_{10} and an underlying network in $\mathrm{Forb}_{\preceq}(K^3 \oplus K^2))$ which is computable in time polynomial in the size of H. Moreover, note that

$$P_{n+1}(x_1, \ldots, x_{n+1}) \equiv x_{n+1} \wedge \bigwedge_{j=1}^{m} C_j.$$

It follows that the number of fixed-point configurations of S_H is $\#_+(H) + 2^n$. Hence, #Pos 2SAT reduces to #Fixed $\textsc{Points}_{\mathrm{F}}(\mathrm{S}_{10}, \mathrm{Forb}_{\preceq}(K^3, K^2 \oplus K^2))$. ❑

Lemma 4.36. #Fixed $\textsc{Points}_{\mathrm{F}}(\mathrm{S}_{00}, \mathrm{Forb}_{\preceq}(K^3, K^2 \oplus K^2))$ *is* #P*-complete.*

Proof. Again we reduce from #Pos 2SAT. Let $H = C_1 \wedge \cdots \wedge C_m$ be a positive 2CNF formula having variables $x_1, \ldots, x_n$. Let $\#_+(H)$ denote the number of satisfying assignments of H. Let

$$S_{00}(x, y, z) =_{\text{def}} (x \vee (y \wedge z))$$

denote the only element in the logical basis of S_{00}. We define S_H to be the dynamical system consisting of the network $G = (V, E)$, where

$$\begin{aligned} V &=_{\text{def}} \{0, 1, \ldots, n, n+1\}, \\ E &=_{\text{def}} \{\ \{i, n+1\} \mid i \in \{0, \ldots, n\}\ \}, \end{aligned}$$

and the set of local transition functions specified as follows: for $i \in \{0, \ldots, n\}$, set

$$P_i(x_i, x_{n+1}) =_{\text{def}} S_{00}(x_i, x_i, x_i)$$

and let f_i be the function represented by P_i. For $i = n+1$, i.e., the center of the star G, we first introduce auxiliary formulas $A_{j_1,\ldots,j_k}(x_0, x_1, \ldots, x_n)$ for $k \in \mathbb{N}_+$ and $j_1 < \cdots < j_k$ inductively defined by

$$A_i(x_0, x_1, \ldots, x_n) =_{\text{def}} S_{00}(x_{i_1}, x_{i_2}, x_{i_2}),$$

such that $C_i = (x_{i_1} \vee x_{i_2})$, and

$$A_{j_1,\ldots,j_k}(x_0, \ldots, x_n) =_{\text{def}} S_{00}(x_0, A_{j_1,\ldots,j_{\lfloor k/2 \rfloor}}(x_0, \ldots, x_n), A_{j_{\lfloor k/2 \rfloor+1},\ldots,j_k}(x_0, \ldots, x_n)).$$

We finally define

$$P_{n+1}(x_0, \ldots, x_{n+1}) =_{\text{def}} S_{00}(x_0, A_{1,\ldots,m}(x_0, \ldots, x_n), x_{n+1}).$$

Clearly, S_H is a dynamical system with local transition functions in S_{00} and a network in $\text{Forb}_{\preceq}(K^3, K^2 \oplus K^2))$ which is computable in time polynomial in the size of H. Moreover, as is easily seen by induction it holds that

$$A_{j_1,\ldots,j_k}(c_0, x_1, \ldots, x_n) \equiv \bigwedge_{\ell=1}^{k} C_\ell \quad \text{and} \quad A_{j_1,\ldots,j_k}(c_1, x_1, \ldots, x_n) \equiv c_1.$$

This leads to the following numbers of fixed-point configurations of S_H:

– there are 2^n fixed-point configurations $\boldsymbol{x}$ such that $x_0 = 0$ and $x_{n+1} = 0$,
– there are $\#_+(H)$ fixed-point configurations $\boldsymbol{x}$ such that $x_0 = 0$ and $x_{n+1} = 1$,
– there is no fixed-point configuration $\boldsymbol{x}$ such that $x_0 = 1$ and $x_{n+1} = 0$, and
– there are 2^n fixed-point configurations $\boldsymbol{x}$ such that $x_0 = 1$ and $x_{n+1} = 1$.

Hence, the number of fixed-point configurations of S_H is just $\#_+(H) + 2^{n+1}$. Consequently, #Pos 2SAT reduces to #Fixed $\text{Points}_{\text{F}}(\text{S}_{00}, \text{Forb}_{\preceq}(K^3, K^2 \oplus K^2))$. ❑

Lemma 4.37. #Fixed $\text{Points}_{\text{F}}(\text{D}_2, \text{Forb}_{\preceq}(K^3, K^2 \oplus K^2))$ *is* #P*-complete.*

Proof. We reduce from #Pos 2SAT. We construct the same network as for the case S_{00} in the proof of Lemma 4.36 on a given positive 2CNF formula $H = C_1 \wedge \cdots \wedge C_m$ having variables $x_1, \ldots, x_n$. Let $\#_+(H)$ denote the number of satisfying assignments of H. The local transition functions are specified as follows. Let

$$D_2(x, y, z) =_{\text{def}} (x \wedge y) \vee (x \wedge z) \vee (y \wedge z)$$

denote the only element in the logical basis of D_2. For $i \in \{0, \ldots, n\}$ set

$$P_i(x_i, x_{n+1}) =_{\text{def}} D_2(x_i, x_i, x_i)$$

and let f_i be represented by P_i. For $i = n + 1$, we again introduce auxiliary formulas $A_{j_1,\ldots,j_k}(x_0, x_1, \ldots, x_n, x_{n+1})$ for $k \in \mathbb{N}_+$ and $j_1 < \cdots < j_k$ inductively defined by

$$A_i(x_0, \ldots, x_{n+1}) =_{\text{def}} D_2(x_{i_1}, x_{i_2}, x_{n+1}),$$

such that $C_i = (x_{i_1} \vee x_{i_2})$, and

$$A_{j_1,\ldots,j_k}(x_0, \ldots, x_{n+1}) =_{\text{def}} D_2(A_{j_1,\ldots,j_{\lfloor k/2 \rfloor}}(x_0, \ldots, x_{n+1}), A_{j_{\lfloor k/2 \rfloor+1},\ldots,j_k}(x_0, \ldots, x_{n+1}), x_0).$$

We finally define

$$P_{n+1} =_{\text{def}} A_{1,\ldots,m}.$$

Evidently, S_H is a dynamical system with local transition functions in D_2 and an underlying network in $\mathrm{Forb}_{\preceq}(K^3, K^2 \oplus K^2))$ which can be computed in time polynomial in the size of H. Moreover, by induction over the formula structure of P_{n+1} we easily obtain the following equivalences:

$$\begin{aligned} P_{n+1}(0, x_1, \ldots, x_n, 0) &\equiv \textstyle\bigwedge_{i=1}^{n} x_i \\ P_{n+1}(0, x_1, \ldots, x_n, 1) &\equiv \textstyle\bigwedge_{i=1}^{m} (x_{i1} \vee x_{i2}) \\ P_{n+1}(1, x_1, \ldots, x_n, 0) &\equiv \textstyle\bigvee_{i=1}^{m} (x_{i1} \wedge x_{i2}) \\ P_{n+1}(1, x_1, \ldots, x_n, 1) &\equiv \textstyle\bigvee_{i=1}^{n} x_i \end{aligned}$$

Thus, the number of fixed-point configurations of S_H is exactly $2\#_+(H) + 2^{n+1} - 2$. Hence, #Pos 2SAT reduces to #Fixed Points$_{\mathrm{F}}(\mathrm{D}_2, \mathrm{Forb}_{\preceq}(K^3, K^2 \oplus K^2))$. ❑

Finally, we combine all results to obtain the following dichotomy theorem.

Theorem 4.38. *Let $\mathcal{F}$ be a boolean clone and let $\mathcal{G}$ be a graph class closed under taking minors. Then, #Fixed Points$_{\mathrm{F}}(\mathcal{F}, \mathcal{G}))$ is intractable if one of the following conditions is satisfied.*

1. $\big(\mathcal{F} \supseteq \mathrm{S}_{00}$ *or* $\mathcal{F} \supseteq \mathrm{S}_{10}$ *or* $\mathcal{F} \supseteq \mathrm{D}_2\big)$ *and* $\mathcal{G} \supseteq \mathrm{Forb}_{\preceq}(K^3, K^2 \oplus K^2)$.
2. $\big(\mathcal{F} \supseteq \mathrm{V}_2$ *or* $\mathcal{F} \supseteq \mathrm{E}_2\big)$ *and* $\mathcal{G} \supseteq \mathrm{Forb}_{\preceq}(K_{3,3}, K^5)$.

Otherwise, #Fixed Points$_{\mathrm{F}}(\mathcal{F}, \mathcal{G})$ is tractable. Moreover, the same classification is true for #Fixed Points$_{\mathrm{C}}(\mathcal{F}, \mathcal{G})$.

Proof. If for $\mathcal{F}$ and $\mathcal{G}$ the first conditions is satisfied then the intractability follows from Lemmas 4.35, 4.36, and 4.37. If $\mathcal{G} \not\supseteq \mathrm{Forb}_{\preceq}(K^3, K^2 \oplus K^2)$ then there is a graph X having a vertex cover of size one such that $\mathcal{G} \in \mathrm{Forb}_{\preceq}(X)$. Lemma 4.32 shows that #Fixed Points$_{\mathrm{F}}(\mathrm{BF}, \mathrm{Forb}_{\preceq}(\mathcal{G}))$ is solvable in polynomial time. Assume that $\mathcal{F} \not\supseteq \mathrm{S}_{00}$, $\mathcal{F} \not\supseteq \mathrm{S}_{10}$, and $\mathcal{F} \not\supseteq \mathrm{D}_2$. The maximal classes satisfying this are V, E, and L. Thus, we only consider subclasses of these three classes. If $\mathcal{F}$ and $\mathcal{G}$ satisfy the second condition then the Lemmas 4.33 and 4.34 establish the intractability. Suppose the second condition does not hold. The maximal class $\mathcal{F}$ such that $\mathcal{F} \not\supseteq \mathrm{V}_2$ and $\mathcal{F} \not\supseteq \mathrm{E}_2$ is L. Lemma 4.29 states that for L counting the number of fixed points can be done in polynomial time. If $\mathcal{G} \not\supseteq \mathrm{Forb}_{\preceq}(K_{3,3}, K^5)$ then we know that $\mathcal{G} \in \mathrm{Forb}_{\preceq}(X)$ for some planar graph X. Lemmas 4.30 and 4.31 imply that #Fixed Points$_{\mathrm{F}}(\mathrm{E}, \mathcal{G})$ and #Fixed Points$_{\mathrm{F}}(\mathrm{V}, \mathcal{G})$ are solvable in polynomial time. ❑

Figure 4.4 shows the islands of tractability for the Fixed-Point Counting problem when transition functions are represented by formula or circuits.

4.5 Summary

In this chapter we characterized the islands of tractability of two computational analysis problem related to fixed-point configurations in dynamical systems.

For the Fixed-Point Existence problem for boolean dynamical systems a complete complexity classification, with respect to transition classes $\mathcal{F}$ closed under superposition and network classes $\mathcal{G}$ closed under taking minors, were presented: If $\mathcal{F}$ contains the self-dual

functions and $\mathcal{G}$ contains the planar graphs, then $\textsc{Fixed Points}_{\mathrm{T}}(\mathcal{F}, \mathcal{G})$ is intractable, otherwise it is tractable. Replacing "planar graphs" by "graphs having a vertex cover of size one" yields the same dichotomy theorem for the succinct representations of local transition functions by formulas or circuits. The linear and monotone functions have been shown to be tractable cases in [15]. There, the authors suggested to find more tractable classes of local transition functions. Over the boolean domain our results show that, aside from two obvious exceptions (the 0- and 1-reproducing functions), there are no more such function classes. Also, the partial results for larger domains unearthed no essentially different clones as candidates for tractability.

For the Fixed-Point Counting problem we presented two dichotomy theorems (with one relying on the intractability conjecture for fixed-point counting in systems with local transition functions from D_2 over planar graphs). Both results demonstrate that the linear boolean functions are the only function class such that counting fixed points is tractable independent of representations and of degrees of variable dependency.

The main open question from this chapter is certainly proving a dichotomy theorem for the Fixed-Point Existence problem over arbitrary finite domains. Furthermore, in order to examine its usefulness the dynamical systems framework should be applied to more schedule-based analysis problems, such as Fixed-Point Reachability or Permutation Existence problems (see, e.g., [70, 141, 12, 13, 14] for existing studies in these directions).

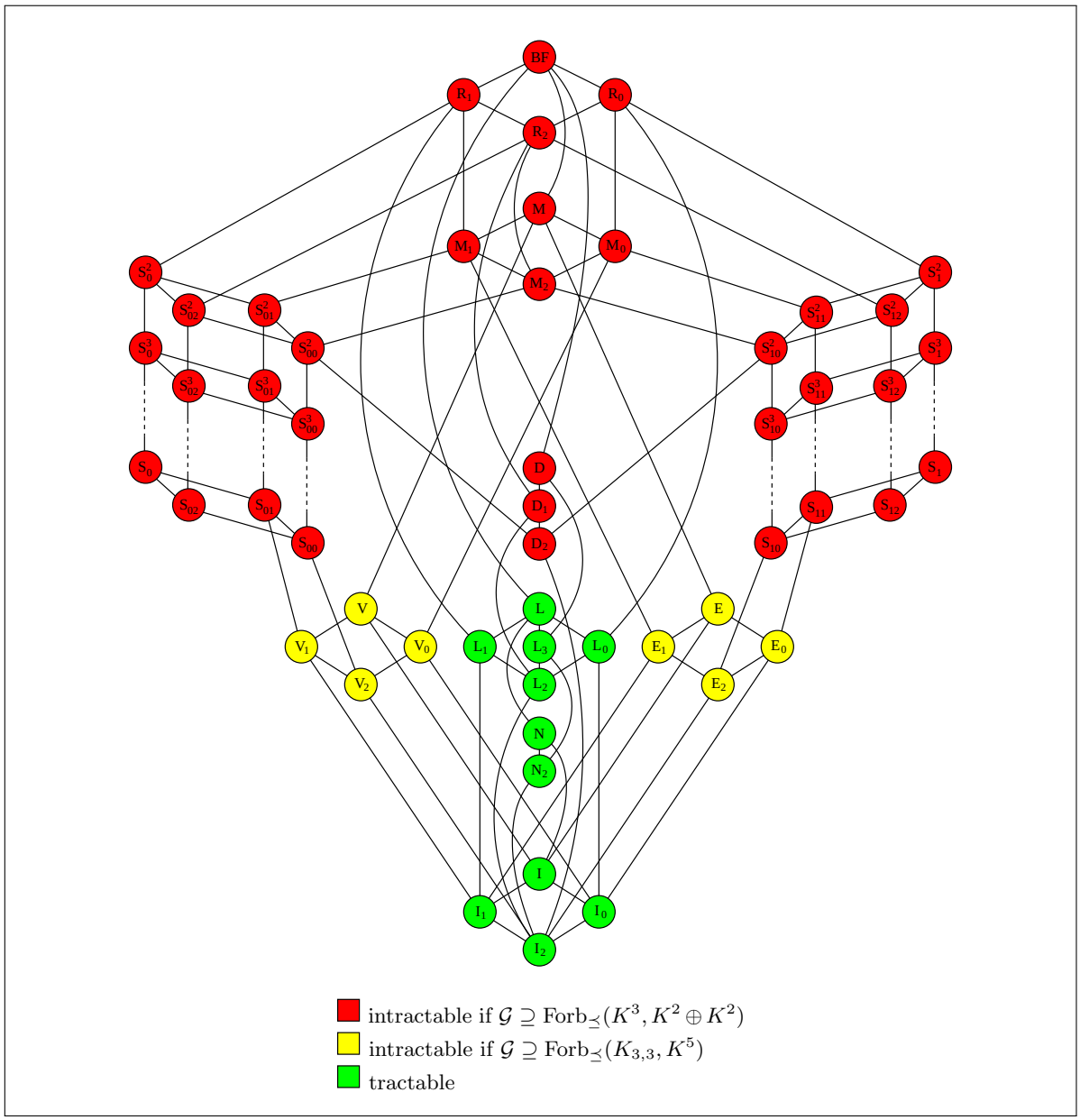


Figure 4.4. Islands of tractability for counting fixed points in boolean systems with succinctly represented local transition functions

Part II

Case study: Internet routing

5. The Border Gateway Protocol

5.1 Background

The Internet is the ultimate communication network at present. Recent estimations from March 2007 [110] show that a total of 1,114,274,426 Internet users worldwide are connected which corresponds to an Internet penetration rate of 16,9 % of the world population. The basic functionality of the Internet is sending data from source entities to destination entities over the data links of the network. The technical characteristics of the data links impose at least two kinds of restrictions on the network:

– only a limited number of entities can share the same data link and
– only a limited amount of data can be sent over the data link in a certain time.

In view of the large variety of Internet infrastructure these restrictions are significant. Consequently, to maintain the functionality the Internet operation mode has been based on the next-hop principle and data fragmentation.

The next-hop principle is involved in the Internet routing problem. The routing problem in the Internet can be characterized by the following terms:

– A *path* is a sequence of entities which pairwise share a direct data link.
– Sending data from an entity to an entity—the next hop—over a direct data link is called *forwarding*.
– A *transmission* is the process of successively forwarding data from a source to a destination over a path with the source as its first and the destination as its last element.
– The selection of a path for a transmission is called *routing*.

To act as intermediaries the entities need routing information. A *datagram* is a self-contained packet consisting of data and routing information. In communication networks of interest all data are transmitted as datagrams. The next-hop principle provides a successful solution to the first restriction above.

A solution to the second restriction is fragmenting and reassembling datagrams when necessary for a transmission through small-capacity networks.

Functionalities of communication networks are specified by protocols, i.e., sets of allowed messages (datagrams) and descriptions of the orders in which these messages have to be communicated. As the Internet aggregates an increasing number of functionalities, many different protocols interact in a non-trivial way. The current Internet is characterized by the TCP/IP protocol suite, a family of about 500 protocols named after the two most important protocols in it: the Transmission Control Protocol (TCP) [125] and the Internet Protocol (IP) [124]. The protocols in the TCP/IP protocol suite can be divided into five layers[1]: the

[1] Note that the TCP/IP layer structure is different to those of the OSI model (Open System Interconnection Basic Reference Model) which provides an abstract seven-layer model for networking (see, e.g., [142]).

physical layer, the datalink layer, the network layer, the transport layer, and the application layer. These layers are hierarchically ordered with the physical layer at the bottom and the application layer at the top. Typically, the functionality of a protocol in a layer above the physical one relies on the functionality of the protocols in the subjacent layers.

The protocols in the network layer are responsible for routing. They handle three major tasks: addressing physical objects, providing fragmentation support of datagram forwarding from a source address towards the destination address, and disseminating and selecting routes from source addresses to destination addresses. The first two tasks are managed by the IP protocol. The third task is under control of routing protocols. The Border Gateway Protocol (BGP or BGP-4) [128] is the *de facto* standard routing protocol for interdomain routing, i.e., it rules how different networks learn which routes they can use to communicate.

This chapter gives a systematic survey on BGP and its functionality. The exposition is based on [154, 93].

5.2 Terminology

In this section we gather the relevant terminology of the IP protocol necessary to see how BGP is involved in Internet routing. In the forthcoming we identify IP with IP version 4 (IPv4). Though newer versions of IP, known as IPng or IPv6 [39], are designed and implemented to overcome some shortcomings of IPv4, mainly the limited number of addresses, IPv4 is still the most used version of IP. Also, the fundamental principles do not differ.

Basically, the Internet Protocol (IP) implements two functions [124]: addressing and fragmentation. Fragmentation is less relevant for routing. So we give a short introduction to addressing and the related forwarding method.

5.2.1 Physical networks

The elementary physical entities in the Internet are computing devices running an implementation of IP and the physical datalink connections between them. We should note that data links typically connect many devices. In this respect they should be graph-theoretically modeled by hyperedges rather than by edges. Technically, connecting many devices via a data link is realized by gadgets such as switches, bridges, and hubs. As long as these gadgets do not follow the IP protocol, they are not considered a visible part of the Internet.

A *physical subnet* is a set of devices connected by a single data link. *Physical networks* are inductively defined as follows:

– Each physical subnet is a physical network.
– The (finite) union of physical networks is a physical network.
– Nothing else is a physical network.

Thus, a physical network can be decomposed in a finite number of physical subnets. If two physical subnets have at least one device in common, they are *connected*. A physical network is connected if and only if for each pair (N, N') of physical subnets there are physical subnets $I_0, I_1, \ldots, I_r$ such that $N = I_0$, $N' = I_r$, and for all $j \in \{1, \ldots, r\}$, I_{j-1} and I_j are connected. The Internet in a physical sense is the physical network maximal subject to set inclusion. Note that the physical size of the Internet is time-depending.

The devices of a physical network are divided into hosts and routers: a *host* is a computing device having only one datalink connection to the physical network. A *router* has at least two datalink connections to the physical network. The boundary of a computing device and a data link is called an *interface*.

5.2.2 Logical networks

The elementary logical entities in the Internet are binary words. Addressing refers to mapping physical entities to logical entities. However, addressing is more than that: an address should indicate where to find the physical entity.

An *IP address* (or simply, *address*) is a word over $\{0,1\}$ of length 32 (or of length 128 in IPv6). Each interface of a physical network has a unique address.

For a word $p \in \{0,1\}^*$ such that $|p| < 32$, the set $p \cdot \{0,1\}^{32-|p|}$ is called a *logical network* (or simply, *network*). The word p is called an *IP prefix* (or simply, *prefix*). Typically, physical networks are embedded into logical networks, i.e., the set of addresses of the interfaces of a physical network is a subset of a logical network. The other way round, a logical network with prefix p can contain an embedded physical network having at most $2^{32-|p|}$ interfaces. Networks are partially ordered according to the dual prefix relation, i.e., if a word p is a prefix of a word p', then the network with prefix p is a *supernetwork* of the network with prefix p' and the network with prefix p' is a *subnetwork* of the network with prefix p.

It is common to denote IP addresses in dot-decimal notation, i.e., the address is separated by dots into four blocks of eight bits and each block is written in decimal.

Example 5.1. In dot-decimal notation the IP address 10000100011001000110011000100001 corresponds to 132.98.100.33.

IP addresses are considered to consist of two parts: a network part and a host part. A network part of size b consists of the first b bits of the IP address and is used to identify the network. It is clear that the network part is just the prefix of the network. The remaining $32 - b$ bits of the IP address are used to identify a certain host or router within the network. The first address in the network, i.e., the address with an all-zero host part, is the network address and cannot be used for addressing a single interface. The last address in the network, i.e., the address with an all-one host part, is the network broadcast address and cannot be used either. Network addresses are denoted by usual addresses.

Following the Classless Inter-Domain Routing (CIDR) address architecture [129, 59], there are two ways to specify uniquely the network part of an address:

– *enhanced prefix format*: Simply attach the length of the network part to the address.
– *netmask format*: A netmask consists of a block of 1's, corresponding to the network part, followed by a block of 0's, corresponding to the host part. Netmasks are also denoted in dot-decimal notation.

In the following we identify prefixes and network addresses. The address 0.0.0.0/0 is the network address of the Internet.

Example 5.2. Suppose the address 132.98.100.33 is assigned to an interface in a network with an 18-bit identifier. Then, the network address is denoted by 132.98.64.0/18 in the enhanced prefix format or is uniquely described by the netmask 255.255.192.0 for IP address

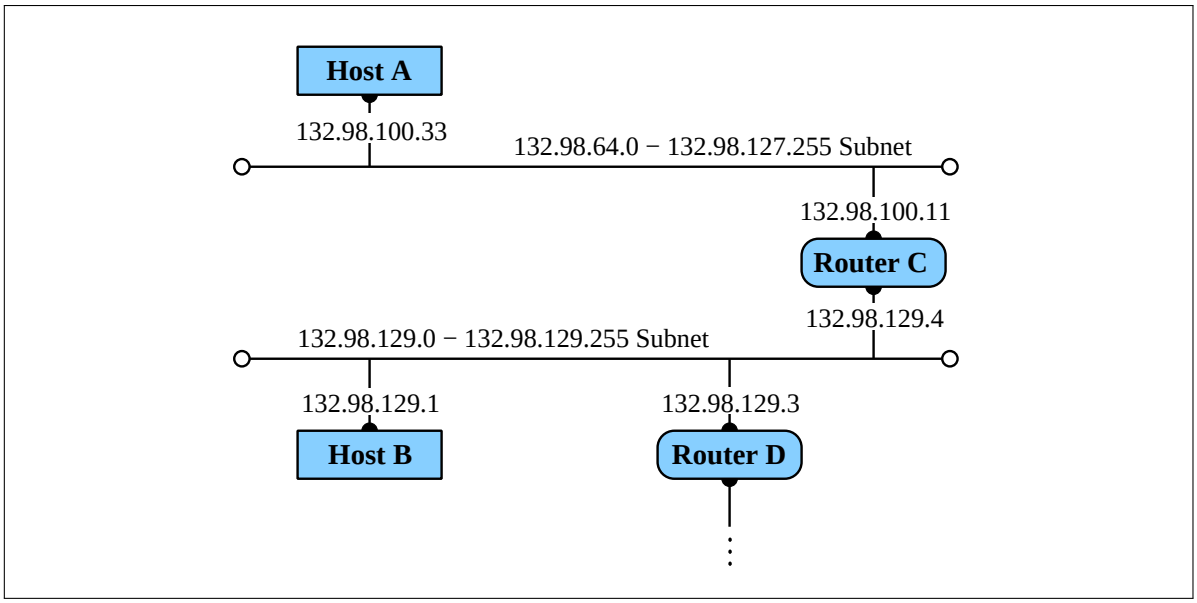


Figure 5.1. A small network example

132.98.100.33 (taking a bit-wise AND). A small network exemplifying the situation is shown in Figure 5.1. The upper subnetwork has the network address 132.98.64.0/18 and the broadcast address 132.98.127.255. The lower subnetwork has the network address 132.98.129.0/24 and the broadcast address 132.98.129.255. All interfaces have addresses in these ranges.

5.2.3 Datagrams and forwarding

A datagram is a self-contained piece of code consisting of user data and routing address information. An *IP datagram* is a datagram starting with a header of 160 bits divided into 12 fields followed by the datagram body containing the user data. The format definition of the header is given in the Figure 5.2. The most interesting fields from the viewpoint of routing are the following:

– Destination Address containing the essential routing information
– Total Length containing the length of the entire IP datagram, including the header, in bytes

It follows that an IP datagram has a maximal size of 65536 bytes (a size not exceeded in IPv6 as well). A detailed description of all fields can be found in [124]. As no datagram type other than IP datagrams occurs in our setting we usually speak of datagrams when meanining IP datagrams.

Forwarding refers to transmitting datagrams from device to device over a common datalink connection in order to bring the data nearer to the destination. Forwarding according to IP works as follows: on the reception of a datagram (which has not reached its destination address), a computing device, typically a router, proceeds with three steps:

1. Take the address written in the Destination Address field of the datagram.
2. Find an associated interface—the *next hop*—for the address in the Forwarding Information Basis.

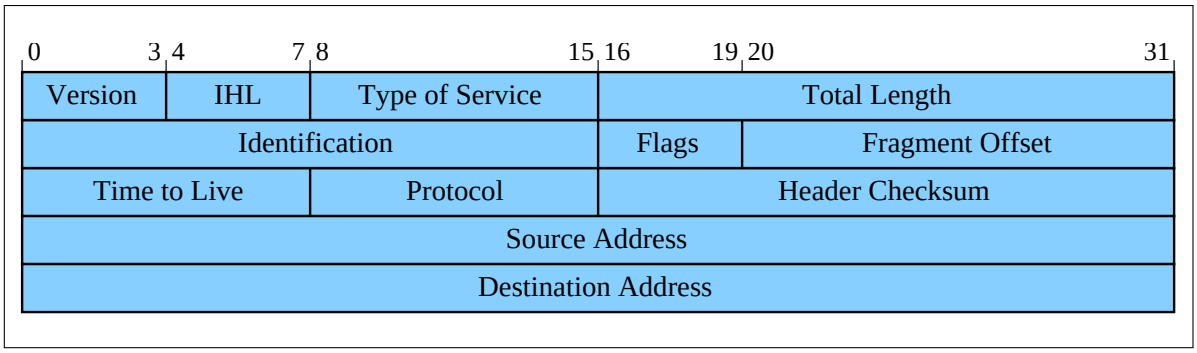


Figure 5.2. The format of the IP header

3. Transmit the datagram over a datalink connection to this interface.

As an additional step the datagram can be fragmented into smaller pieces (which too have the IP datagram format) if the original datagram is too large to traverse a data link. In IPv4 each router on a transmission path is allowed to do this. In IPv6 this is reserved for hosts.

The Forwarding Information Basis (FIB) of a computing device simply consists of a list of network addresses and associated IP addresses (interfaces) in the logical (physical) network. Given an FIB and given a destination address the correct interface is found by solving the *longest prefix matching* problem, i.e., find the network address in the FIB which is the longest IP prefix of the destination address. Several algorithms and data structures have been designed to solve this problem efficiently (see, e.g., [155, 80]).

Destination	**Host A**	**Host B**	**Router C**	**Router D**
132.98.64.0/18	directly connected	132.98.129.4	directly connected	132.98.129.4
132.98.129.0/24	132.98.100.11	directly connected	directly connected	directly connected
0.0.0.0/0	132.98.100.11	132.98.129.3	132.98.129.3	...

Table 5.1. Forwarding Information Bases for hosts and routers in Figure 5.1.

Example 5.3. Table 5.1 shows the content of the Forwarding Information Bases of all devices of our example network in Figure 5.1. Suppose we want to send data from host B to the address 132.98.100.33. That is, the longest prefix that matches is 132.98.64.0/18 and the associated interface has address 132.98.129.4 which belongs to Router C. Note that the prefix 0.0.0.0/0 always matches. This is the default prefix which has to be contained in each forwarding table. For instance, if we want to send data from host B to the address 212.126.34.12, then the default prefix is the only prefix that matches the address. In this case the address 212.126.34.12 lies outside the network under consideration and we do not know where to find a specific network containing the address. We thus send packets to somewhere in the Internet where routers have more information on the address.

5.3 Autonomous Systems

The introduction of Autonomous Systems reduces the complexity of routing, simply by reducing the number of possible paths through the Internet. In this section we define Autonomous Systems and discuss related concepts.

5.3.1 Definition

According to [74], an *Autonomous System* (AS) is a connected group of one or more IP prefixes run by one or more network operators which has a single and clearly defined routing policy. Each AS has a unique natural number for identification. The current version of BGP limits the size of these numbers to 16 bits. More precisely, the set of possible AS numbers is $\{1, 2, \ldots, 65535\}$. In practice, however, the AS numbers in the range 64512–65535 are reserved for private use without global visibility [74].

Recall that an IP prefix is a range of IP addresses a physical network is embedded into. In a connected group of IP prefixes the underlying physical networks are connected. An interface belongs to an AS if its IP address belongs to the AS. A computing device belongs to an AS if one of its interfaces belongs to the AS. A router belonging to some AS is called *visible* if it shares a data link with a computing device belonging to another AS. The other routers are called invisible or *internal*. A visible router is also called a *gateway router*.

The *connectivity graph* (at the AS level) or simply *AS graph* is an undirected graph having the set of assigned AS numbers as its vertex set. The edge set is defined as follows: there is an edge between ASes u and v if and only if a router belonging to AS u and a router belonging to AS v share a common datalink connection. The AS graph is an abstract view on the underlying physical Internet level. Actually, it can be considered as a minor of the graph based on the router level.

Example 5.4. Figure 5.3 shows a small Internet example at the AS level. There is an edge drawn between two ASes if there is a data link connecting routers from both ASes. For instance, AS 2 is connected to AS 1, AS 3, AS 4, and AS 5. As indicated by the detailed view into AS 2, there are two routers (A and B) connected with AS 1, router D shares one data link with AS 4 and another with AS 5, and router E is connected with AS 3. The routers A, B, D, and E are visible. The router E is an internal router. Note that AS 2 embeds a connected physical network.

5.3.2 Interrelationships

An AS is associated with an organization or an administrative domain. Within this organization a network operator is an institution responsible for managing the network at both the physical and the logical level (see, e.g., [154, Chapter 8] for a list of responsibilities). A typical organization owning an AS is an Internet Service Provider (ISP). An ISP sells transmission paths (with differentiated quality of service) to its customers. Thus, a primary operating goal of the network operator within an ISP is maintaining global reachability for its customers, i.e., ideally providing a path in both directions to every reachable prefix.

An ISP can be a customer of another ISP. A customer ISP uses its provider ISP for *transit*, i.e., for transmissions along paths through the network of the provider ISP to destination

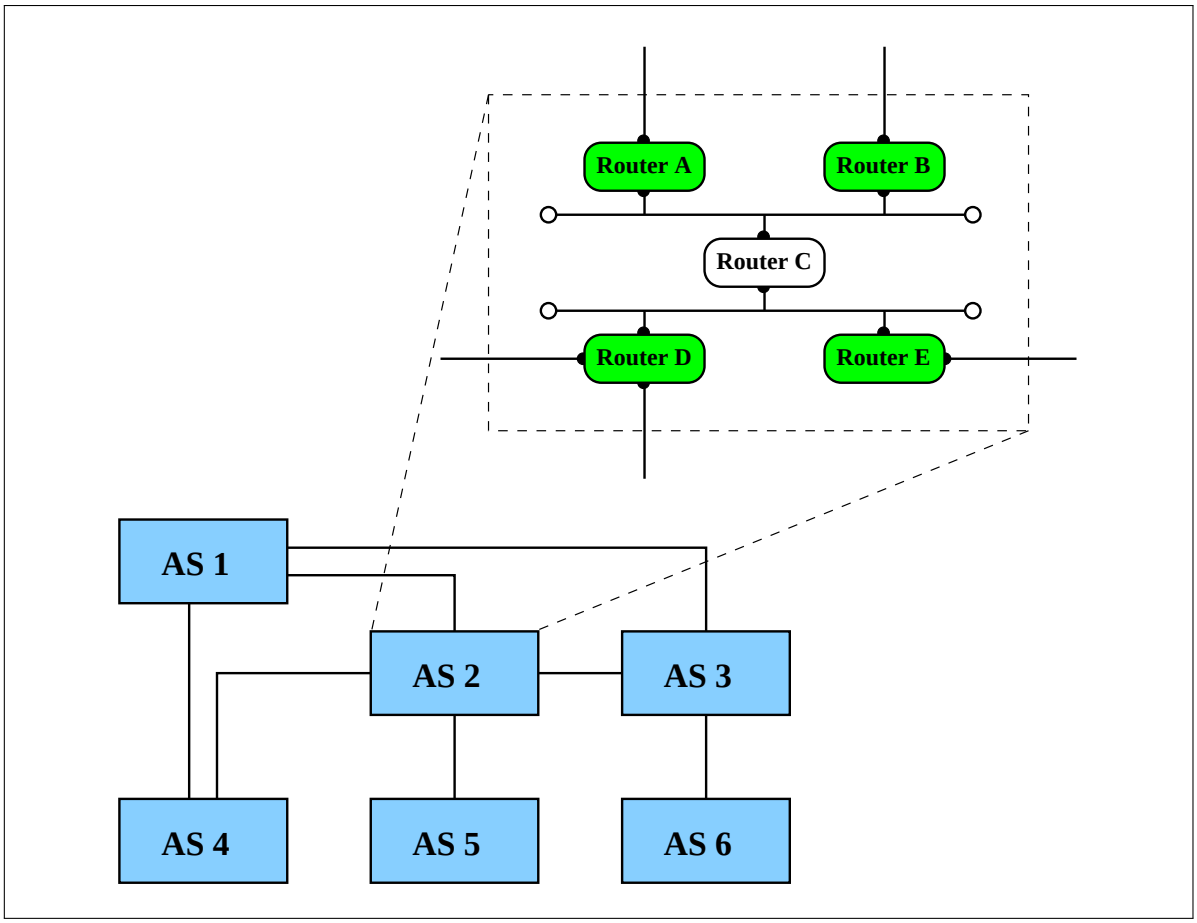


Figure 5.3. A small Internet example at the AS level

prefixes possibly outside any AS owned by the provider ISP. An AS inherits the business relationships of its owning ISP. There are three fundamental types of interrelationships between an AS i and an AS j:

– *customer-to-provider*: the AS i is a customer of the AS j if the ISP owning AS i buys transmission paths from the ISP owning AS j,
– *peer-to-peer*: the ASes i and j provide special paths to their customers without the owning ISPs having a customer-to-provider relationship in either direction,
– *sibling-to-sibling*: the ASes i and j belong to the same ISP.

More peculiar types of relationships appear in the real world (see, e.g., [61]). We restrict ourselves to the three mentioned types.

Example 5.5. Figure 5.4 shows a different view on our Internet example. First, we are only interested in the relation between the different ASes. An arrow from an AS to another AS indicate a customer-to-provider relationship. For instance, AS 4 is a customer of both AS 1 and AS 2. AS 1 has no provider. The large ISP behind AS 1 is a so-called *tier-1* provider. All other ASes have to pay at least one AS for transit. The ISPs 2 and 3 have a peer-to-peer relationship.

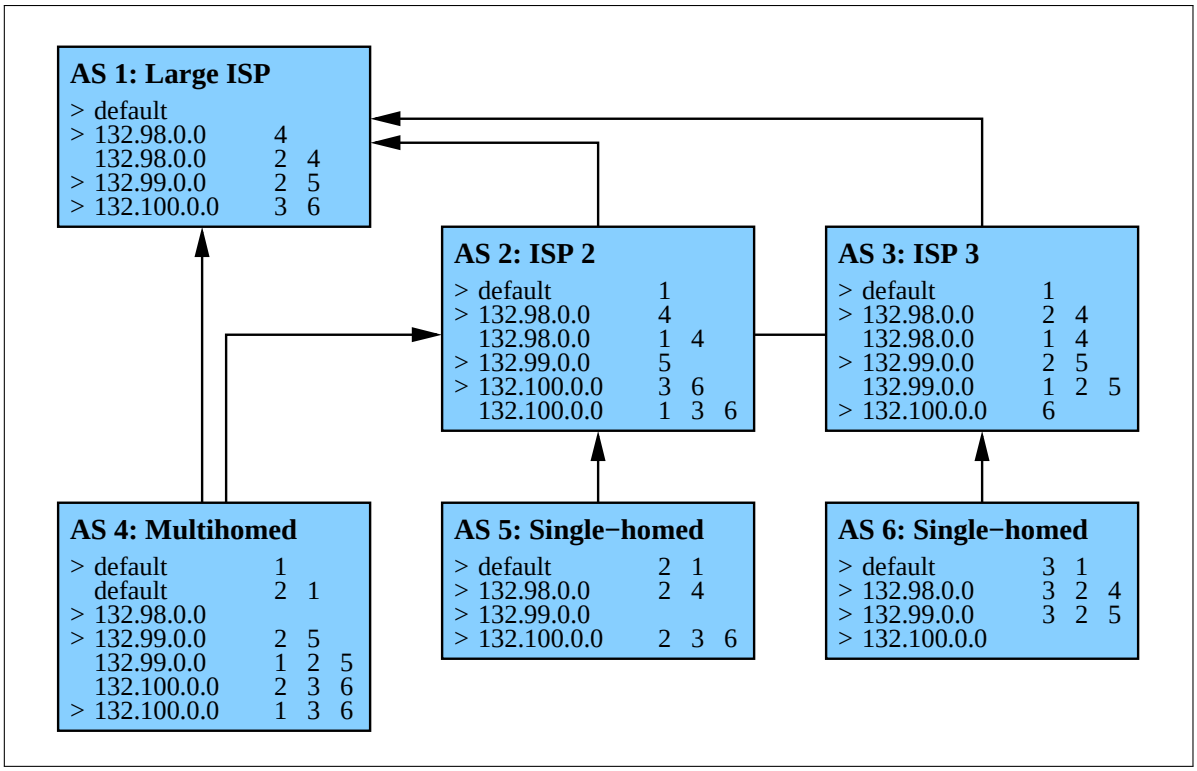


Figure 5.4. A small Internet example with local Routing Information Bases

5.3.3 Routing policies

A *routing policy* is a fixed and formalized set of rules outlining the distribution of routing information between an AS and the neighbored ASes. As routing policies are implemented in gateway routers, a routing policy can be viewed as the union of all implemented sets of rules of gateway routers of an AS. In case of a policy change the gateway routers need to synchronize their set of rules to implement a single routing policy of the AS. Reasonably, the routing policies of an AS are in accordance with existing customer-to-provider and peer-to-peer relationships.

Example 5.6. Inside the boxes of each AS in Figure 5.4 it is shown which path is considered best by the AS to reach a certain IP prefix. As an example let us look at AS 4. The IP prefix 132.98.0.0/16 is located in AS4 itself and is thus reachable over a local path. In contrast, to reach the prefix 132.100.0.0/16 which is owned by AS 6 the chosen path goes through AS 1. Thus, the routers within AS 4 forward datagrams with destination prefix 132.100.0.0/16 towards some gateway routers sharing a data link with a gateway router of AS 1. There the datagrams leave AS 4 and enter AS 1. An analysis of the local Routing Information Base of AS 4 leads to the following (incomplete) rule for all prefixes: a shorter path is preferred over a longer path and paths through AS 1 are preferred among equal-length paths. As another example, AS 3 prefers the path through AS 2 over the path through AS 1 to reach the IP prefix 132.98.0.0/16 in AS 4. This is reasonable as AS 3 and AS 2 have a peer-to-peer relationship whereas AS 3 is a customer of AS 1. Transmissions through AS 2 are thus typically cheaper than those through AS 1.

5.3.4 Routing hierarchy

Example 5.6 demonstrates that two mechanisms are needed to select an address-to-address path:

– one for deciding which sequence of ASes leads to the destination address
– one for deciding which sequence of routers traverses an AS

The mechanisms are hierarchically ordered: a local path can only be selected after the decision has been made which pair of gateway routers is to be connected for an AS traversal. Any routing protocol used within an AS for selecting local paths is called an *intra-AS routing protocol* or *interior gateway protocol* (IGP). There are two classical types of such protocols (see, e.g., [93]):

– A *link-state protocol* implements a link-state algorithm. A link-state algorithm consists of two phases: in the first phase, the cost of each connection in the network is broadcast until all devices know the complete cost matrix. In the second phase, at each device an appropriate single-source shortest path algorithm is executed on the same cost matrix. Due to their communication overhead link-state algorithms do not scale well with an increasing number of devices. Examples of link-state protocols are the Open Shortest Path First (OSPF) protocol [115, 116] or the Intermediate-System-to-Intermediate-System (IS-IS) protocol [118, 29].
– A *distance-vector protocol* implements a distance-vector algorithm. A distance-vector algorithm works in an iterative, asynchronous, and distributed fashion and converges always. In essence, each device communicates to its neighbors its currently known distances to all destination devices in the network. On reception of such a distance vector a source device computes a new distance to each destination in the network as the minimum of the cost of a connection to some neighbor plus the neighbor's distance to the destination (distance-relaxation rule). If a distance to some destination has changed, then the new distance vector is communicated. Distance-vector protocols tend to congest routers having a high betweenness centrality, i.e., those routers that lie on many shortest paths. They do not respect administrative autonomy which makes them only suitable for usage within a single AS. Examples of such protocols are the Routing Information Protocol (RIP) [75, 103] or the Enhanced Interior Gateway Routing Protocol (EIGRP) [4].

An *inter-AS routing protocol* or *exterior gateway protocol* (EGP) is a protocol responsible for routing at the AS level. BGP is the *de facto* standard inter-AS routing protocol. Note that BGP can be also applied as an intra-AS routing protocol. Sometimes BGP used within an AS is denoted by iBGP and BGP used between ASes is denoted by eBGP.

5.4 Protocol outline

We turn to the specification of BGP as proposed in [128]. What is currently understood as BGP refers to BGP-4. The precursor versions of the protocol are completely obsolete. In contrast to the link-state and distance-vector protocols mentioned in Subsection 5.3.4, BGP can be characterized as a *path-vector protocol*.

A computing device capable of performing BGP communications is called a *BGP speaker*. The primary function of a BGP speaker is exchanging routing information with neighbored

BGP speakers. The information consists of network reachability information and a list of ASes that the information has traversed. BGP uses the TCP protocol for establishing reliable connections.

In the following we describe only those parts of the BGP protocol that are relevant for the routing problem, thus omitting details concerning the communication process itself. In particular, we omit all error handling features. Moreover, we assume that all necessary BGP communications can be safely processed.

5.4.1 Operating mode

The general operating mode of BGP is the following: Two BGP speakers form a TCP connection between one another. They exchange messages to open and confirm the connection parameters. A connection between BGP speakers of different ASes is called an *external link*. A connection between BGP speakers within the same AS is called *internal link*. Initially, the complete routing information is exchanged between the BGP speakers. Subsequently, incremental updates are sent whenever routing information of one of the BGP speaker has changed.

A *route* is defined to be a tuple consisting of an IP prefix and attributes of a path to the AS containing the IP prefix. Routes are advertised between a pair of BGP speakers in update messages. The IP prefix is contained in the Network Layer Reachability Information (NLRI) field, and the path together with a set of attributes is reported in the same update message. Routes are stored in Routing Information Bases (RIBs). Conceptually, there are three types of RIBs:

- An *Adj-RIB-In* contains information on routes that have been learned from inbound update messages.
- A *Loc-RIBs* contains information on routes that have been selected from the Adj-RIBs-In by applying local policies.
- An *Adj-RIBs-Out* contains information on routes that have been selected for advertisement to BGP speakers in an outbound update message.

A BGP speaker has an Adj-RIB-In and an Adj-RIB-Out for each possible connection but only one Loc-RIB. Note that RIBs need not be implemented separately.

BGP also provides mechanisms to inform another BGP speaker that a previously advertised route is no longer available, e.g., the IP prefix can be included in the Withdrawn Routes field of an update message or a replacement route to the same IP prefix can be advertised.

5.4.2 Message formats

All BGP information is exchanged in form of messages. A BGP message has up to 4096 bytes consisting of a fixed-size header of 19 bytes.

Marker	Length	Type	Message contents
16 bytes	2 bytes	1 byte	0-4077 bytes

Table 5.2. BGP message format.

Table 5.2 shows the structure of a BGP message. The Marker field usually contains all one's and is intended to check whether communication works correctly. Any zero occuring in the Marker field induces an error handling procedure. The Length field contains the length of BGP message in bytes. The Type field contains a code for the type of the message. Originally, there are four types of messages: open, update, keepalive, and notification messages. For our purposes, only update messages are relevant.

UR Length	Withdrawn Routes	Total PA Length	Path Attributes	NLRI
2 bytes	Variable	2 bytes	Variable	Variable

Table 5.3. BGP update message format.

In Table 5.3 the structure of an update message is shown. The fields of an update message are used as follows:

– The Unfeasible Routes (UR) Length field contains the length of the Withdrawn Routes field in bytes; zero means that the field is not present in this message.
– The Withdrawn Routes field is optional and contains a variable-length list of IP prefixes encoded by the enhanced prefix format.
– The Total Path Attribute (PA) Length field contains the length of the Path Attribute field in bytes; zero means that the Path Attribute field and the NLRI field are not present in this message.
– The Path Attribute field is optional and contains a variable-length list of path attributes which are explained in more detail in the next subsection.
– The Network Layer Reachability Information (NLRI) field is optional and contains a variable-length list of IP prefixes encoded by the enhanced prefix format.

Note that an update message can advertise only one route but can simulteanously withdraw many routes. An advertised route thus contains information on how to reach each prefix present in the NLRI field of an update message.

5.4.3 Path attributes

Path attributes are contained in the Path Attribute field of an update message.

Attribute Flags	Attribute Type Code	Attribute Length	Attribute Value
1 byte	1 byte	1-2 bytes	Variable

Table 5.4. Path attribute format.

The structure of a path attribute is shown in Table 5.4. The meaning of the fields is as follows:

– The Attribute Flags field contains information on how to process the attribute. For instance, if Bit 0 is set to zero and Bit 1 is set to one, then the attribute is well-known mandatory, i.e., it must be included in every update message. If Bit 3 is set to zero, then the Attribute Length field is one byte, otherwise it is two bytes. Though flags are important in practice, they are not relevant in our setting.

– The Attribute Type Code field contains information on the meaning of the attribute.
– The Attribute Length field contains the length of the Attribute Value field in bytes.
– The Attribute Value field contains data according to the Attribute Type Code field.

Important available types of attributes as described in the Attribute Type Code field are the following:

– ORIGIN is a well-known mandatory attribute that defines the origin of the path information.
– AS_PATH is a well-known mandatory attribute that is composed of a sequence of AS path segments. A path segment can be one of the two types AS_SET, i.e., an unordered set of ASes a route in the update message has traversed, and AS_SEQUENCE, i.e., an ordered set of ASes a route in the update message has traversed. The AS_SET type of path segment is involved in route aggregation.
– NEXT_HOP is a well-known mandatory attribute that defines the IP address of a gateway router that should be used for datagram forwarding towards the destination listed in the NLRI field of the update message.
– MULT_EXIT_DISC is an attribute that can be used to discriminate among multiple external links of an AS.
– LOCAL_PREF is an attribute that is used to inform the BGP speakers with the same AS of the degree of preference of the originating BGP speaker for an advertised route.
– COMMUNITIES is an attribute for aiding policy management. A community is a set of destination prefixes sharing a common property [149].

The attribute type most relevant for route dissemination is the AS_PATH type. At an internal link the AS_PATH attribute is processed as follows:

– When a BGP speaker originates a route then the originating speaker includes an empty AS_PATH attribute in all update message sent to BGP speakers in its own AS.
– When a BGP speaker propagates a route learned from another speaker then the AS_PATH attribute associated with the route is not modified.

At an external link the AS_PATH attribute is processed as follows:

– When a BGP speaker originates a route then the originating speaker includes its own AS number as the only entry in the AS_PATH attribute in all update messages sent over the external link.
– When a BGP speaker propagates a route learned from another speaker then the AS_PATH attribute associated with the route is modified depending on the type of the path segment:
  – If the first path segment of the AS_PATH attribute is of type AS_SEQUENCE, the BGP speaker prepends its own AS number to the sequence.
  – If the first path segment of the AS_PATH attribute is of type AS_SET, the BGP speaker prepends a new path segment of type AS_SEQUENCE to the AS_PATH attribute, including its own AS number in that segment.

5.4.4 Route propagation

The update message format is designed to support the process of propagating routes among BGP speakers. When a BGP speaker receives an update message from a neighboring BGP speaker containing withdrawn routes, it removes all routes from the corresponding Adj-RIB-In which have an NLRI listed in the Withdrawn Routes field of the message. When a BGP

speaker receives a new route in an update message from a neighboring BGP speaker, it executes the following procedure:

1. Check the *inbound filters* defined for the connection. If the route does not pass all filters, then the procedure stops.
2. Insert the new route in the corresponding Adj-RIB-In.
3. Execute the *route-selection algorithm* on all routes present in any Adj-RIB-In which have the same NLRI as the new route. If the new route is not selected as the best route, then the procedure stops.
4. Include the new best route in the Loc-RIB and include the next-hop address for NLRI in the Forwarding Information Base. The old best route to NLRI is removed from Loc-RIB and the old next-hop address is removed from the Forwarding Information Base.
5. Revoke the old best route in update messages to those BGP speakers that had received the old best route.
6. Check the *outbound filters* for each link. If the new best route passes the filter for the link, then include the route in the corresponding Adj-RIB-Out. Note that for an internal link there is typically no link.
7. Send update messages advertising the new best route to all neighboring BGP speakers for which the Adj-RIB-Out has changed. Recall how the AS_PATH attributes are processed for each link.

The procedure follows an idealtypical scheme. In practice there is space for implementation diversity. This does not only concern filters and the route-selection algorithm. For instance, route flap damping leads to a different route-revocation procedure in the fifth step (see, e.g., [154, chapter 10]). The technique is a best practice which is not covered by the BGP protocol [156, 104]. Examples which are covered by the protocol are route aggregation and handling overlapping routes [128]. Both techniques lead to minor changes in the propagation process.

5.4.5 Route-selection algorithms and filters

Local routing policies influence the route propagation process

– by inbound filters,
– by the route-selection algorithm, and
– by outbound filters.

The route-selection algorithm is the central tool for implementing a routing policy. It provides a mechanism for selecting the best route from the set of available routes from different neighbors. A standard implementation of the route-selection algorithm works as follows: compute for each route under consideration an integer local preference according to the local Policy Information Base and choose the route with highest local preference. If the highest local preference is taken by more than one route, then iteratively apply a set of tie-breaking rules until a single route is selected. The following set of criteria is recommended for application in exactly this order [128]:

1. highest local preference
2. lowest number of AS path segments
3. lowest MULT_EXIT_DISC entry (if all routes have such an entry)
4. lowest cost of a path through the same AS to the next-hop address

5. lowest IP address of a BGP speaker advertising over an external link
6. lowest IP address of a BGP speaker advertising over an internal link

In contrast to the prescribed BGP message format which allows no deviation practical route-selection algorithms have additional steps throughout the selection process. The scheme above ignores such vendor-dependent extra steps. Furthermore, there is no requirement that all BGP speakers within the same AS agree on how to select the best route.

Filters are algorithms for discarding undesirable routes. Routes not discarded can be transformed in the process. Inbound filters implement the import policies of an AS. They are primarily intended to exclude routing loops, i.e., if a BGP speakers detect its own AS number in the received AS path, then the route is discarded. Furthermore, inbound filters are used to influence the route-selection algorithm, e.g., by assigning weights to routes or by prepending the own AS number one or more times to the beginning of the AS paths in order to make the path longer. Outbound filters implement the export policies of an AS.

The rules for manipulating routes and for setting local preferences are specified by *filter lists* or *route maps* in Policy Information Bases. In general, there is no specific formalization for expressing these rules. Typically, they are described by if-then-constructions: if the head of the rule matches a route, then a specified action is executed on this route.

Example 5.7. A unifying approach to define import and export policies is the Routing Policy Specification Language (RPSL)[2, 108] intended to collect routing policies in a Routing Registry. For instance, the routing policy of AS4 in Figure 5.4 could look like the following in RPSL format:

```
aut-num:    AS4
import:     from AS1
            action pref=2; med=10
            accept ANY
import:     from AS2
            action pref=2; med=20
            accept ANY
export:     to AS1
            announce NOT AS2
export:     to AS2
            announce NOT AS1
```

In this fragment, `aut-num` is the object that describes the AS by its number. The `import` clauses specify that the routers in AS 4 import all routes advertised by AS 1 routers and AS 2 routers. Note that the `pref` object differs from the local preference value in the route-selection algorithm: in the RPSL semantics a lower `pref` value is preferred over a larger one. In our case routes received from AS 1 and from AS 2 have equal values. The difference in the `med` values (which correspond to MULT_EXIT_DISC field) implies that shorter AS paths are preferred. The `export` clauses specify that AS 4 exports no route to AS 1 received from AS 2 and vice versa.

5.5 The Selective Export Rule

Using various kinds of simple path-vector protocols, in several studies it has been shown that certain configurations can force the BGP route propagation process to not converge (see, e.g., [73, 72, 52, 53, 54, 34]). Things are worse: in general the problem of deciding whether an explictly given configuration is bad in this sense is NP-complete, even for very extensional representations [72]. However, all these configurations are based on rather bizarre export-policy patterns, e.g., "always prefer a path of length two over other paths" [73]. In practice, routing policies are much more reasonable.

In the last section of this chapter we describe a prominent example of a policy pattern—the Selective Export Rule [1, 82, 61]. It is an extension of the most basic rule for policy routing "send routes only to paying customers" [154, p. 11] and applies to all business relationships among ASes.

In the following we consider an AS v in the AS graph. According to the business relationships we divide the set $N(v)$ of neighbors of v into the following sets:

- $\mathrm{Cust}(v)$ is the set of all customers of v.
- $\mathrm{Prov}(v)$ is the set of all providers of v.
- $\mathrm{Peer}(v)$ is the set of all peering partners of v, i.e., if $u \in \mathrm{Peer}(v)$ then u and v have a peer-to-peer relationship.
- $\mathrm{Sibl}(v)$ is the set of all siblings of v, i.e., if $u \in \mathrm{Sibl}(v)$ then u and v have a sibling-to-sibling relationship. We let $\mathrm{Sibl}(v)$ contain v as well.

Some of the sets may be empty.

Let $R(v)$ denote the set of all AS paths contained in the Loc-RIB of the AS v. Assumed that there are no misconfigurations of BGP, all AS paths in $R(v)$ are loopless and not including v. Here, we say that an AS path is *loopless* whenever between two sibling ASes on the path, no non-sibling AS is passed. Based on the neighborhood classification we further divide $R(v)$ into four categories. A loopless AS path $(u_1, \ldots, u_r) \in R(v)$ is

$$\begin{aligned}
\text{a } \textit{customer route} \text{ of } v &\iff_{\text{def}} \text{ leftmost } u_i \notin \mathrm{Sibl}(v) \text{ lies in } \mathrm{Cust}(v),\\
\text{a } \textit{provider route} \text{ of } v &\iff_{\text{def}} \text{ leftmost } u_i \notin \mathrm{Sibl}(v) \text{ lies in } \mathrm{Prov}(v),\\
\text{a } \textit{peer route} \text{ of } v &\iff_{\text{def}} \text{ leftmost } u_i \notin \mathrm{Sibl}(v) \text{ lies in } \mathrm{Peer}(v),\\
\text{an } \textit{own route} \text{ of } v &\iff_{\text{def}} \text{ for all } 1 \le i \le r,\ u_i \in \mathrm{Sibl}(v).
\end{aligned}$$

AS v exports to	**Provider**	**Customer**	**Peer**	**Sibling**
Own routes	Yes	Yes	Yes	Yes
Customer routes	Yes	Yes	Yes	Yes
Provider routes	No	Yes	No	Yes
Peer routes	No	Yes	No	Yes

Table 5.5. The Selective Export Rule.

The Selective Export Rule summarized in Table 5.5 provides a scheme of which route is allowed to be exported to a certain neighbor depending on the type of the route and the class of the neighbor.

Example 5.8. It is easily seen that all AS paths shown in the Loc-RIBs of our small Internet example in Figure 5.4 result from export policies which respect the Selective Export Rule. In particular, consider AS 4 together with the RPSL fragment from Example 5.7. Every route AS 4 receives from AS 1 is a provider route and should not be exported to the provider AS 2 according to Table 5.5. Analogously each route received from AS 2 should not be exported to AS 1. The both `export` clauses of the RPSL fragment do exactly this.

Based on the Selective Export Rule some guidelines can be identified guaranteeing convergence of the BGP route propagation process. The following guideline has been proposed in [62]:

> Suppose an AS v possess two routes r_1 and r_2 containing the AS paths $(u_1, \ldots, u_k)$ and $(u'_1, \ldots, u'_\ell)$, respectively. If $u_1 \in \mathrm{Cust}(v)$ and $u'_1 \in \mathrm{Peer}(v) \cup \mathrm{Prov}(v)$ then the LOCAL_PREF of r_1 is higher than LOCAL_PREF of r_2

If all ASes respect the Selective Export Rule for their export policies and this guideline for the route-selection algorithms then it can be proven that the route propagation process always stops [62]. This guideline can be relaxed (at the cost of certain structural assumptions to the business relations in the AS graph) to equally ensure convergence. However, a general description of all policy patterns with convergence guarantee is not yet available.

5.6 Summary

In this chapter we presented an introduction to the Border Gateway Protocol comprehensive enough to build future models of path-vector protocols on it. The main functionality of BGP is a process of disseminating AS paths through the Internet for guiding the selection of transmission paths. It does so with guaranteeing any convergence properties. The convergence heavily depends on the local routing policies.

The dynamics of the route propagation is far from being understood. Business relationships between ASes have significant influence on this dynamics. Accurate models and precise analysis methods for AS relationships are key ingredients for getting deeper insights into the dynamical process.

6. Interrelationship analysis

6.1 Background

In the last chapter we have seen that the dynamics of Internet inter-domain routing is massively influenced by business relations among ASes which find their way into local routing policies. Numerous studies have exposed that for the purpose of ex post analyses of poor network behavior this information is essential (see, e.g., [73, 136, 96, 71, 95, 94, 144, 102, 55] to name only few). For instance, the accuracy of heuristic methods for locating routing instabilities depends on the quality of knowledge on these interrelationships [55, 97]. As business contracts are intentionally considered to be commercial secrets—storing the relevant RPSL information in an Internet Routing Registry is recommended but voluntary [67]—, many researchers have proposed methods to elicit this crucial information indirectly (see, e.g., [61, 140, 40, 50, 41, 165, 130, 45, 44]).

In [61], the seminal work on inferring contractual relationships on basis of sets of AS paths observable from BGP updates, heuristic approaches were devised to classify relationships into customer-to-provider, peer-to-peer, and sibling-to-sibling. A key observation is that under rational economic behavior, i.e., the Selective Export Rule, AS paths exhibit a regular, so-called valley-free structure. That is, after traversing a provider-to-customer or peer-to-peer edge, the AS path cannot traverse a customer-to-provider or peer-to-peer edge. Thus, each path has some top provider. To identify top providers in AS paths, heuristics follow basically two evident assumptions (see, e.g., [68, 61, 143]):

A Providers are much larger than customers in terms of infrastructure.
B The size of an AS is proportional to its degree in the AS graph.

In principle the algorithms in [61] iteratively search for vertices with maximum degree in a given AS path set to identify top providers. This approach has been further elaborated in [65, 140, 165, 79]. Degree-based heuristics are certainly practicable if we are able to get a representative sample of all AS paths of the Internet (see [32] for a critical discussion). But, due to their over-sensitivity to path sets, they have weaknesses in well-defined analytical test situations (such as, e.g., in [55, 62]).

A purely combinatorial treatment, neither involving **A** nor **B**, was done in [40, 50, 41]. There, the authors described a linear-time algorithm solving the All-Paths Type-of-Relationship Problem, suggested in [140]: given a path set P, is there an orientation of the edges (indicating provider-to-customer or customer-to-provider relation) such that all paths of P are valley-free. The algorithm is based on a reduction to 2-SAT which is well-known to be solvable in linear time. In contrast, finding an orientation maximizing the number of valley-free paths is not polynomial-time approximable within factor $O(n^{1-\varepsilon})$ (unless NP = ZPP) [40, 50, 41]. Positively, for path lengths at most ℓ, there is a polynomial-time algorithm with

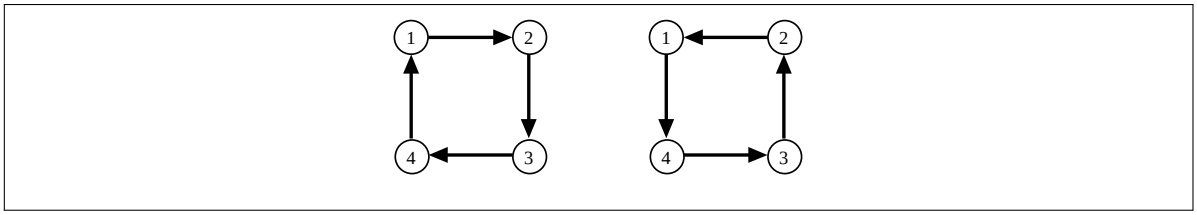


Figure 6.1. All valley-free orientations for path set $\{(1,2,3),(2,3,4),(3,4,1),(4,1,2)\}$

approximation ratio $\frac{\ell+1}{2^\ell}$ (sometimes better by semi-definite programming for MAX 2-SAT) [40, 50, 41]. Though computationally elegant and robust, these algorithms often lead to unrealistic relationships (i.e., well-known global providers appearing as customers of small ASes [45]).

To get closer to reality, several proposals have been made. One of them calls for partialness-to-entireness algorithms [165, 40]. The basic idea is to infer the entire AS relationships from partial information obtainable from data sources other than BGP paths (see, e.g., [138, 165]). Another proposal links degree-based heuristics and combinatorial optimization by considering weighted MAX 2-SAT with weights depending on degree gradients [45]. Positive experimental results suggest the fruitfulness of incorporating degree information into a combinatorial setting. Nevertheless, over-sensitivity to path sets when using assumption **B** is still problematic.

Our contribution also lies in the middle of the heuristic and the combinatorial approaches to the inference problem. Instead of using both assumptions **A** and **B**, we will only employ assumption **A** to avoid path-set over-sensitivity. The size rule **A** (and some similar one's) bears enough information to impose a global structure on oriented AS graphs. We deduce several unrealistic relationship patterns which we will forbid to appear in AS graph orientations. A fundamental pattern of this type is an oriented cycle within the graph, as this would imply that some provider is its own customer. Acyclic orientations are constructed by the degree-based heuristic approach of [61, 140]. In contrast, the 2-SAT-based algorithm generally constructs cycles due to the internal computation of strongly connected components. As an example, Figure 6.1 shows a path set for which all valley-free orientations (restricted to customer-to-provider relationships) impose a cycle on the AS graph. However, as a prerequisite for studies related to the Internet topology (see, e.g., [62, 49]), acyclicity is a requirement for realistic AS relationships.

In this chapter we focus on algorithms solving the Acyclic Type-of-Relationship Problem, i.e., given a path set, find an orientation of the edges according to some types of AS relationships such that the oriented AS graph is acyclic (with respect to our forbidden patterns) and all AS paths are valley-free. As possible AS relationships we include customer-to-provider, peer-to-peer, and sibling-to-sibling. Furthermore, we meet the partialness-to-entireness requirement by examining a number of problems versions which are parameterized by sets of edge types available for describing explicit pre-knowledge and sets of edge types available for completion of partial orientations. We give a complete complexity classification of all 56 cases (8 type sets for pre-knowledge and 7 type sets for completion). The most relevant practical result is a linear-time algorithm for finding an acyclic and valley-free completion using customer-to-provider relations given *any* kind of pre-knowledge. Interestingly, if we allow sibling-to-sibling relations for completions, then most of the non-trivial inference problems become NP-hard.

The Acyclic Type-of-Relationship Problem asks for algorithms testing whether all paths of a given path set allow acyclic and valley-free orientations. In practice, collected path sets are expected to fail this test. Thus, we also study the problem to find acyclic orientations which maximize the number of valley-free paths. We prove approximability lower bounds for the Maximum Acyclic Type-of-Relationship Problem and we design a fast heuristic for finding acyclic orientations which are valley-free on a large part of a given path set.

We finally complement our theoretical studies with an experimental analysis.

6.2 Terminology

In this section we gather graph-theoretical notations relevant for studying AS relationship inference problems.

Let $G = (V, E)$ be an undirected (simple) graph. We assume that $(u, v) \in E \Leftrightarrow (v, u) \in E$. A (mixed) *orientation* φ *of* G is a mapping from E to T where T denotes the set of possible edge-types. Note that we assume that the orientation is a total mapping. In case that a mapping φ is partial we explicitly speak of the partial orientation φ. An edge is called *oriented* (with respect to a partial orientation φ) if it lies in the domain of φ. When a graph G is oriented by φ, we denote this by the pair (G, φ). A directed graph is a graph oriented with type set $T = \{\leftarrow, \rightarrow\}$. We consider type sets having the following edge-types and interpretations:

$\rightarrow$	indicating a customer-to-provider relationship
$\leftarrow$	indicating a provider-to-customer relationship
$-$	indicating a peer-to-peer relationship
$\leftrightarrow$	indicating a sibling-to-sibling relationship

Throughout this work we only consider orientations φ that are consistent with respect to $\rightarrow$. That is, for all $(u, v) \in E$ if $\varphi(u, v) = \leftarrow$ then $\varphi(v, u) = \rightarrow$ and if $\varphi(u, v) = \rightarrow$ then $\varphi(v, u) = \leftarrow$. Thus, if we allow $\rightarrow$ as a possible edge type, then we immediately allow $\leftarrow$ as a possible edge type as well.

We extend φ from edges to walks homomorphically. Let $(v_0, v_1, \ldots, v_m)$ be any walk in a graph G. Then $\varphi(v_0, v_1, \ldots, v_m)$ is defined to be the sequence $\varphi(v_0, v_1)\varphi(v_1, v_2) \ldots \varphi(v_{m-1}, v_m)$, i.e., generally a word in $\{\leftarrow, \rightarrow, -, \leftrightarrow\}^*$. We typically use regular expressions to describe walk types given an orientation. We also use the infix notation $u\varphi(u, v)v$ for an oriented edge $(u, v) \in E$. This also extends to walks.

6.3 Relationship models

In this section we describe two models needed to infer meaningful relationship structure: valley-freeness and acyclicity.

6.3.1 Valley-freeness

Valley-freeness is a graph-theoretical consequence of the Selective Export Rule. We define valley-freeness in terms of regular patterns of paths.

Definition 6.1 (Gao [61]). Let (G, φ) be an oriented graph. A loopless path $(v_0, \ldots, v_m)$ is said to be *valley-free* in (G, φ) if and only if $\varphi(v_0, \ldots, v_m)$ belongs to

$$\{\rightarrow, \leftrightarrow\}^*\{\leftarrow, \leftrightarrow\}^* \quad \cup \quad \{\rightarrow, \leftrightarrow\}^* - \{\leftarrow, \leftrightarrow\}^*.$$

Theorem 6.2 (Gao [61]). *Let $G = (V, E)$ be an AS graph. Let P be any set of AS paths of all local Routing Information Bases, i.e., $P \subseteq \bigcup_{v \in V} R(v)$. If all ASes export their routes according to the Selective Export Rule, then there exists an orientation of G such that all AS paths in P are valley-free.*

Proof. Recall from Section 5.5 that the neighbors of an AS can be divided into the four sets $\mathrm{Cust}(v)$, $\mathrm{Prov}(v)$, $\mathrm{Peer}(v)$, and $\mathrm{Sibl}(v)$. Assume to the contrary that for all orientations φ there is a vertex $v \in V$ such that $(u_1, \ldots, u_k) \in P \cap R(v)$ but $(u_1, \ldots, u_k)$ is not valley-free in (G, φ). Fix such φ and $(u_1, \ldots, u_k)$. That is, there exist $1 \leq i < j < k$ such that one of the following cases is true:

1. $u_{i+1} \in \mathrm{Cust}(u_i)$ and $u_{j+1} \in \mathrm{Prov}(u_j) \cup \mathrm{Peer}(u_j)$
2. $u_{i+1} \in \mathrm{Peer}(u_i)$ and $u_{j+1} \in \mathrm{Prov}(u_j) \cup \mathrm{Peer}(u_j)$

Both cases can be treated similarly. We consider the first case. Let u_{i+1} be a customer of u_i. Let j be the smallest index such that $u_{j+1} \in \mathrm{Prov}(u_j) \cup \mathrm{Peer}(u_j)$. Then, for all $1 \leq \ell < j$ it holds that $u_{\ell+1} \in \mathrm{Cust}(u_\ell) \cup \mathrm{Sibl}(u_\ell)$. Define $\ell^* =_{\mathrm{def}} \min\{\ \ell \mid \ell < j \text{ and } u_{\ell+1} \in \mathrm{Cust}(u_\ell)\ \}$. Thus, for all $\ell < \ell^* < j$ it holds that $u_{\ell+1} \in \mathrm{Sibl}(u_\ell)$. Hence, the path $(u_{\ell^*+1}, \ldots, u_k)$ is a provider or peer route for u_{ℓ^*}. Since $u_{\ell^*} \in \mathrm{Prov}(u_{\ell^*+1})$ the path $(u_{\ell^*+1}, \ldots, u_k)$ has not been exported to u_ℓ according to the Selective Export Rule. Consequently, $(u_1, \ldots, u_k)$ does not belong to $R(v)$. A contradiction. Thus, the assumption is false or we are in the second case. As similar arguments lead to a contradiction as well, we are done. ❑

6.3.2 Acyclicity

In the previous section, we have seen how some rational economic behavior implies valley-freeness of locally observable routes in our simple, abstract BGP. These routes reflect short-term behavior determined by routing policies based on commercial relationships. Commercial relationships typically are stable over a longer period and they impose a global structure on the connectivity graph independent of concrete BGP routes.

Next we summarize common knowledge on business relations between ASes to obtain a reasonable acyclicity structure within an AS graph. We do so by identifying patterns of oriented cycles which we will forbid to be contained in the graph. An oriented cycle can be interpreted as someone being its own provider and customer. In Table 6.1 and Figure 6.2, the 16 non-isomorphic triads of the 64 possible orientations of a complete graph on three vertices are shown. Table 6.1 lists 8 forbidden triads together with plausibility arguments why they are forbidden. The generalizations of the forbidden patterns are fairly obvious. Plausibility is based on size rules:

1. If AS u is a customer of AS v, then AS u has much smaller size (i.e., number of routers) than AS v (see, e.g., [68, 61, 140]). This is assumption **A** from this chapter's introductory section.
2. If AS u is a peering partner of AS v, then AS u and AS v are roughly of the same size (see, e.g., [117]. Moreover, we consider *roughly the same size* to be a transitive relation.

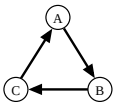
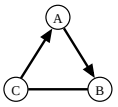
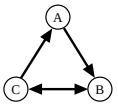
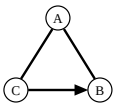
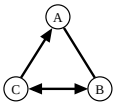
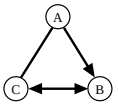
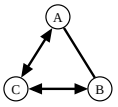
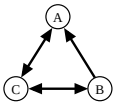
Triad	Plausibility	Pattern
	Typically, C is provider of B only if C is much larger than B, A is provider of C only if A is much larger than C. So, B is not much larger than A. A contradiction to B being provider of A.	$\rightarrow^*$
	A typical criterion for a peer-to-peer relation is roughly the same size or traffic. This does not hold if A is much larger than C and B is much larger than A.	$\{\rightarrow, -\}^* \rightarrow \{\rightarrow, -\}^*$
	Typically, as B and C are siblings, they behave like one AS. So, B and C together are not much larger and much smaller than A at the same time.	$\{\rightarrow, \leftrightarrow\}^* \rightarrow \{\rightarrow, \leftrightarrow\}^*$
	Typically, C and A have a peer-to-peer relation if they are roughly the same size. The same holds for A and B. So, B should not be much larger than C.	$\{\rightarrow, -\}^* \rightarrow \{\rightarrow, -\}^*$
	Typically, A and B have a peer-to-peer relation if A and B together with its sibling C have roughly the same size. So, A is not much larger than C together with its sibling B.	$\{\rightarrow, \leftrightarrow, -\}^* \rightarrow \{\rightarrow, \leftrightarrow, -\}^*$
	Typically, A and C have a peer-to-peer relation if A and C together with its sibling B have roughly the same size. So, A is not much smaller than B together with its sibling C.	$\{\rightarrow, \leftrightarrow, -\}^* \rightarrow \{\rightarrow, \leftrightarrow, -\}^*$
	Due to the transitivity of the sibling-to-sibling relation, A and B are siblings. So, typically, A and B do not have a peculiar peer-to-peer relation. Note that larger cycles with siblings and at least two peer-to-peer relations make sense.	$\leftrightarrow^* - \leftrightarrow^*$
	Due to the transitivity of the sibling-to-sibling relation, A and B are siblings. So, A together with its siblings B and C is not much larger than C together with its siblings B and A.	$\{\rightarrow, \leftrightarrow\}^* \rightarrow \{\rightarrow, \leftrightarrow\}^*$

Table 6.1. Forbidden triads and their generalized forbidden patterns.

3. If AS u and AS v are siblings, then they count as one AS, i.e., we assume that the size of u is determined by its own number of routers and the number of routers of all its sibling ASes.

Each of these rules may have its exceptions but they certainly describe typical behavior. We idealize the BGP world by assuming that all contracts follow these rules. The union of all generalized forbidden patterns given in Table 6.1 leads to the following definition of an oriented cycle.

Definition 6.3. Let (G, φ) be an oriented graph. Let C be any simple cycle of G. C is said to be an *oriented cycle* of (G, φ) if and only if $\varphi(C)$ belongs to

$$\{-, \leftrightarrow\}^* \rightarrow \{\rightarrow, -, \leftrightarrow\}^* \quad \cup \quad \{-, \leftrightarrow\}^* \leftarrow \{\leftarrow, -, \leftrightarrow\}^* \quad \cup \quad \leftrightarrow^* - \leftrightarrow^* .$$

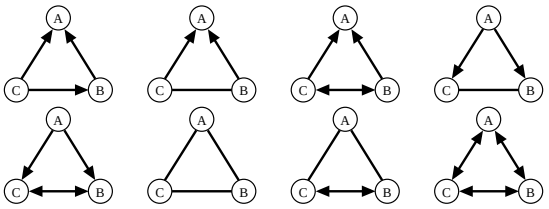


Figure 6.2. Allowed triads

The simplicity of cycles is required since we exactly count occurrences of peer-to-peer edges in oriented cycles. To complement our view on forbidden triads, Figure 6.2 shows the 8 allowed triads.

Note that in the case that φ does not exhaust the full type set $\{\rightarrow, —, \leftrightarrow\}$, the patterns of oriented cycles simplify. For instance, if the type set is $\{\rightarrow\}$, then we obtain that a minimal cycle C is an oriented cycle if and only if $\varphi(C)$ belongs to $\rightarrow^*$ or $\leftarrow^*$ which is the usual understanding of a cycle. As a second example, if the type set is $\{\leftrightarrow, —\}$, then a minimal cycle C is an oriented cycle if and only if $\varphi(C)$ belongs to $\leftrightarrow^* — \leftrightarrow^*$.

We call an orientation *acyclic* if it contains no oriented cycles. In the forthcoming we will need fast algorithms for testing acyclicity which are all based on standard techniques (see, e.g., [35]).

Lemma 6.4. *Let K be any subset of $\{\rightarrow, —, \leftrightarrow\}$. Testing whether a given graph, with n vertices and m edges, which is oriented with type set K is acyclic can be done in time $O(n+m)$.*

Proof. We briefly describe algorithms for all cases individually:

- The cases where the type set is either $\{—\}$ or $\{\leftrightarrow\}$ are trivial, as graphs oriented in these ways are always acyclic.
- Acyclicity for type set $\{\rightarrow\}$ can be tested by topological sorting.
- For type set $\{\rightarrow, \leftrightarrow\}$, replace each sibling-to-sibling edge $u \leftrightarrow v$ with two edges $u \rightarrow v$ and $v \rightarrow u$. Compute the strongly connected components and test whether each component only contains sibling-to-sibling edges in the original orientation. This can be done in time $O(n + m)$.
- Similarly, for type set $\{\rightarrow, —\}$, replace a peer-to-peer edge $u—v$ with two edges $u \rightarrow v$ and $v \rightarrow u$. Compute the strongly connected components and test whether each component only contains peer-to-peer edges in the original orientation. Again, this can be done in time $O(n + m)$.
- For $\{—, \leftrightarrow\}$, compute the simply connected components with respect to the $\leftrightarrow$ edges and check for each oriented edge $u—v$ whether u and v belong to different components. If for some edge this is not true, we have learned that the graph is not acyclic. This can be done in time $O(n + m)$.
- Given the full type set $\{\rightarrow, —, \leftrightarrow\}$ we first test acyclicity of the oriented graph with type set $\{—, \leftrightarrow\}$ induced by ignoring customer-to-provider edges. If the graph passes the test then we replace in the original orientation $\leftrightarrow$ and $—$ edges with directed edges of opposite directions, compute the strongly connected components, and check for each $u \rightarrow v$ or $u \leftarrow v$ whether u and v belong to different components. If for some edge this is not true, the graph is not acyclic. Clearly, this can be done in time $O(n + m)$.

This completes the proof of the lemma. ❑

6.4 The Acyclic Type-of-Relationship problem

The central problem we are concerned with in this section is the Acyclic Type-of-Relationship problem: given any set of AS paths is there an orientation such that all AS paths are indeed valley-free and the induced AS graph has the additional property of being acyclic? We study the problem under the viewpoint of its computational complexity.

6.4.1 Definition

A very general, purely combinatorial formulation as a decision problem is the following. Let K and U be two type sets.

Problem:	ACYCLIC TOR(K, U)
Input:	An undirected graph G, a path set P, and a partial orientation φ given by an edge set R with labels from K
Question:	Is there a completion of the orientation φ using only edge types from U such that the completed orientation, with respect to type set $K \cup U$, contains no cycle and all oriented paths in P are valley-free?

Of course, in case that the answer is "Yes" for an input we require the answer to be witnessed by an appropriate orientation.

We briefly discuss two interpretations how the problem might be used in practical settings:

- Applied to a real-world scenario, G is the AS graph, P stands for the set of observed AS paths, e.g., gathered at certain observation points. The set R is an explicit pre-knowledge we have of certain relations between two ASes, as observable from several resources on the Internet. The task is to find a hypothetical valley-free and acyclic orientation in accordance with our pre-knowledge.
- In a test setting, G is a BGP-world model, P could be the set of information made available to BGP speakers, and R could describe a specified situation with types from K. Here, we want to find an orientation that guarantees valley-freeness and acyclicity and which does not destroy the given specification. For this purpose, an appropriate choice of a type set U is useful. The most important case, of course, is $U = \{\rightarrow\}$.

The size of the input is always denoted by N, i.e., $N = \|V(G)\| + \|E(G)\| + |P| + |R|$ where $|P|$ and $|R|$ are the sums of the path lengths in P and in R. The length of a path with k edges is the sum of the lengths of the $k + 1$ vertex descriptions.

Since many of our algorithms run in linear time, we should say a word about the representation of instances. For the sake of simplicity we assume that all vertices and edges of G actually appear in P, i.e., $G = G(P) = (V(P), E(P))$ where $V(P)$ denotes the set of vertices appearing in P and $E(P)$ denotes the set of edges appearing in P. Thus, $N = O(|P|)$. We use $V(p)$ and $E(p)$ to denote $V(\{p\})$ and $E(\{p\})$. We suppose that an instance is given as an adjacency list of an AS graph $G(P)$ with vertex set $V(P)$ and edge set $E(P)$, where edges

from R are already labeled according to the partial orientation. Moreover, we assume that the path set is represented as a collection of lists having cross-links to the corresponding edges in the AS graph and vice versa. Orientations are stored with the edges in $G(P)$. Since there are no more vertices and edges in $G(P)$ than appearing in P, this guarantees that we can always test valley-freeness in time $O(N)$.

6.4.2 Complexity results

The computational complexity of the Acyclic Type-of-Relationship problem depends on the types of allowed orientations.

	Type U						
Type K	$\{\rightarrow\}$	$\{-\}$	$\{\leftrightarrow\}$	$\{\rightarrow,-\}$	$\{\rightarrow,\leftrightarrow\}$	$\{-,\leftrightarrow\}$	$\{\rightarrow,-,\leftrightarrow\}$
$\emptyset$	$O(N)$	$O(N)$	$O(N)$	$O(N)$	$O(N)$	$O(N)$	$O(N)$
$\{\rightarrow\}$	$O(N)$	$O(N)$	$O(N)$	$O(N)$	NP-complete	$O(N)$	NP-complete
$\{-\}$	$O(N)$	$O(N)$	$O(N)$	$O(N)$	NP-complete	NP-complete	NP-complete
$\{\leftrightarrow\}$	$O(N)$	$O(N)$	$O(N)$	$O(N)$	$O(N)$	$O(N)$	$O(N)$
$\{\rightarrow,-\}$	$O(N)$	$O(N)$	$O(N)$	$O(N)$	NP-complete	NP-complete	NP-complete
$\{\rightarrow,\leftrightarrow\}$	$O(N)$	$O(N)$	$O(N)$	$O(N)$	NP-complete	$O(N)$	NP-complete
$\{-,\leftrightarrow\}$	$O(N)$	$O(N)$	$O(N)$	$O(N)$	NP-complete	NP-complete	NP-complete
$\{\rightarrow,-,\leftrightarrow\}$	$O(N)$	$O(N)$	$O(N)$	$O(N)$	NP-complete	NP-complete	NP-complete

Table 6.2. Complexity classification of Acyclic ToR(K,U) for $K,U \subseteq \{\rightarrow,-,\leftrightarrow\}$.

Table 6.2 shows a complete complexity classification depending on specified edge-type sets. The basic (and typically considered) combinatorial problem appears as case $K = \emptyset$, i.e., where we have no pre-knowledge. The most remarkable results within the classification are:

- Acyclic ToR$(\{\rightarrow,\leftrightarrow,-\},\{\rightarrow\})$ can be solved in time $O(N)$. This result underlines that the partialness-to-entireness methodology of [165], aside from producing more and more realistic AS relationships, is a very feasible one.
- Acyclic ToR$(\{\rightarrow\},\{\rightarrow,\leftrightarrow\})$ is NP-hard. This is surprising since there is a linear-time algorithm for Acyclic ToR$(\{\rightarrow\},\{\rightarrow\})$ and $\leftrightarrow$ very often is a negligible type of relationship.

Observe that in Table 6.2 either there is a linear-time algorithm for a problem or the problem is NP-complete. Hence, the classification is optimal.

The remaining parts of this section are devoted to a proof of this classification. The complexities of the single problems can be related according to the following obvious proposition.

Proposition 6.5. *Let $K,K',U \subseteq \{\rightarrow,\leftrightarrow,-\}$ be any type sets. Suppose $K \subseteq K'$. Then the following statements are true:*

1. *If* Acyclic ToR(K',U) *is solvable in time $O(N)$ then* Acyclic ToR(K,U) *is solvable in time $O(N)$.*
2. *If* Acyclic ToR(K,U) *is* NP*-complete then* Acyclic ToR(K',U) *is* NP*-complete.*

From Lemma 6.4 we immediately obtain the entries for the second and third column of our classification.

Proposition 6.6. *Let $K \subseteq \{\rightarrow, \leftrightarrow, —\}$ be any type set. Then,* Acyclic ToR$(K, \{—\})$ *and* Acyclic ToR$(K, \{\leftrightarrow\})$ *can be solved in time $O(N)$.*

To further reduce the number of results to be proven, in the next subsection we will see that peer-to-peer relations can essentially be disregarded as options for completions. Subsections 6.4.4–6.4.6 then contain the remaining algorithms and completeness results.

6.4.3 Handling peer-to-peer relations

To handle peer-to-peer edges in both type sets K and U, we first show that in an acyclic and valley-free orientation they exhibit a very simple structure, namely they induce a partial ordering among vertex sets. To see this, it is useful to introduce some further notation.

Suppose we are given a partial orientation φ of a graph $G = (V, E)$ using the type set $\{\rightarrow, —\}$. We say that a vertex $v \in V$ is *P2P-reachable* from a vertex u in (G, φ) if there exists a path $p = (w_0, w_1, \ldots, w_k)$ in G such that $w_0 = u$, $w_k = v$, and for all $i \in \{1, \ldots, k\}$, it holds that $\varphi(w_{i-1}, w_i)$ is defined and $\varphi(w_{i-1}, w_i) = —$. Note that P2P-reachability is a reflexive, symmetric, and transitive relation; hence, an equivalence relation. The vertex set thus can be divided into equivalence classes which are called *P2P-components*.

Let $p = (w_0, \ldots, w_k)$ be a path of any path set P. We define a set-mapping $\xi[p]$ which for any vertex set $A \subseteq V(P)$ collects all positions in p that lie between two positions of vertices of A. More specifically, we define $\xi[p] : \mathcal{P}(V(P)) \rightarrow \mathcal{P}(\mathbb{N})$ for $A \subseteq V(P)$ by

$$\xi[p](A) =_{\text{def}} \{\ i \mid \text{ there are } \ell \leq i \leq r \text{ such that } [w_\ell \in A \wedge w_r \in A]\ \}.$$

Notice that if $A \cap V(p) \neq \emptyset$, then $\min \xi[p](A)$ and $\max \xi[p](A)$ are positions where vertices from A occur.

An important position in a path is the turning point. For any valley-free orientation φ of a graph $G(P)$ for a path set P (we also say orientation of P), the *turning point* of a path $p = (w_0, \ldots, w_k) \in P$ is defined to be the maximal position $0 \leq i \leq k$ such that $\varphi(w_{i-1}, w_i) = \rightarrow$. If no such position exists, then the turning point is defined to be 0. If the orientation is additionally acyclic, then we can isolate the range of positions within paths where turning points occur. Observe that generally, in order to have an acyclic and valley-free orientation, in each path that contains vertices s and t P2P-reachable from each other and that are not neighbors, the orientation has to point from s in direction of t and from t in direction of s. Otherwise we would find an oriented cycle. An easy consequence is the following proposition.

Proposition 6.7. *Let P be a path set. Suppose we are given an acyclic and valley-free orientation of P using $\{\rightarrow, —\}$. Let x and y be distinct vertices in the same P2P-component. For each path $p \in P$ containing both vertices x and y, the turning point of p belongs to $\xi[p](\{x, y\})$.*

Example 6.8. Figure 6.3 shows an example of an AS graph and a path set together with an acyclic and valley-free completion using customer-to-provider and peer-to-peer edges. Acyclicity is easily seen from oriented graph on the left-hand side and valley-freeness is witnessed by

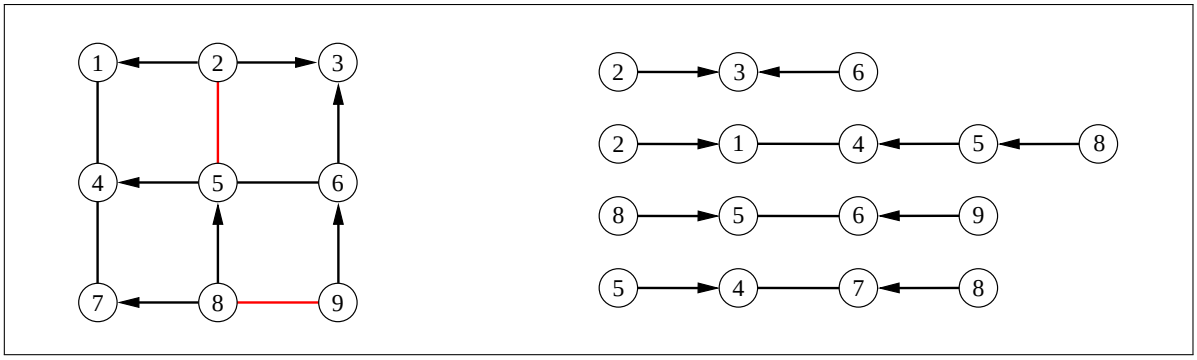


Figure 6.3. Example of an acyclic and valley-free completion using peer-to-peer edges

the set of oriented paths on the right-hand side. The red peer-to-peer edges are given. The P2P-components are $\{1,4,7\}$, $\{2,5,6\}$, $\{8,9\}$, and $\{3\}$. Table 6.3 contains the values of the set-mapping ξ for all P2P-components and all paths. Note that the first vertex in a path has index 0. The turning point in all paths is 1.

Component A	$\xi[(\mathbf{2},\mathbf{3},\mathbf{6})](\mathbf{A})$	$\xi[(\mathbf{2},\mathbf{1},\mathbf{4},\mathbf{5},\mathbf{8})](\mathbf{A})$	$\xi[(\mathbf{8},\mathbf{5},\mathbf{6},\mathbf{9})](\mathbf{A})$	$\xi[(\mathbf{5},\mathbf{4},\mathbf{7},\mathbf{8})](\mathbf{A})$
$\{1,4,7\}$	$\emptyset$	$\{1,2\}$	$\emptyset$	$\{1,2\}$
$\{2,5,6\}$	$\{0,1,2\}$	$\{0,1,2,3\}$	$\{1,2\}$	$\{0\}$
$\{8,9\}$	$\emptyset$	$\{4\}$	$\{0,1,2,3\}$	$\{3\}$
$\{3\}$	$\{1\}$	$\emptyset$	$\emptyset$	$\emptyset$

Table 6.3. The image of ξ.

The following lemma shows that in an acyclic and valley-free orientation, all pairs of vertices from the same P2P-component form a total ordering within each path.

Lemma 6.9. *Let P be a path set. Suppose we are given an acyclic and valley-free orientation using $\{\rightarrow,-\!\!-\}$. Let $\{u,v\}$ and $\{x,y\}$ be two pairs of vertices such that $u \neq v$, $x \neq y$, $\{u,v\} \neq \{x,y\}$, v is P2P-reachable from u, and y is P2P-reachable from x. For all paths $p \in P$, if $\{u,v,x,y\} \subseteq V(p)$, then*

1. *$\{u,v\} \cap \{x,y\} = \emptyset$ and*
2. *$\xi[p](\{u,v\}) \subseteq \xi[p](\{x,y\})$ or $\xi[p](\{x,y\}) \subseteq \xi[p](\{u,v\})$.*

Proof. Suppose we have an acyclic and valley-free orientation of P using $\{-\!\!-,\rightarrow\}$. First, let $p \in P$ be any path which does not contain a peer-to-peer edge (in particular, neither u and v are neighbors in p nor are x and y) and let both pairs $\{u,v\}$ and $\{x,y\}$ be contained in p. From Proposition 6.7 we obtain that the turning point belongs to $\xi[p](\{u,v\}) \cap \xi[p](\{x,y\})$. This excludes the case that $\xi[p](\{u,v\})$ and $\xi[p](\{x,y\})$ have at most one common element. Without loss of generality, assume that $\min \xi[p](\{u,v\}) \leq \min \xi[p](\{x,y\})$. Consider the vertex sequence

$$(w_{\min \xi[p](\{u,v\})}, \ldots, w_{\min \xi[p](\{x,y\})}, w_{\max \xi[p](\{x,y\})}, \ldots, w_{\max \xi[p](\{u,v\})}, w_{\min \xi[p](\{u,v\})}).$$

For this sequence not to be an oriented cycle in $G(P)$, it must hold that

$$\min \xi[p](\{u,v\}) < \min \xi[p](\{x,y\}) \quad \text{and} \quad \max \xi[p](\{x,y\}) < \max \xi[p](\{u,v\}).$$

Recall that the vertices at positions $\min \xi[p](\{u,v\})$ and $\max \xi[p](\{u,v\})$ can be connected in $G(P)$ by a path totally consisting of peer-to-peer edges. Equally, the vertices at $\min \xi[p](\{x,y\})$ and $\max \xi[p](\{x,y\})$ can be bridged by such a path. The case of p having a peer-to-peer edge can be treated in a similar way. The statement of the lemma follows. ❑

Note that Lemma 6.9 also shows that in acyclic and valley-free orientations, paths contain at most two vertices from an equivalence class according to P2P-reachability. In addition to Lemma 6.9, P2P-components show on all paths the same total ordering.

Lemma 6.10. *Let P be a path set. Suppose we are given an acyclic and valley-free orientation using $\{\rightarrow, —\}$. Let A and B be vertex sets in distinct P2P-components. If there is a path $p \in P$ such that $\|A \cap V(p)\| \geq 2$, $\|B \cap V(p)\| \geq 2$ and $\xi[p](A) \subseteq \xi[p](B)$, then for each path $q \in P$ satisfying $\|A \cap V(q)\| \geq 2$ and $\|B \cap V(q)\| \geq 2$, it holds that $\xi[q](A) \subseteq \xi[q](B)$.*

Proof. Let P be a path set which allows an acyclic and valley-free orientation φ of $G(P)$ using $\{\rightarrow, —\}$. Let A and B be contained in two different equivalence classes with respect to P2P-reachability. Let $p \in P$ be a path which contains some distinct vertices $x_A, y_A \in A$ and some distinct vertices $x_B, y_B \in B$ such that $\xi[p](\{x_A, y_A\}) \subseteq \xi[p](\{x_B, y_B\})$. We may assume that x_A occurs before y_A in p and x_B occurs before y_B in q. Assume to the contrary that there is a path $q \in P$ containing distinct vertices $u_A, v_A \in A$ and $u_B, v_B \in B$ such that $\xi[q](\{u_A, v_A\}) \not\subseteq \xi[q](\{u_B, v_B\})$. From Lemma 6.9, we obtain $\xi[q](\{u_B, v_B\}) \subseteq \xi[q](\{u_A, v_A\})$. Here, we also may assume that u_B occurs before v_B in q and u_A occurs before v_A in q. By Proposition 6.7, the turning position of p belongs to $\xi[p](A)$ and the turning position of q belongs to $\xi[q](B)$. It follows that $G(P)$ contains an oriented cycle that can be built by a path from u_A to u_B belonging to $\rightarrow^*$ (as in q), followed by a path from u_B to x_B belonging to $—^*$, followed by a path from x_B to x_A belonging to $\rightarrow^*$ (as in p), and followed by a path back from x_A to u_A belonging to $—^*$. A contradiction. ❑

We use the lemma to define an auxiliary directed graph $H(\varphi, P)$, depending on the path set P and an acyclic and valley-free orientation φ of P with type set $\{\rightarrow, —\}$, as follows:

$$V(H(\varphi,P)) \;=_{\text{def}}\; \{\, A \mid \emptyset \neq A \subseteq V(P) \text{ is contained in a P2P-component (with respect to } \varphi) \,\}$$

$$E(H(\varphi,P)) \;=_{\text{def}}\; \{\, (A,B) \mid (\exists p \in P)[\, \xi[p](A) \neq \emptyset \;\wedge\; \xi[p](A) \subseteq \xi[p](B) \,] \,\}$$

By Lemma 6.10, this graph is always well-defined. Moreover, the graph is a directed acyclic graph since it describes a partial ordering of the vertex subsets.

Example 6.11. The auxiliary graph for the oriented path set (together with the two given peer-to-peer edges $(2,5)$ and $(8,9)$) from Example 6.8 is shown in Figure 6.4. The structure within the P2P-components follows easily from the fact that $\xi[p](A) \subseteq \xi[p](B)$ whenever $A \subseteq B$. Moreover, we have

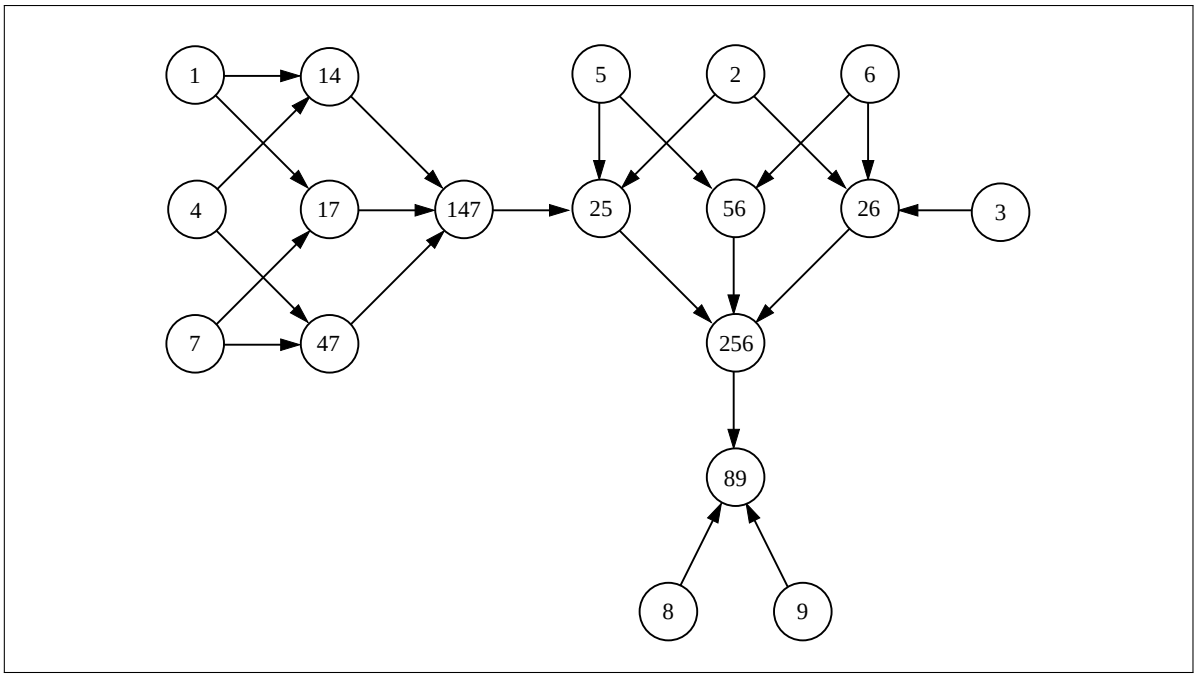


Figure 6.4. Example of an auxiliary graph

$$\begin{aligned} \xi[(2,1,4,5,8)](\{1,4,7\}) &\subseteq \xi[(2,1,4,5,8)](\{2,5\}) \\ \xi[(2,3,6)](\{3\}) &\subseteq \xi[(2,3,6)](\{2,6\}) \\ \xi[(8,5,6,9)](\{2,5,6\}) &\subseteq \xi[(8,5,6,9)](\{8,9\}) \end{aligned}$$

This implies the remaining three edges.

We will use this auxiliary graph as a structural tool to see that under certain circumstances we can disregard peer-to-peer edges for finding acyclic and valley-free completions. Note that the graph can have a size exponential in the size of the path set. However, since we do not compute the graph this is not problematic.

Definition 6.12. Let P be a path set and let φ be a partial orientation of P using —. A P2P-component A is said to be *φ-complete* if and only if for all edges $\{u,v\} \in E(P)$, if both u and v belong to A then $\varphi(u,v) = —$.

Notice that an edge $\{u,v\}$ of a P2P-component cannot be oriented as a customer-to-provider edge if we want to guarantee acyclicity.

Lemma 6.13. *Let P be an arbitrary path set. Let φ be a partial orientation of P with type set $\{\rightarrow, —\}$ such that all P2P-components are φ-complete. There is an acyclic and valley-free completion of φ on P with type set $\{\rightarrow, —\}$ if and only if there is an acyclic and valley-free completion of φ on P with type set $\{\rightarrow\}$.*

Proof. The direction ($\Leftarrow$) holds trivially.

For ($\Rightarrow$), suppose $\hat{\varphi}$ is an acyclic and valley-free completion of φ on P with type set $\{\rightarrow, —\}$. We show that each peer-to-peer edge introduced by the completion can be changed into a customer-to-provider or a provider-to-customer edge without introducing oriented cycles

while keeping paths valley-free. Let us consider the auxiliary graph $H(\hat{\varphi}, P)$. Suppose the vertex set $\{A'_1, \ldots, A'_r\}$ of $H(\hat{\varphi}, P)$ is topologically sorted, i.e., if (A'_i, A'_j) is an edge in $H(\hat{\varphi}, P)$ then $i < j$. Let $A_1, \ldots, A_k$ be the P2P-components of φ. Since $\hat{\varphi}$ is a completion of φ the sets $A_1, \ldots, A_k$ appear as vertices in the auxiliary graph $H(\hat{\varphi}, P)$. Thus, we may assume that $A_1, \ldots, A_k$ are enumerated in the order they appear in the topological order of $H(\hat{\varphi}, P)$. We define an orientation ψ, for all $u, v \in V(P)$, as follows:

$$\psi(u,v) =_{\text{def}} \begin{cases} \hat{\varphi}(u,v) & \text{if } \hat{\varphi}(u,v) \neq - \text{ or } u \text{ and } v \text{ lie in one P2P-component of } \varphi \\ \rightarrow & \text{if } \hat{\varphi}(u,v) = - \text{ and } u \in A_i, v \in A_j \text{ with } i < j \\ \leftarrow & \text{if } \hat{\varphi}(u,v) = - \text{ and } u \in A_i, v \in A_j \text{ with } j < i \end{cases}$$

Since all P2P-components are φ-complete, ψ is a total function and thus, ψ is a completion of φ. We have to prove that ψ is acyclic and valley-free. We examine both criteria separately.

1. *Valley-freeness.* Let e be an edge in the graph $G(P)$ such that $\hat{\varphi}(e) = -$. Since $\hat{\varphi}$ is a valley-free orientation, any path p of P that contains e must have an orientation of type $\rightarrow^* - \leftarrow^*$. Orienting e either as $\rightarrow$ or as $\leftarrow$ satisfies valley-freeness. So does ψ.
2. *Acyclicity.* Let $e = \{u, v\}$ be any edge of $G(P)$ such that $\hat{\varphi}(e) = -$ and the vertices u, v do not belong to the same P2P-component (with respect to φ). Let C be any cycle containing e as its last edge. Recall from Definition 6.3 that an oriented cycle using type set $\{\rightarrow, -\}$ belongs to

$$-^* \rightarrow \{\rightarrow, -\}^* \ \cup \ -^* \leftarrow \{\leftarrow, -\}^*.$$

   As $\hat{\varphi}(C)$ is an acyclic orientation, i.e., $\hat{\varphi}(C)$ does not show the just-mentioned pattern, we have the following three cases to consider:
   a) $\hat{\varphi}(C)$ contains an edge oriented as $\rightarrow$ before coming an edge oriented by $\leftarrow$ and ending with the edge e oriented as $-$. Clearly, orienting e in the way ψ does does not introduce a cycle.
   b) $\hat{\varphi}(C)$ contains an edge oriented as $\leftarrow$ before coming an edge oriented by $\rightarrow$ and ending with the edge e oriented as $-$. This is similar to the case above. Thus, orienting e in the way ψ does does not introduce a cycle.
   c) $\hat{\varphi}(C)$ completely consists of peer-to-peer edges. Note that a cycle has at least three edges. Since u and v belong to different P2P-components there must exist further edges in C with vertices from different P2P-components (with respect to φ). Since the P2P-components (with respect to φ) are totally ordered by natural numbers, ψ does not introduce a cycle.

This completes the proof of the lemma. ❑

Example 6.14. Figure 6.5 shows how the construction from Lemma 6.13 helps to get rid of peer-to-peer edges in our completion in Example 6.8. There, the partial orientation φ was given by the two peer-to-peer edges $(2, 5)$ and $(8, 9)$. By a possible topological ordering of the auxiliary graph in Figure 6.4 the P2P-components of φ could be enumerated as $\{1\}, \{3\}, \{4\}, \{6\}, \{7\}, \{2, 5\}$, and $\{8, 9\}$. Note that all P2P-components are φ-complete. From the contructions we obtain that the edge $\{1, 4\}$ is oriented from 1 to 4, the edge $\{4, 7\}$ is oriented towards 7, and the edge $\{5, 6\}$ is oriented from 6 to 5. Thus, the new orientation is an acyclic and valley-free orientation of φ only using customer-to-provider edges for completions.

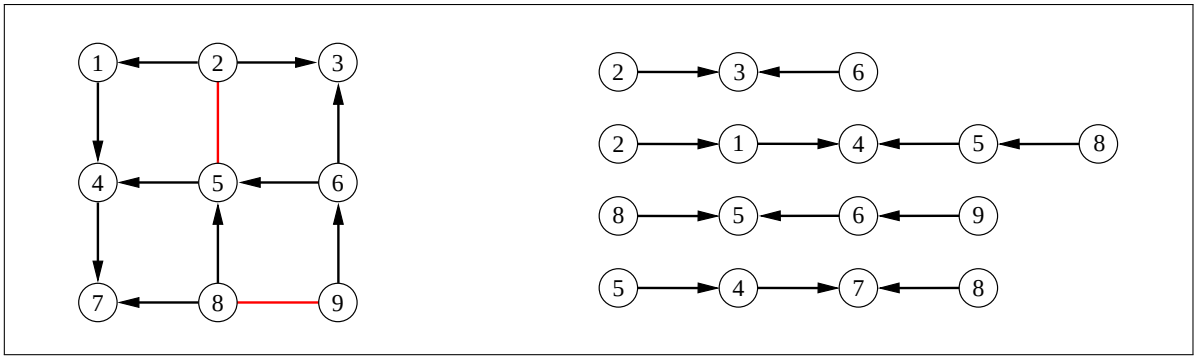


Figure 6.5. Example of an acyclic and valley-free completion without peer-to-peer edges

Lemma 6.13 is a twofold generalization of a proposition in [40] which states that for any path set P, there exists a valley-free orientation with type set $\{\rightarrow, —\}$ if and only if there exists a valley-free orientation using only $\{\rightarrow\}$. First, it proves the same property for additionally acyclic orientations. Second, it proves this property independently of the initially given partial orientation. Note that in respect thereof, the standard case of [40] appears as the nowhere-defined partial orientation.

6.4.4 Completing with customer-to-provider relations

In this subsection we will see how we can compute, in linear time, an acyclic and valley-free orientation no matter of which kind of information our pre-knowledge is. That is, the goal is to show that Acyclic ToR$(\{\rightarrow, \leftrightarrow, —\}, \{\rightarrow\})$ is solvable in linear time (see Theorem 6.22). For the sake of clarity, it is reasonable to approach this theorem over several intermediate stages.

We start with the very base case $K = \emptyset$. For obtaining a linear time algorithm, the crucial observation is that each vertex appearing somewhere in the middle of a path has in-degree at least one, valley-freeness supposed. This allows us to employ a topological-sort approach.

Theorem 6.15. Acyclic ToR$(\emptyset, \{\rightarrow\})$ *can be solved in time* $O(N)$.

Proof. Suppose we are given a path set P. Let v be a vertex such that, for each path in P, if v lies on p then v is an endpoint of the path. If $G(P)$ can be acyclically oriented such that all paths are oriented valley-free then such a vertex v must exists: for any acyclic orientation of $G(P)$ there must exist at least one vertex u such that all edges $\{u, w\}$ are oriented as $u \rightarrow w$. Since all paths are valley-free, u cannot be in the middle of any path because that would result in an orientation containing $\leftarrow\rightarrow$. Therefore, one vertex v can be forced.

We iteratively reduce the problem by removing such vertices v for all path-ends and orienting the removed edges away from v as $v \rightarrow w$ for all neigbors w in P. (Note that P has changed.) Assume a reduced path $(v_0, \ldots, v_m)$ is oriented as $\rightarrow^*\leftarrow^*$, then adding v to any end with the edge u, v_1 or u, v_m oriented as $\rightarrow$ results in a valley-free orientation. Furthermore, if the reduced graph is acyclic then the graph with v added is also acyclic.

A precise description is given as Algorithm 2. Clearly, this algorithm can be implemented in such a way that its running time is $O(|P|)$. ❑

Algorithm 2: Linear-time algorithm for Acyclic ToR($\emptyset, \{\rightarrow\}$).

Input: Undirected graph G, path set P
Output: Acyclic and valley-free orientation of the induced graph $G(P)$, if it exists, or indication that it does not exist

```
1  foreach vertex v ∈ V(P) do
2  |  count(v) := 0
3  end
4  foreach p ∈ P do
5  |  foreach vertex v ∈ p do
6  |  |  if v is not an endnode of p then
7  |  |  |  count(v) := count(v) + 1
8  |  |  end
9  |  end
10 end
11 U := ∅
12 foreach vertex v ∈ V(P) do
13 |  if count(v) = 0 then
14 |  |  U := U ∪ {v}
15 |  end
16 end
17 V' := ∅
18 while U ≠ ∅ do
19 |  remove a vertex u from U
20 |  foreach vertex v such that v ∈ V \ V' and {u, v} ∈ E(P) do
21 |  |  orient {u, v} as u → v
22 |  |  foreach p ∈ P such that u and v are neighbors in p do
23 |  |  |  if v has a neighbor w in p on the side opposite to u such that w ∈ V \ V' then
24 |  |  |  |  count(v) := count(v) − 1
25 |  |  |  end
26 |  |  |  if count(v) = 0 then
27 |  |  |  |  U := U ∪ {v}
28 |  |  |  end
29 |  |  end
30 |  end
31 |  V' := V' ∪ {u}
32 end
33 if V' ≠ V then
34 |  return path set P fails to allow an acyclic and valley-free orientation
35 end
```

The basic Algorithm 2 described in Theorem 6.15 can be extended by additional linear-time preprocessing phases to handle non-trivial pre-knowledge.

Theorem 6.16. Acyclic ToR($\{\rightarrow\}, \{\rightarrow\}$) *can be solved in time* $O(N)$.

Proof. We modify Algorithm 2 appropriately. Let $\perp$ be a vertex which is neither in path set P nor in edge set R. For each oriented edge $u \rightarrow v$ of R, add a path $(u, v, \perp)$ to the path set. Let P' be the resulting path set. Now apply Algorithm 2 on path set P' where $\perp$ is not considered as a vertex of $V(P')$. Clearly, this algorithm solves the problem correctly (by an argumentation similar to Theorem 6.15) and runs in time $O(N)$. ❑

For the case $K = \{\leftrightarrow\}$ we need some more notation. Let P be a path set and let φ be a partial orientation of $G(P)$ given by a set R of sibling-to-sibling edges in $E(P)$. We define a vertex set $U \subseteq V(P)$ to be an *S2S-component* if for all vertices $u, v \in U$, there exists a path in (P, φ) from u to v consisting only of sibling-to-sibling edges, and U is maximal subject

to this property. Clearly, for any given set R the vertex set $V(P)$ can be divided into S2S-components which can be easily computed in linear time. We say that an S2S-component U is *S2S-complete* if for all $u, v \in U$, the edge $\{u, v\} \in E(P)$ belongs to R, i.e., $\varphi(u, v)$ is defined and $\varphi(u, v) = \leftrightarrow$. We say that an S2S-component U is *S2S-loopless* if for each path $p \in P$, there does not occur any vertex not belonging to U between two vertices from U on the path p. An S2S-component is *S2S-closed* if it is both S2S-loopless and S2S-complete. It is moreover obvious that completeness, looplessness, and closeness can easily be tested in linear time.

Theorem 6.17. Acyclic ToR$(\{\leftrightarrow\}, \{\rightarrow\})$ *can be solved in time* $O(N)$.

Proof. As we are not allowed to introduce new sibling-to-sibling edges, we only have to check whether the set R of sibling-to-sibling edges induces an S2S-closed vertex set (meaning the set $V(R)$ of vertices occurring in R). If not reject the instance. Otherwise replace all occurrences of vertices from an S2S-component A with a representative vertex and remove consecutive occurrences of same vertices in all paths. Now we have an Acyclic ToR$(\emptyset, \{\rightarrow\})$ instance which can be solved in time $O(N)$ by Theorem 6.15. Note that computing the reduced instance only needs $O(N)$ preprocessing time. ❑

Corollary 6.18. Acyclic ToR$(\{\rightarrow, \leftrightarrow\}, \{\rightarrow\})$ *can be solved in time* $O(N)$.

Proof. Observe that the preprocessing phase of the algorithm described in the proof of Theorem 6.17 does not dependent on edges oriented with types other than $\{\leftrightarrow\}$. Thus, after this $O(N)$ preprocessing step we have to solve an Acyclic ToR$(\{\rightarrow\}, \{\rightarrow\})$ instance which is just Theorem 6.16. ❑

The case $K = \{-\!\!-\}$ is more complex. Recall from Lemma 6.9 that in acyclic and valley-free orientations, pairs of P2P-reachable vertices build an inclusion chain within each path.

Theorem 6.19. Acyclic ToR$(\{-\!\!-\}, \{\rightarrow\})$ *can be solved in time* $O(N)$.

Proof. For a given instance (G, P, R), let $S \subseteq P$ be the set of paths containing a peer-to-peer edge from R. Let $G_R(P)$ denote the graph induced by P which only contains the peer-to-peer edges from R. The algorithm has two phases.

In the first, structure-testing phase of the algorithm we do the following:

1. Check for each path of S whether there is at most one peer-to-peer edge and, if there is one, orient all edges towards that edge. If the check fails then return an indication and stop.
2. Next check for each path $p \in P \setminus S$ whether for all pairs $\{u, v\} \subseteq V(p)$ such that v is P2P-reachable from u in $G_R(P)$, the sets $\xi[p](u, v)$ (introduced in the preceding subsection) form an inclusion chain according to Lemma 6.9 and compute the vertex pair e^p such that $\xi[p](e^p)$ is minimal with respect to set-inclusion. Normally, this would imply a worst-case time $O(|P \setminus S|^2 \cdot |R|) = O(N^3)$ (with a linear-time preprocessing step to compute the connected components in $G_R(P)$). However, we can do better to stay within $O(N)$. We simply compute some pair of vertices with minimal distance of their positions in p (using buckets for each P2P-component) and take this pair as e^p. We postpone the test of the chain-condition until we have to test for acyclicity and valley-freeness anyway.
3. For each path $p \in P \setminus S$ and its corresponding unique minimal pair $e^p = \{u, v\}$, if it exists, orient both the edge going from u in direction of v and the edge going from v in direction of u as out-going edges. Orient all edges outside of $\xi[p](e^p)$ towards e^p.

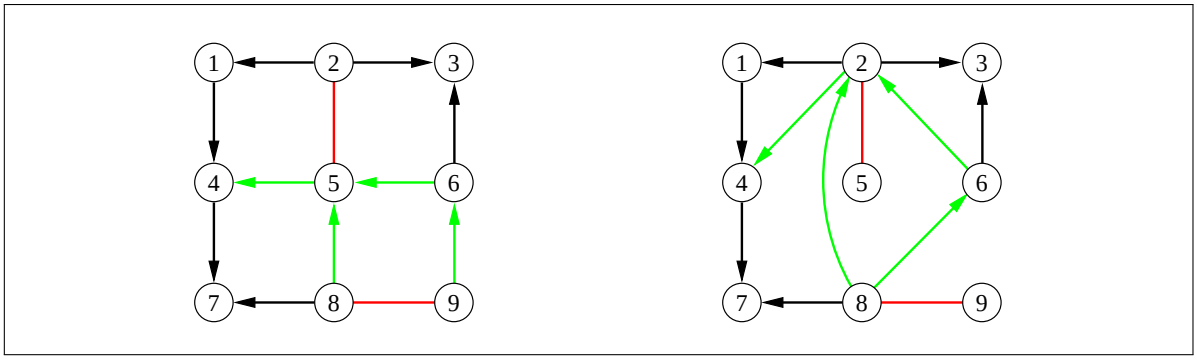


Figure 6.6. Constructing an acyclic and valley-free completion given peer-to-peer edges

4. Now test whether the graph induced by the already oriented edges is acyclic. If not reject the instance. Here, we would also identify cases where the above-mentioned chain-condition is not satisfied. Otherwise proceed as follows. Eliminate from each path $p \in P \setminus S$ all oriented edges. The remaining edges thus have indices in $\xi[p](e^p)$ not including e^p's vertices. Notice that until we have arrived at this point, by Lemma 6.9, we had no freedom of choice in orienting the edges as we did.

Let P' denote the remaining path set and let X denote the set of additionally oriented customer-to-provider edges. Note that $S \subseteq P'$. Since replacing all peer-to-peer edges with sibling-to-sibling edges does not destroy an acyclic and valley-free orientation, in the second phase of the algorithm we solve $(G, P' \setminus S, R \cup X)$ as an Acyclic ToR$(\{\rightarrow, \leftrightarrow\}, \{\rightarrow\})$ instance (where edges in R are taken as sibling-to-sibling edges). It is important to note that by our preprocessing phase, no new sibling-to-sibling edges have to be included. This computing step takes time $O(|P' \setminus S| + |R \cup X|) = O(N)$ by Corollary 6.18. Finally, re-insert all peer-to-peer edges from S in the obtained oriented graph, if the graph could be computed. Thus, the overall complexity is $O(N)$. ❑

Example 6.20. The left-hand side of Figure 6.6 shows an acyclic and valley-free completion for the path set $P = \{(2,3,6),(2,1,4,5,8),(8,5,6,9),(5,4,7,8)\}$ and the set $R = \{(2,5),(8,9)\}$ of peer-to-peer edges. We convince ourselves that the completion is a potential outcome of the algorithm described in the proof of Theorem 6.19. First note that the set S is empty in our case. There are two minimal pairs according to the second step: for the path $(2,1,4,5,8)$ the minimal pair is $(2,5)$ and for the path $(8,5,6,9)$ it is $(8,9)$. After executing the third and fourth step where no conflicts are detected we obtain as the instance for the second phase the path set $P' = \{(2,3,6),(1,4),(5,6),(5,4,7,8)\}$ and the edge set $R' = \{2 \leftrightarrow 5, 8 \leftrightarrow 9, 2 \rightarrow 1, 4 \leftarrow 5, 5 \leftarrow 8, 8 \rightarrow 5, 6 \leftarrow 9\}$. According to Theorem 6.16, Theorem 6.17, and Corollary 6.18 this instance is transformed further into an instance $P'' = \{(2,3,6),(1,4),(2,6),(2,4,7,8),(2,1,\perp),(2,4,\perp),(8,2,\perp),(8,6,\perp)\}$. Here, 2 represents 5 and 8 repesents 9. Algorithm 2 produces the acyclic and valley-free orientation depicted on the right-hand side of Figure 6.6 if the vertices of P'' (except $\perp$) are inserted into the set V in the order $8,6,2,1,4,7$. Finally, undoing our replacements we obtain the left orientation as desired.

The next corollary follows easily from Theorem 6.19.

Corollary 6.21. ACYCLIC TOR($\{\rightarrow, —\}, \{\rightarrow\}$) *can be solved in time* $O(N)$.

Proof. On a given instance (G, P, R), we use basically the same algorithm as described in the proof of Theorem 6.19. The only difference is that we have additional customer-to-provider edges in our set R that may contradict orientations of edges as forced by the peer-to-peer edges in a path. If we detect such a contradiction after the first phase of the algorithm, then we reject the instance. Otherwise we proceed as in the original algorithm. This is certainly an $O(N)$ algorithm. ❑

Theorem 6.22. ACYCLIC TOR($\{\rightarrow, \leftrightarrow, —\}, \{\rightarrow\}$) *can be solved in time* $O(N)$.

Proof. For a given path set P and a partial orientation R, first test whether the S2S-components are S2S-closed. If not reject the instance. This is correct since we are not allowed to introduce new sibling-to-sibling edges and any allowed cycle in $G(P)$ containing only sibling-to-sibling edges cannot be transformed into an allowed cycle by replacing one edge with a customer-to-provider edge or a peer-to-peer edge. Otherwise compute a reduced path set P' and a new partial orientation R' which is the same as R with the sibling-to-sibling edges being removed which is basically the same as in the preprocessing phase of the algorithms used for Theorem 6.17. Finally, we solve $(G(P'), P', R')$ as an ACYCLIC TOR($\{\rightarrow, —\}, \{\rightarrow\}$) instance in time $O(|P'| + |R'|) = O(N)$ using Corollary 6.21. ❑

6.4.5 Completions using peer-to-peer relations

We briefly discuss the implications of Theorem 6.22 for the inference problems where we use type set $\{\rightarrow, —\}$. Mainly, this can be handled with Lemma 6.13.

Theorem 6.23. ACYCLIC TOR($\{\rightarrow, \leftrightarrow, —\}, \{\rightarrow, —\}$) *can be solved in time* $O(N)$.

Proof. First eliminate all sibling-to-sibling edges by computing a reduced instance $(G(P'), P', R')$ where R' only consists of customer-to-provider edges and peer-to-peer edges (as in Theorem 6.22). Second, orient those not-yet-oriented edges as — that make P2P-components φ-complete (φ given by R'). Let R'' denote the new set of already oriented edges. Third, solve $(G(P'), P', R'')$ as an ACYCLIC TOR($\{\rightarrow, —\}, \{\rightarrow\}$) instance (as in Corollary 6.21). Correctness follows from Lemma 6.13. ❑

The orientations obtained by Theorem 6.23 do not give us much information as they ignore peer-to-peer edges. In particular for the case $K = \emptyset$, it is therefore desirable to know how we obtain a candidate set for peer-to-peer edges from an acyclic and valley-free orientation with type set $\{\rightarrow\}$. To find such a set of *maximum* cardinality is already NP-hard for the case of only valley-free orientations [40]. Though it could be possible that we get a lower complexity if we additionally suppose an acyclic orientation, the proof in [40] actually shows that this is not the case: deciding whether there are at least k edges for a given path set P and a given valley-free and acyclic orientation with type set $\{\rightarrow\}$ that can be oriented as — is NP-complete. On the other hand, we can easily compute a *maximal* set of peer-to-peer candidate edges in time $O(|P|^2)$ simply by iterating over all edges and checking valley-freeness and acyclicity of the resulting orientation after re-orienting the edges solely.

6.4.6 Completions using sibling-to-sibling relations

Finally, we turn to the inference problem using type sets U for completions such that $\leftrightarrow$ lies in U. In contrast to all cases we considered so far and which all can be solved in linear time we will now obtain computationally hard problems.

We first mention some exceptions that are easily solvable.

Proposition 6.24. *Let U be any type set containing $\leftrightarrow$. Then,* Acyclic ToR$(\emptyset, U)$ *and* Acyclic ToR$(\{\leftrightarrow\}, U)$ *can be solved in time $O(N)$.*

Proof. Orienting all not-yet-oriented edges as sibling-to-sibling edges is a solution. ❑

Proposition 6.25. *Let $K \subseteq \{\rightarrow, \leftrightarrow\}$ be any type set. Then,* Acyclic ToR$(K, \{\leftrightarrow, —\})$ *can be solved in time $O(N)$.*

Proof. Again, orient all remaining edges as sibling-to-sibling edges. In contrast to the previous proposition, we now have to check whether the orientation is acyclic and valley-free. This can be done in time $O(N)$. ❑

All remaining cases constitute NP-complete inference problems. The containment of Acyclic ToR(K, U) in NP for all pairs (K, U) of edge types is obvious. To prove NP-hardness results we take a closer look at sibling-to-sibling relations. As sibling-to-sibling relations establish an equivalence relation in any acyclic and valley-free orientation, we obtain a partition $[B_1, \ldots, B_r]$ of V from our orientation, i.e., a collection of non-empty subsets of V satisfying $B_i \cap B_j = \emptyset$ for $i \neq j$, and $B_1 \cup \cdots \cup B_r = V$, where the B_i are just equivalence classes according to the sibling-to-sibling relation, i.e., all B_i's are S2S-closed.

If the orientation is not already known to us then we search for suitable candidates for such partitions. We say that a pair $(P, [B_1, \ldots, B_r])$ is an *admissible decomposition* of P if and only if $[B_1, \ldots, B_r]$ is a partition of $V(P)$ such that the path set obtained by replacing all vertices of the same block with a unique representative and afterwards removing all multiple occurrences of representatives in all paths, allows an acyclic and valley-free orientation without $\leftrightarrow$. (Notice that this also includes S2S-looplessness of paths.) We further define the standard refinement relation $\subseteq$ on partitions which allows us to order partitions. Let $[A_1, \ldots, A_s]$ and $[B_1, \ldots, B_r]$ be two partitions of the same set V. We define

$$[A_1, \ldots, A_s] \subseteq [B_1, \ldots, B_r] \iff_{\text{def}} (\forall i, 1 \leq i \leq s)\,(\exists j, 1 \leq j \leq r)\,[\ A_i \subseteq B_j\ ].$$

Intuitively, the refinement relations holds between two partitions if we can union components of the finer partition to obtain the coarser partition, i.e., the partition with fewer components. We easily observe that admissible decompositions behave monotonically with respect to the refinement relation, i.e., if for partitions $\mathcal{A}$ and $\mathcal{B}$, $\mathcal{A} \subseteq \mathcal{B}$ and $\mathcal{A}$ is an admissible decomposition of P, then so is $\mathcal{B}$.

The following five theorems contain the simplest cases (in terms of pre-knowledge type sets) that are NP-complete and that all together are sufficient to imply NP-completeness for all inference problems not considered so far.

Theorem 6.26. Acyclic ToR$(\{\rightarrow\}, \{\rightarrow, \leftrightarrow\})$ *is* NP*-complete.*

Proof. For the proof of the NP-hardness, we use Two-in-Three Sat, i.e., the problem where we ask, given a 3CNF H, whether there is an assignment to all variables of H such that in each clause exactly two of the three literals are true. This problem is easily seen to be NP-complete by reduction from the well-known NP-complete problem One-in-Three SAT [63]. We show that Two-in-Three Sat reduces to Acyclic ToR($\{\rightarrow\}, \{\rightarrow, \leftrightarrow\}$). Let H be an arbitrary 3CNF having $m \geq 3$ clauses $C_1, \ldots, C_m$, each having exactly three different literals, and variables $x_1, \ldots, x_n$, i.e., $H = C_1 \wedge C_2 \wedge \cdots \wedge C_m$. We construct a path set P on the vertex set $\{C_1, \ldots, C_m, x_1, \ldots, x_n, \overline{x_1}, \ldots, \overline{x_n}\}$. Define the following sets of paths:

$$P_1 =_{\text{def}} \{ (C_1, C_2, C_3),\ (C_2, C_3, C_1),\ (C_3, C_1, C_2) \} \cup \{(C_1, C_j, C_2) \mid 4 \leq j \leq m\}$$

$$P_2 =_{\text{def}} \{ (x_i, x_j, \overline{x_i}) \mid 1 \leq i, j \leq n \text{ and } i \neq j\} \cup \{ (x_1, C_i, \overline{x_1}) \mid 1 \leq i \leq m\}$$

$$P_3 =_{\text{def}} \{ (l_{i1}, C_i, l_{i2}),\ (l_{i2}, C_i, l_{i3}),\ (l_{i3}, C_i, l_{i1}),\ (l_{i1}, \overline{l_{i2}}, l_{i3}) \mid 1 \leq i \leq m \text{ and } C_i = (l_{i1} \vee l_{i2} \vee l_{i3}) \}$$

The set P_1 guarantees that all clause vertices C_i belong to the same set, P_2 separates the literals from their negated literals, and P_3 indicates which literals will be satisfied. Now define $P =_{\text{def}} P_1 \cup P_2 \cup P_3$. Note that clearly, P can be computed in time polynomial in the number of clauses and variables of the input. We have to show that

$$H \in \text{Two-in-Three Sat} \iff (G(P), P, \{x_1 \rightarrow \overline{x_1}\}) \in \text{Acyclic ToR}(\{\rightarrow\}, \{\rightarrow, \leftrightarrow\}).$$

We prove both directions separately.

For ($\Rightarrow$), let $I : \{x_1, \ldots, x_n\} \rightarrow \{0, 1\}$ be an assignment to variables witnessing that $H \in$ Two-in-Three Sat. Define a set U to consist of all literals made true and all clauses. More specifically,

$$U =_{\text{def}} \{ x_i \mid 1 \leq i \leq n \text{ and } I(x_i) = 1 \} \cup \{ \overline{x_i} \mid 1 \leq i \leq n \text{ and } I(x_i) = 0 \} \cup \cup \{ C_i \mid 1 \leq i \leq m \}.$$

Hence, $\overline{U} = \{ \overline{x_i} \mid 1 \leq i \leq n \text{ and } I(x_i) = 1 \} \cup \{ x_i \mid 1 \leq i \leq n \text{ and } I(x_i) = 0 \}$. Clearly, $x_1 \in U \Leftrightarrow \overline{x_1} \notin U$. We are done if we can show that $[U, \overline{U}]$ is an admissible decomposition of P. In the following we use U (or $\overline{U}$) to denote a representative element of U (or, $\overline{U}$, respectively). We now consider all path sets individually:

1. Since $C_i \in U$ for all $1 \leq i \leq m$, all paths in P_1 have the form (U, U, U) which simplifies to (U).
2. Without loss of generality, suppose $x_i \in U$ which immediately implies that $\overline{x_i} \notin U$. It follows that the paths of P_2 have the form $(U, U, \overline{U})$ or $(U, \overline{U}, \overline{U})$ which both simplify to $(U, \overline{U})$.
3. In each clause, exactly two literals are satisfied. Without loss of generality, suppose that l_{i_1} and l_{i2} are these literals for clause C_i. Then, the paths of P_3 that correspond to C_i all have forms (U, U, U), $(U, U, \overline{U})$, or $(U, \overline{U}, \overline{U})$, thus, simplify to (U) or $(U, \overline{U})$.

Consequently, after eliminating multiple occurrences of paths, P simplifies to a subset of $\{(U), (\overline{U}), (U, \overline{U}), (\overline{U}, U)\}$ which, obviously, allows acyclic and valley-free orientations. Hence, $(G(P), P, \{x_1 \rightarrow \overline{x_1}\}) \in$ Acyclic ToR($\{\rightarrow\}, \{\rightarrow, \leftrightarrow\}$).

For ($\Leftarrow$), we assume that $(G(P), P, \{x_1 \to \overline{x_1}\})$ is a solvable Acyclic ToR($\{\to\}, \{\to, \leftrightarrow\}$) instance, i.e., there exists a decomposition $[U, \overline{U}] \subset [V]$ of P such that $x_1 \in U \Leftrightarrow \overline{x_1} \notin U$. Note that we can restrict ourselves to such 2-component decompositions because of the $\subseteq$-monotonicity of admissible decompositions. Without loss of generality, we may assume that $C_1 \in U$. Thus, $\{C_1, \ldots, C_m\} \subseteq U$ because of the definition of P_1. Furthermore, we have for all $1 \leq i \leq n$, $x_i \in U \Leftrightarrow \overline{x_i} \notin U$ (otherwise $U = V$ or $\overline{U} = V$ because of the definition of P_2). This allows to define an assignment $I : \{x_1, \ldots, x_n\} \to \{0, 1\}$ as follows

$$I(x_i) =_{\text{def}} \begin{cases} 1 & \text{if } x_i \in U, \\ 0 & \text{if } \overline{x_i} \in U. \end{cases}$$

We have to prove that I satisfies exactly two literals in each clause. Let C_i be any clause with literals l_{i1}, l_{i2}, and l_{i3}. As there are paths in P_3 having the form (l_{ij}, C_i, l_{ik}), there is an $l_{ir} \in U$. Without loss of generality, we assume that l_{i1} is such a literal. Since there exists for each pairs of literals of C_i such a path in P_3, there exists another literal $l_{is} \in U$, $r \neq s$. Without loss of generality, l_{i2} is such a literal. Due to the path $(l_{i1}, \overline{l_{i2}}, l_{i3}) \in P_3$, we know that l_{i3} is not in U. Overall, by definition of I, for each clause exactly two literals are made true. This shows $H \in$ Two-in-Three Sat. ❑

Theorem 6.27. Acyclic ToR($\{—\}, \{\to, \leftrightarrow\}$) *is* NP*-complete.*

Proof. We describe a polynomial-time many-one reduction from Acyclic ToR($\{\to\}, \{\to, \leftrightarrow\}$). Let (G, P, R) be an instance to that problem. Let $u \to v \in R$ be any oriented edge. We are done if we can force such an edge to be oriented in the same way without using a customer-to-provider edge. This can be achieved as follows. We add to the path set two new paths: $(u, v, \bot)$ and $(u, \bot, v)$ where $\bot$ is neither contained in P nor in R. Furthermore, we add the peer-to-peer edge $v—\bot$ to R. (Note that we can take the same new vertex $\bot$ for all edges in R.) It suffices to show that u and v cannot be siblings. Assume to the contrary, that u and v are siblings. Then, to avoid an oriented cycle $(u, v, \bot, u)$, u must have a peer-to-peer edge with $\bot$. However, this is a contradiction to valley-freeness, as $(u, \bot, v)$ now has two peer-to-peer edges. ❑

Theorem 6.28. Acyclic ToR($\{—\}, \{\leftrightarrow, —\}$) *is* NP*-complete.*

Proof. For $U = \{\leftrightarrow, —\}$, a careful analysis of the proof of Theorem 6.26 reveals that a 3CNF H belongs to Two-in-Three Sat if and only if $(G(P), P, \{x_1—\overline{x_1}\})$ belongs to Acyclic ToR($\{—\}, \{\leftrightarrow, —\}$). ❑

Theorem 6.29. Acyclic ToR($\{—\}, \{\to, \leftrightarrow, —\}$) *is* NP*-complete.*

Proof. For $U = \{\to, \leftrightarrow, —\}$, note that the proof of Theorem 6.28 relies on a partition of the vertex set into two components. Thus any acyclic and valley-free orientation assigns to edges whose vertices belong to different components the same edge type. Thus, we obtain that a 3CNF H belongs to Two-in-Three Sat if and only if $(G(P), P, \{x_1—\overline{x_1}\})$ lies in Acyclic ToR($\{—\}, \{\to, \leftrightarrow, —\}$). ❑

Theorem 6.30. Acyclic ToR($\{\to\}, \{\to, \leftrightarrow, —\}$) *is* NP*-complete.*

Proof. Let K be a type set containing $\rightarrow$ but not —. Since simultaneously replacing *all* peer-to-peer edges of an acyclic and valley-free orientation with sibling-to-sibling edges maintains acyclicity and valley-freeness, (G, P, R) lies in Acyclic ToR$(K, \{\rightarrow, \leftrightarrow, —\})$ if and only if (G, P, R) lies in Acyclic ToR$(K, \{\rightarrow, \leftrightarrow\})$. Thus, Acyclic ToR$(K, \{\rightarrow, \leftrightarrow, —\})$ is NP-complete by Theorem 6.26. ❑

6.5 The Maximum Acyclic Type-of-Relationship problem

In the last section we presented a linear-time algorithm for finding an acyclic and valley-free orientation of an AS graph for a given path set using customer-to-provider edges for completion and arbitary edge types for expressing pre-knowledge. If there is no such orientation the algorithm reports this. In real-world scenarios, of course, a path set of relevant size almost surely does not allow acyclic and valley-free orientations. In this section we will show how to modify this algorithm to find an acyclic orientation which is valley-free for a *large part* of the AS paths followed by some improvements to the basic scheme.

6.5.1 Definition

Formally, we consider an optimization problem as defined, e.g., in [6]. An optimization problem P consists of a quadruple $(I_P, \mathrm{SOL}_P, m_P, \mathrm{goal}_P)$ where

- I_P is the set of instances of P
- SOL_P is a function that associates to any input instance $x \in I_P$ the set of *feasible solutions* of x
- m_P is the *measure function*, defined for pairs (x, y) such that $x \in I_P$ and $y \in \mathrm{SOL}_P(x)$; for every such pair (x, y), $m_P(x, y)$ provides a positive integer which is the value of the feasible solution y
- $\mathrm{goal}_P \in \{\min, \max\}$ specifies whether P is a maximization or a minimization problem

The goal is typically contained in the name of the optimization problem. Given an input instance x, we denote by $\mathrm{SOL}^*_P(x)$ the set of optimal solutions of x, that is the set of solutions whose value is optimal. The value of any optimal solution of x is denoted by $m^*_P(x)$.

In the forthcoming we are interested in the following maximization problem which is specified by making use of a more intuitive notation:

Problem:	Maximum Acyclic ToR
Input:	A path set P (with possibly multiple occurrences of paths)
Solution:	A subset $P' \subseteq P$ allowing an acyclic and valley-free orientation (using the type set $\{\rightarrow\}$)
Measure:	The cardinality $\|P'\|$ of the subset P'

The problem formulation has some apparent deviations from the Acyclic Type-of-Relationship problem formulation of the last section:

- The graph $G(P)$ is not part of the input.
- The path sets are considered to be multisets.

– Pre-knowledge is not part of the input.

The problem setup is motivated by a rigorous approximability analysis. Later it will be more convenient (in particular, for algorithms) to have also the graph induced by P as part of the input. Also, pre-knowledge will be handled.

The goal of this section is to design algorithms that find a solution on a given P with the cardinality as high as possible.

6.5.2 Approximability results

We start with analyzing the existence of feasible algorithms for Maximum Acyclic ToR having provable approximation guarantees.

On the negative side, we rule out the existence of polynomial-time approximation schemes, unless P = NP. A polynomial-time approximation scheme (for a maximization problem) is an algorithm which computes in time polynomial in the size of the instance (x, ε) where $\varepsilon \leq 1$ a solution which is within a factor ε of the optimal solution for the input x (see, e.g., [6]). We feel that the achievable approximation quality is much poorer than stated here but we do not have a proof for this.

Technically, we prove that Maximum Acyclic ToR is hard for the class APX. APX is the subclass of optimization problems admitting algorithms with a constant approximation ratio (see, e.g., [6]).

Theorem 6.31. Maximum Acyclic ToR *is* APX*-hard.*

Proof. Maximum Acyclic Subgraph is the problem to compute, on a given directed graph $G = (V, E)$, a maximum-cardinality edge subset $E' \subseteq E$ such that (V, E') is acyclic. It is known that Maximum Acyclic Subgraph can be approximated with in a factor $\frac{1}{2}$ of the optimum but does not admit a polynomial-time approximation scheme, i.e., it is APX-complete [120]. We describe an L-reduction from Maximum Acyclic Subgraph to Maximum Acyclic ToR. According to [120, 6], an L-reduction from an optimization problem $P_1 = (I_{P_1}, \mathrm{SOL}_{P_1}, m_{P_1})$ to an optimization problem $P_2 = (I_{P_2}, \mathrm{SOL}_{P_2}, m_{P_2})$ is a quadruple (f, g, β, γ), where f, g are functions and β, γ are positive constants, such that:

– For any $x \in I_{P_1}$, $f(x) \in I_{P_2}$ is computable in polynomial time.
– For any $x \in I_{P_1}$, if $\mathrm{SOL}_{P_1}(x) \neq \emptyset$ then $\mathrm{SOL}_{P_2}(f(x)) \neq \emptyset$.
– For any $x \in I_{P_1}$ and for any $y \in \mathrm{SOL}_{P_2}(f(x))$, $g(x, y) \in \mathrm{SOL}_{P_1}(x)$ is computable in polynomial time.
– For any $x \in I_{P_1}$, $m^*_{P_2}(f(x)) \leq \beta \cdot m^*_{P_1}(x)$.
– For any $x \in I_{P_1}$ and for any $y \in \mathrm{SOL}_{P_2}(f(x))$,

$$|m^*_{P_1}(x) - m_{P_1}(x, g(x, y))| \leq \gamma \cdot |m^*_{P_2}(f(x)) - m_{P_2}(f(x), y)|.$$

We now construct an L-reduction in our special case. Given a directed graph $G = (V, E)$ (without loops) let f be the function defined by

$$f(G) =_{\mathrm{def}} \left\{ (u, v, (u, v)^-, (u, v)^+), (u, v, (u, v)^+, (u, v)^-) \;\middle|\; (u, v) \in E \right\}.$$

Here, $(u, v)^-$ and $(u, v)^+$ are new vertices which are associated with the directed edge (u, v) (and thus are different to vertices $(v, u)^-$ and $(v, u)^+$ if the edge (v, u) also belongs to E) and

which are not contained in V. Note that each acyclic and valley-free orientation of a path set containing both paths $(u, v, (u, v)^-, (u, v)^+)$ and $(u, v, (u, v)^+, (u, v)^-)$ orients u towards v. Therefore, if G is acyclic then $f(G)$ allows an acyclic and valley-free orientation. For any path set $P \subseteq f(G)$ define the corresponding edge set $g(G, P)$ by

$$g(G, P) =_{\text{def}} \left\{ (u, v) \in E \;\middle|\; (u, v, (u, v)^-, (u, v)^+) \in P \text{ and } (u, v, (u, v)^+, (u, v)^-) \in P \right\}$$

Clearly, f and g are computable in polynomial time. Note that $\|f(G)\| = 2\|E\|$ and $\|g(G, P)\| \geq \|P\| - \|E\|$. Let E^* be an optimal solution for $G = (V, E)$ and let P^* be an optimal solution for $f(G)$. Then, we easily obtain that

$$\|P^*\| \leq \|f(G)\| = 2\|E\| \leq 4\|E^*\|.$$

For the latter note that for each $G = (V, E)$ there is always an acyclic subgraph having at least $\frac{1}{2}\|E\|$ edges [120]. On the other hand, since (V, E^*) is acyclic, there is a topological ordering $\pi : V \to \{1, \ldots, \|V\|\}$ for (V, E^*), i.e., π is bijective and satisfies that if $(u, v) \in E^*$ then $\pi(u) < \pi(v)$. Moreover, since E^* is maximal, for each $(u, v) \in E \setminus E^*$ we have $\pi(u) > \pi(v)$. Thus, if we reverse all edges $(u, v) \in E \setminus E^*$, then we obtain an acyclic graph witnessed by the same topological ordering π. So, the path set

$$P(E^*) =_{\text{def}} f(G) \setminus \left\{ (u, v, (u, v)^-, (u, v)^+) \;\middle|\; (u, v) \in E \setminus E^* \right\}$$

allows an acyclic and valley-free orientation. Hence, we conclude $\|P^*\| \geq \|P(E^*)\| = \|E^*\| + \|E\|$. Thus, the following bound on the absolute error holds:

$$\|E^*\| - \|g(G, P)\| \leq \|P^*\| - \|E\| - (\|P\| - \|E\|) = \|P^*\| - \|P\|$$

Consequently, Maximum Acyclic Subgraph L-reduces to Maximum Acyclic ToR via $(f, g, 4, 1)$. Since Maximum Acyclic Subgraph belongs to APX, we obtain the APX-hardness of Maximum Acyclic ToR (see [6, Lemma 8.2]). ❑

The decision version of Maximum Acyclic ToR consists of all instances (P, k) such that P is a (multi)set of AS paths containing a subset $P' \subseteq P$ which has at least k paths and which allows an acylic and valley-free orientation.

Corollary 6.32. *The decision version of* Maximum Acyclic ToR *is* NP*-complete.*

Proof. The containment in NP is obvious. The hardness follows by reduction from the decision version of Maximum Acyclic Subgraph. The decision version of Maximum Acyclic Subgraph consists of all pairs (G, k) such that $G = (V, E)$ is a directed graph containing an acyclic subgraph with at least k edges. This problem is known to be NP-complete [84]. For the reduction we use the function f from the proof of Theorem 6.31 and prove the following: Let $G = (V, E)$ be a directed graph with m edges. The pair (G, k) belongs to the decision version of Maximum Acyclic Subgraph if and only if the pair $(f(G), k+m)$ belongs to the decision version of Maximum Acyclic ToR. For the direction from left to right let $E' \subseteq E$ be an edge set such that $\|E'\| \geq k$ and (V, E') is acyclic. Define $P(E')$ as in the proof of Theorem 6.31. Then, by the same arguments as there, we obtain that $P(E')$ has an acyclic and valley-free orientation and $\|P(E')\| = k+m$. For the direction from right to left let $P \subseteq f(G)$ be a path set such that $\|P\| \geq k+m$ and P has an acyclic and valley-free orientation. Define $E' =_{\text{def}} g(G, P)$ for the same g as in the proof of Theorem 6.31. We conclude that (V, E') is acyclic and $\|E'\| \geq \|P\| - m \geq k$. This proves the NP-hardness of the decision version of Maximum Acyclic ToR. ❑

On the positive side, we do not know any non-trivial upper bound on the approximation quality of Maximum Acyclic ToR. However, if we restrict the allowed path lengths, then we obtain a constant approximation ratio. Maximum Acyclic ToR$_k$ denotes the problem to find, given a set of paths of length at most k, a maximum-cardinality subset which allows an acyclic and valley-free orientation.

Theorem 6.33. *For each $k \in \mathbb{N}_+$, the problem* Maximum Acyclic ToR$_k$ *can be approximated within a factor of $\frac{2^k}{(k+1)!}$ of the optimum in polynomial time.*

Proof. The proof is by induction over k. For the base of induction, suppose $k = 1$. Obviously, each path set consisting of paths of length one, i.e., edges, allows an acyclic and valley-free orientation. So, the optimum can be achieved which implies an approximation ratio 1. As to the induction step, let $k > 1$. Suppose we are given a path set P. For a vertex $v \in V(P)$, let $\beta_P(v)$ denote the number of paths of P such that v is located at the boundary and let $\mu_P(v)$ denote the number of paths of P such that v is located in the middle. Let $P(v) \subseteq P$ be the set of all paths containing the vertex v. By the pigeon-hole principle, there is always a vertex $v \in V(P)$ such that $\beta_P(v) \geq \frac{2}{k-1} \cdot \mu_P(v)$. Fix such a vertex v_0. Iteratively find such vertices in the path set $P \setminus P(v_0)$. Finally, this produces a sequence of vertices $v_0, v_1, \ldots, v_r$ such that

1. $P = \bigcup_{i=0}^{r} P_i$, where $P_0 =_{\text{def}} P(v_0)$ and for $1 \leq i \leq r$, $P_i =_{\text{def}} (P \setminus \bigcup_{j=0}^{i-1} P_j) \cap P(v_i)$, and
2. for all $0 \leq i \leq r$ it holds that $\beta_{P_i}(v_i) \geq \frac{2}{k-1} \cdot \mu_{P_i}(v_i)$.

Let P_i' be the set of paths obtained from the path set P_i as follows: remove completely all paths with v_i in the middle. In the remaining paths of P_i remove the vertex v_i from the path (which is a boundary vertex). Clearly, the paths of P_i' have length at most $k-1$. Define $P' =_{\text{def}} \bigcup_{i=0}^{r} P_i'$ by multiset union. That is, the multiplicity of a path $p \in P'$ corresponds to the number of v_i's such that $p \in P_i$. Then, P' consists of paths of length at most $k-1$ and can be computed in polynomial time. Moreover, we obtain

$$\|P\| = \sum_{i=0}^{r} \|P_i\| = \sum_{i=0}^{r} \beta_{P_i}(v_i) + \mu_{P_i}(v_i) \leq \left(\frac{k-1}{2} + 1\right) \cdot \sum_{i=0}^{r} \beta_{P_i}(v_i) = \frac{k+1}{2} \cdot \|P'\|.$$

By induction hypothesis, there is an algorithm outputting a path set $L' \subseteq P'$ such that L' allows an acyclic and valley-free orientation and $\|L'\| \geq \frac{2^{k-1}}{k!} \cdot \|P'\|$. We easily obtain from L' a path set L by attaching the vertices v_i back to the corresponding paths and orienting the edges from v_i towards the middle, i.e., $L =_{\text{def}} \{(v_i, u_0, \ldots, u_\ell) \in P | (u_0, \ldots, u_\ell) \in L'\}$ where we assume that each path occurs with maximum multiplicity prescribed by P. It is easily seen that if we fix any acyclic and valley-free orientation of L', then this orientation of L is acyclic and valley-free. Furthermore,

$$\|L\| \geq \|L'\| \geq \frac{2^{k-1}}{k!} \cdot \|P'\| \geq \frac{2^{k-1}}{k!} \cdot \frac{2}{k+1} \cdot \|P\| = \frac{2^k}{(k+1)!} \cdot \|P\|.$$

This proves the theorem. ❑

Observe that the proof of the theorem shows that each set of paths of length at most k contains at least a $\frac{2^k}{(k+1)!}$ fraction which allows an acyclic and valley-free orientation. Unfortunately, these fractions decrease very quickly as path length increases, e.g., for path length 2 the fraction is $\frac{2}{3} \approx 66.7\%$, for length 3 it is $\frac{1}{3} \approx 33.3\%$, for length 4 it is $\frac{2}{15} \approx 13.3\%$, and for length 5 it is already $\frac{2}{45} \approx 4.4\%$.

6.5.3 The basic heuristic

The linear-time algorithm for Acyclic ToR$(\emptyset, \{\rightarrow\})$ from the last section is based on the observation that a source vertex (*i.e.*, one that itself has no incoming edges) cannot be in the middle of a path. In this subsection we describe how it can be combined with the extension for discarding interfering paths, but before digging into the details we should fix some notation.

During its execution the algorithm will remove vertices which have been finished. To avoid having to change all the paths we manage a set R of *removed* vertices. Given such a set, a vertex v in a path p is called *inner vertex* of p relative to R if it is surrounded in p by vertices u and w with $u, w \notin R$. A vertex not belonging to R that is not an inner vertex for any path of the paths set is called *free*.

In the algorithm (details in Algorithm 3) we count for each vertex v in count(v) the number of paths for which v is an inner vertex. The set F of free nodes is then easily initialized by all vertices v satisfying count$(v) = 0$. The main loop (lines 12–40) can be separated into two phases. The first phase (lines 13–24) is the taken from Algorithm 2 in Section 6.4. While there is a free vertex u in F we interpret it as a source vertex and thus orient the edges to its neighbors (not yet in R) as customer-to-provider. As it is removed afterwards (*i.e.*, put into R) we adjust the count variables accordingly to find nodes which are now freed. If we can remove all vertices this way ($R = V$) we know that the orientation is valley-free and acyclic, as the vertices have been removed in topologically sorted order (each vertex had indegree 0 when it was removed).

If the first phase ran out of free vertices before all vertices could be removed, we need to create additional free vertices by discarding paths (starting from line 28). As we want to discard as little paths as possible, we select a vertex v which is an inner vertex for the minimal number of paths (as indicated by count(v)). By removing all those paths, v becomes a free vertex and we continue with the first phase.

We point out that the algorithm can easily be modified to work for a weighted path set, where the goal consists of minimizing the overall weight of the dropped path. The variables count(v) then would not store the number of paths for which v is an inner vertex but the overall weight of those paths. All steps updating count(v) are adjusted accordingly for this.

6.5.4 Handling pre-knowledge

In Section 6.4 the influence of pre-knowledge on the acyclic type-of-relationship problem has been discussed and an extension for the acyclic inference algorithm for handling known customer-to-provider edges has been presented. As our heuristic is a modification of the algorithm given there, we can easily transfer this extension.

The idea in the proof of Theorem 6.16 was to introduce for each known customer-to-provider edge $u \rightarrow v$ a new path $(u, v, \perp)$ where $\perp$ is an artificial vertex with count$(\perp) = \infty$. So the only way to make v a free vertex is to remove u which includes the introduction of an edge $u \rightarrow v$ as desired. Of course these new paths need not be constructed explicitly, but can be handled implicitly by modifying the above heuristic.

As the heuristic is allowed to drop paths hindering a consistent orientation, interpreting known edges as paths also allows dropping these edges in case of a conflict. Often however the explicit pre-knowledge is trusted more than an set of AS paths. For this case we can increase the weight of the (virtual) paths introduced in this step. In our implementation all

Algorithm 3: Heuristic for Maximum Acyclic ToR

Input: AS path set P, AS graph $G = (V, E)$ for P
Output: A set N of discarded paths and an orientation of G with customer-to-provider edges such that the orientation is acyclic and valley-free for all paths in $P \setminus N$

```
1  foreach v ∈ V do count(v) := 0
2  foreach p ∈ P do
3  |  foreach inner node v of p relative to ∅ do
4  |  |  count(v) := count(v) + 1
5  |  end
6  end
7  F := ∅, R := ∅, N := ∅
8  foreach v ∈ V with count(v) = 0 do
9  |  F := F ∪ {v}
10 end
11 done := false
12 while ¬ done do
13 |  while F ≠ ∅ do
14 |  |  remove vertex u from F
15 |  |  foreach v ∈ V \ R with {u, v} ∈ E do
16 |  |  |  orient {u, v} as customer-to-provider
17 |  |  |  foreach p ∈ P containing u and v as neighbors do
18 |  |  |  |  if v is an inner node of p relative to R then
19 |  |  |  |  |  count(v) := count(v) − 1
20 |  |  |  |  |  if count(v) = 0 then F := F ∪ {v}
21 |  |  |  |  end
22 |  |  |  end
23 |  |  end
24 |  |  R := R ∪ {u}
25 |  end
26 |  if R = V then
27 |  |  done := true
28 |  else
29 |  |  v := argmin_{u∈V\R} count(u)
30 |  |  foreach path p ∈ P with v ∈ p do
31 |  |  |  if v is an inner node of p relative to R then
32 |  |  |  |  N := N ∪ {p}
33 |  |  |  |  foreach inner node u of p relative to R do
34 |  |  |  |  |  count(u) := count(u) − 1
35 |  |  |  |  |  if count(u) = 0 then F := F ∪ {u}
36 |  |  |  |  end
37 |  |  |  end
38 |  |  end
39 |  end
40 end
41 return N
```

AS paths are weighted with 1 and all paths originating from known edges are assigned the same weight W. Thus a customer-to-provider edge may only be discarded, if we can save at least W paths instead.

6.5.5 Readding

After finding a path set which can be oriented valley-free, we enter a second phase where edges which had to be dropped before are readded to the path set if possible.

This is done as follows: The first run of the heuristic returns a set D of discarded paths from which we can determine the set of orientable paths P. We decide on the number $k > 1$

of additional runs we are willing to spend and partition D arbitrarily into k sets $D_1, \ldots, D_k$. For each such set D_i we then run above heuristic for the path set $P \cup D_i$ and again receive a set of discarded paths D'. If $\|D'\| < \|D_i\|$ we could reduce the number of dropped paths and use $(P \cup D_i) \setminus D'$ from now on instead of P, otherwise we stick with the original P. As we treat edges from the pre-knowledge as paths as described before, this strategy will work for readding those discarded edges as well.

We should note that other readding strategies proposed in [40] do not work well in our case.

6.6 Experiments

In this section we report experimental findings from testing our algorithmic methods and the acyclicity assumptions.

6.6.1 Obtaining real-world data

To run the inference algorithms we need valid AS paths used in the Internet. Additionally we are interested in at least partial information on the business relationships between ASes, to both verify our inference results as well as using them as previous knowledge for the inference algorithms. Unfortunately there is no single exhaustive source listing those relationships, so we have to use other publicly available information and try to extract them from it.

In this subsection we discuss the sources and methods used to retrieve the data used herein in the following paragraphs. All data was collected on 3/31/2006. In case of multiple data samples for this day (as offered by some archives) the latest one available for the day was used.

AS paths

We extracted AS paths from the Routing Information Bases provided at two sources:

- *Route collectors* from RIPE RIS [131] and the Route Views project [150]: Both sites operate several *BGP beacons*, i.e., modified BGP routers which exchange routing information via peering sessions with other routers but do not advertise routes. Both sites provide an archive with the routing tables of these beacons from which AS paths can be extracted. We used the last Routing Information Bases available for 3/31/2006 from each beacon available (for RIPE RIS these are rrc00 to rrc07, and rrc10 to rrc15, for RouteViews these are RouteViews2, EQIX, ISC, KIXP, LINX, and WIDE).
- *Public route servers* listed at [86]: We established a telnet session and obtained their RIBs tables using the `show ip bgp` command. Due to technical issues (time-outs, etc.) not all of them could be collected. Table 6.4 lists all RIBs used for our experiments.

The AS paths retrieved from the source are further transformed in two steps:

- *Preprocessing:* By design the AS paths used in BGP contain no cycles. The only exception is prepending where the same AS appears multiple times at consecutive positions. As prepending is only relevant for routing decisions but not for the inference algorithms discussed here, we preprocessed the paths by discarding all but the first AS from a sequence

Autonomous Systems	Internet Service Provider
AS 553	BelWue
AS 852	Telus - East Coast
AS 852	Telus - West Coast
AS 3257	Tiscali
AS 3561	Savvis Communications
AS 3741	Internet Solutions
AS 4323	Time Warner Telecom
AS 5388	Planet Online
AS 5511	Opentransit
AS 6539	GT Group Telecom
AS 6648	Bayantel Inc.
AS 6667	Eunet Finland
AS 6730	Sunrise
AS 6939	Hurricane Electric
AS 7474	Optus Route Server Australia
AS 7911	Wiltel
AS 8220	Colt Internet
AS 9132	Broadnet Mediascape Communications AG
AS 12312	Tiscali Germany
AS 13645	Host.net
AS 15290	Allstream - East
AS 15290	Allstream - West

Table 6.4. Routing Information Bases extracted from public route servers.

of identical AS numbers. Even after this preprocessing there are still paths containing cycles or ASes from the private range (AS64512 to AS65534), probably due to configuration errors. Such paths were completely removed from the data pool.

– *Nomalization:* We normalize the path set to include only those paths that were not already implied by other AS paths. More formally, an AS path is dropped if itself or the reversed path is a substring (including identity) of another AS path in the data pool. Note that this include elimination of multiple occurrences of an AS path.

Using the sources presented above and the preprocessing and normalization procedure we received a set of 2,002,714 AS paths with an average path length of 3.44.

RPSL information from WHOIS

Our primary data source for relationships are the WHOIS databases which besides other information contain structured RPSL records for ASes which are identified over the `aut-num` object. A list of available databases can be found at the Internet Routing Registry [107]. We decided to use the databases from APNIC, ARIN, LEVEL3, RADB, RIPE, and VERIO. Other approaches for extracting business relationships from WHOIS data are described in

[42, 138]. As we are only interested in a set of relationships to test our inference results with and not in a complete view on the Internet, we use a simpler approach which is motivated by those mentioned before and described next.

The most valuable information in these `aut-num` objects are the import and export rules which encode the routing policy of the AS. As RPSL is a very rich language (the Bison grammar given in [2] covers eight pages) we do not try to understand all of it but discard all rules not of the most primitive format:

– the number of the AS these rules apply to, followed by
– a list of ASes (possibly using `as-set` objects) routes are imported from, followed by
– a list of ASes (possibly using `as-set` objects) routes are exported to.

To indicate the lack of filtering AS lists can be replaced by the `ANY` keyword. As most rules are using this simple format we are not losing to much information. To get rid of conflicting entries all records appearing in multiple databases are skipped. Additionally we discard all records not modified since 1/1/2005 to eliminate unmaintained entries.

When processing the `aut-num` object of some AS i we check each AS j for which both import and export policies have been parsed. The import-export policies of AS i with respect to AS j can be roughly categorized into *unfiltered*, i.e., if `ANY` is used, and *filtered*, i.e., if a list of AS numbers follows. The Selective Export Rule only allows to export anything or to export only own and customer routes. This maps nicely with our filtered and unfiltered policies, so having both the import and export rules between i and j we can directly infer one of the four relationship types (counting customer-to-provider and provider-to-customer relations separately). More specifically, we infer the relationship between AS i and AS j as follows from the `aut-num` object for AS i:

– If the import policy is filtered and the export policy is unfiltered, then AS i is a customer of AS j.
– If the import policy is unfiltered and the export policy is filtered, then AS i is a provider of AS j.
– If both policies are filtered, then AS i and AS j have a peer-to-peer relationship.
– If both policies are unfiltered, then we assume that AS i and AS j have a sibling-to-sibling relationship.

However as we do not expect ISPs to document internal routing policies in public registries such sibling-to-sibling links are considered configuration errors (or documentation inconsistencies) and discarded.

Example 6.34. The following two (hypothetical) RPSL entries in a WHOIS database show a very simple policy configuration between AS 123 and AS 456.

```
aut-num:    AS123
import:     from AS456
            accept ANY
export:     to AS456
            announce AS456

aut-num:    AS456
import:     from AS123
```

```
            accept AS123
export:     to AS123
            announce ANY
```

AS 123 exports only own and customer routes to AS 123 but imports everything from it, so the export policy is filtered while the import policy is unfiltered. From our scheme we thus conclude that AS 123 is a customer of AS 456. Observe that this relationship is confirmed by the `aut-num` object for AS 456. Both entries provide consistent information.

As the information in the WHOIS tables is usually provided voluntarily and often outdated we cannot expect the results from this approach to be absolutely correct. In [138] an estimation of the correctness of a more refined approach is given to be at least 83% due to the errors in the WHOIS tables. On the other hand there more complicated rules are analyzed in order to a more complete picture of the Internet relationships than we do here. As we only need partial information and may drop everything not fitting with our assumptions we expect to have more accurate results.

One step towards improving the quality of the extracted relationship information is exploiting the symmetry of the import and export rules. In Example 6.34 we observed consistent information from both WHOIS entries. Ideally for each pair of ASes we have two characterizations of their relationships. If they are consistent (both peer-to-peer or customer-to-provider with provider-to-customer) our trust in this information increases, otherwise we discard the information for this AS pair. As many ASes do not disclose their routing policy we often only find one characterization for a relationship. We keep this information separate and use it for testing the robustness of the inference algorithms on erroneous data.

From the WHOIS databases listed above we extracted 4,438 edges which are confirmed twice by the RPSL statements and 33,828 that were only encountered once. Because the inference algorithms only find relationships for AS pairs adjacent on at least one path we removed all AS pairs from above edge sets not neighbored on any path. After this cleaning process there are 964 customer-to-provider and 598 peer-to-peer relationships in the first set and 6,564 customer-to-provider and 3,988 peer-to-peer edges in the second set.

Using BGP communities

In [165] a different approach for extracting relationships using AS path attribute COMMUNITY (see Section 5.4)) is described. Communities are arbitrary numbers that are tagged to routes and can be used in routing decisions. To make these numbers unique the number of the AS which added the community is included in the community number. Besides influencing other routing parameters communities are often used to document the origin of a route including whether they were received from a provider or from a peer. Unfortunately there is no general agreement on the usage of community numbers, so their meaning depends on the originating AS. Furthermore, there is no unique way of documenting them. However some ISPs are using the `remarks` entries in the WHOIS `aut-num` objects for this purpose (an example is discussed belows).

As the contents of the `remarks` blocks are not standardized it is complicated to automatically extract the meaning of documented communities, so we performed this step manually. Although harvesting the entire WHOIS databases for this information by hand is nearly impossible, limiting this search to `remarks` lines containing an additional colon and one of the

phrases “peer”, “customer”, “provider”, “uplink”, or “upstream” makes this a manageable task.

Example 6.35. The following RPSL fragment contains a block of `remarks` clauses to document BGP communities.

```
aut-num:    AS8437
as-name:    UTA-AS
...
remarks:    Communities tagged on inbound routes
remarks:    ===================================
remarks:
remarks:    8437:1000   Customer Routes
remarks:    8437:1004   Peering Routes
```

The usage of this block is as follows. For instance, let us consider an AS path `AS1 AS2 AS8437 AS3 AS4` with the community `8437:1000` tagged then we know that AS 8437 received the path `AS3 AS4` from its customer AS 3, so we can infer a customer-to-provider edge between AS 8437 and AS 3.

As explained in Example 6.35 we can infer the relationship for one pair of ASes if we are given an AS path with a community tag containing a meaningful phrase. With a list of known communities extracted from all WHOIS databases above we used this approach on all Routing Information Bases collected from RIPE RIS and Route Views. The RIBs obtained using `show ip bgp` cannot be used as they lack information about communities. Although many communities are discarded by routers on the path, enough of them are in RIBs to allow us finding 1,882 distinct customer-to-provider and 1,495 peer-to-peer edges. Due to the method used they are automatically on an AS path.

Sibling-to-sibling relationships

The approaches presented so far find no sibling-to-sibling edges, as ISPs only document their *external* routing behavior. To get around this we assume that all ASes belonging to the same organization are siblings of each other which is a sensible assumption under economic aspects. Further we expect a company to have the same contact persons for all its ASes, so `aut-num` objects having different entries in one of the `mnt-by`, `admin-c`, or `tech-c` clause are considered not to be siblings. From the remaining pairs of “sibling candidates” we only keep those sharing a long common prefix or suffix in the name or the description of the ASes. This gives a set of potential sibling clusters which are then manually verified in a final step.

It turns out that using only the information from WHOIS we sometimes cannot decide whether two ASes are siblings or if one is a customer AS which is only maintained by its provider. As only small ASes at the border of the Internet are likely to delegate their administration, and introducing a wrong sibling-to-sibling relationship in this case does not do too much harm, we decide for keeping the sibling-to-sibling edge when in doubt. This way we obtain 5,301 sibling-to-sibling edges from 360 sibling cliques.

Contrary to the customer-to-provider and peer-to-peer edges the sibling-to-sibling edges are not used to validate the results of the algorithms but rather to preprocess the input data by replacing each AS number with a representative sibling in all paths and edges.

Summary of data used

We summarize the data used for the experiments presented later. For each experiment we used the complete AS path set as described before. The edges are separated into two sets which we call the *reliable set* and the *full set* respectively. In the reliable set there are the customer-to-provider and peer-to-peer edges obtained from BGP communities and those appearing twice in the RPSL extracted edges. We expect this set to contain only little errors. The full set consists of the reliable set and additionally all edges appearing only once in the RPSL edges. This set is especially useful for testing the algorithms with uncertain pre-knowledge, as probably 5 to 20 percent of the information contained there is incorrect.

As a preprocessing step for each experiment we used the sibling edges obtained before to replace all sibling ASes with a single AS. This slightly influences the numbers reported before as some paths and edges had to be dropped due to being malformed after this step. Additionally inconsistent edges (e.g., a peer-to-peer and customer-to-provider relationship for the same edge) were removed. The details for the final path and edge sets ultimately used as

AS Paths	**Reliable Edge Set**	**Full Edge Set**
2,002,680 paths	2,739 customer-to-provider edges	8,850 customer-to-provider edges
containing 21,862 ASes and	2,000 peer-to-peer edges	5,434 peer-to-peer edges
inducing 56,922 AS pairs		

Table 6.5. Technical summary of the data used.

input and reference for the algorithms are shown in Table 6.5. Notice that the average length of the AS paths is 3.43.

6.6.2 Validating the acyclicity model

Using the data from the previous subsection we compare our acyclicity model to the real Internet. Therefore we build graphs from the edges collected and check if they are acyclic according to our formulation. The main findings are the following:

– When using customer-to-provider relationships only the graphs we receive are acyclic for both the reliable and the full data set.
– When including peer-to-peer edges into these graphs cycles are observed for both data sets. To get an impression of how much acyclicity is violated we tested every triads in the graphs whether it was a directed circle or not. For the reliable edge set the graph contains 2,826 triads from which 253 (8.95%) induce a directed cycle. In the full edge set there are 18,621 triads, 2,427 (13.03%) of them are cyclic.

We take these results as an indication that indeed the overall structure of AS relationships is acyclic but our model of acyclicity is still imprecise when it comes to peer-to-peer relationships. This is probably mostly due to assuming the relation "roughly of the same size" to be transitive when interpreting peer-to-peer edges.

6.6.3 Comparing inference algorithms

In this subsection we examine the accuracy and sensitivity to pre-knowledge of the inference heuristic developped in Section 6.5. Forthcomingly, we refer to this heuristic as AHeu. Our examination of AHeu is based on an experimental comparison with other inference algorithms proposed in the literature.

Existing algorithms

We compare AHeu to following three standard algorithms:

- *Gao:* In the seminal paper [61] on inferring business relationships the following heuristic is given based on two assumptions:
  - For a pair of autonomous systems which are connected, always the larger one is the provider for the smaller one.
  - The size of an autonomous system is proportional to its degree in the AS graph.

  The algorithm operates in three phases. At first the AS graph is constructed from the set of AS paths by inserting an edge for each pair of ASes adjacent on any path. This graph is also used to calculate the degree of an autonomous system. The second phase finds for each path the AS with maximum degree (the *top provider*) and records for each edge on the path the direction towards the top provider as preferred. Finally the third phase iterated over all edges of the AS graph and compares how often each of both possible directions was preferred. If the vote is clear, the edge is oriented as customer-to-provider in the implied direction, in case of a tie a sibling-to-sibling edge is inferred. The decision whether the vote is clear is based on a threshold L which is recommended to be set to 1 (for details see [61]).
- *APX:* In [40] a simple approximation algorithm based on a random orientation of the AS graph is presented. The idea is to orient each edge either as customer-to-provider or provider-to-customer (with probability $\frac{1}{2}$). Then a path of length k is valley-free with probability $\frac{k+1}{2^k}$ as there are only $k+1$ valley-free configurations out of 2^k possible orientations. Thus, if the length of the input paths can be bounded by k, it is expected that a fraction of $\frac{k+1}{2^k}$ of all given paths is valid. Using conditional probabilities this algorithm is derandomized yielding the desired approximation algorithm with approximation factor $\frac{k+1}{2^k}$. We refer to this algorithm as APX. The mentioned approach can be improved by means of semidefinite programming (SDP) as described in the paper.
- *DPP*:* Another heuristic approach based on the 2SAT problem is also given in [40]. They start with an arbitrary orientation of all edges of the AS graph as either customer-to-provider or provider-to-customer. To each edge a boolean variable is assigned which is true if this edge should be reversed and false otherwise. The crucial observation is that valley-freeness can be checked for locally by inspecting edge pairs adjacent in an AS path. For a single pair of edges there is exactly one situation which is forbidden (both edges pointing away from each other). If an orientation is found where this forbidden situation is

avoided for all edge pairs, all AS paths are automatically valley-free. These forbidden edge configurations are formulated as a boolean expression over the edge variables introduced before. As only two edges are involved in each forbidden pattern, the conjunction of all these expressions yields a 2SAT instance which can be solved in linear time. From this 2SAT solution the orientation for the AS graph is easily found by flipping all edges whose variables are true.
The algorithm just described can answer the question whether it is possible to find an orientation making *all* paths valid. To get an algorithm for maximizing the number of valid paths, the above algorithm is applied on a set of AS paths. If it finds a suitable orientation, all paths are valid. Otherwise the set of "blocking" variables (and thus edges) is calculated (this information is easily available from the way the 2SAT problem is solved). From these edges the one contained in the least number of paths is selected and all paths containing this edge are removed. With this reduced path set the procedure is restarted until finally an orientation for which all remaining paths are valid is found. It should be pointed out that using suitable data structures the described process can be performed incrementally without having to reconstruct the 2SAT instance in each iteration. After this first phase the heuristic tries to reinsert the removed paths.
In [40] two methods for this are mentioned. The first one (called DPP) simply adds each single path and tests if there is still a valid orientation. If not, the path is rejected. The second method (DPP*) considers each edge not yet oriented and counts how many of the removed paths would benefit from each of the possible orientations of this edge. Then the edge is oriented according to the majority of these "votes". As this second approach is reported to be both faster and better in terms of the number of invalid paths we only use DPP*.

We also explore the influence of previous knowledge on the quality of the induced relationships. As none of the mentioned algorithms has support for this kind of information we are using a simple preprocessing algorithm in conjunction with them when calculating inference results for given pre-knowledge. This algorithm calculates a partial orientation of the AS graph as a set of remaining AS paths. The final result is then obtained by applying one of the presented inference algorithm on the remaining paths and merging the orientations from the inference algorithm with those from the preprocessor.

The simple preprocessing algorithm only handles customer-to-provider edges in the pre-knowledge. It exploits the fact that a single edge on an AS path fixes the orientation of many other edges on this path. These edges are added to the pool of known edges. We repeat this step for all paths until we gain no new information. The details are outlined in Algorithm 4. Two kinds of errors can occur during the application of the algorithm which both indicate errors in the data (partially from BGP misconfigurations). On the one hand a path might be not orientable valley-free with respect to the already known orientations, on the other hand newly found orientations can be in conflict with the existing knowledge. We implemented no advanced correction scheme for these situations but simply react with discarding new the conflicting information.

Experimental findings

We executed Gao, APX, DPP*, and two versions of AHeu on the path set from Subsection 6.6.1 and compared the resulting edge classification to the reliable and full edge set described

Algorithm 4: Preprocessing algorithm for handling pre-knowledge

Input: AS path set P, set of known customer-to-provider edges E
Output: A reduced set of AS paths usable to infer the edges which could not be inferred from E

```
1  P' := P, P := ∅
2  while P ≠ P' do
3  |   P := P', P' := ∅
4  |   foreach path p ∈ P do
5  |   |   let p = p_1, p_2, ..., p_n
6  |   |   up := max{i | p_{i-1} → p_i is in E} ∪ {0}
7  |   |   down := min{i | p_i ← p_{i+1} is in E} ∪ {n}
8  |   |   if up > down then
9  |   |   |   drop this path, as is can not be valley-free for E
10 |   |   else
11 |   |   |   for i := 2 to up −1 do
12 |   |   |   |   insert edge p_{i-1} → p_i into E unless edge p_{i-1} ← p_i exists
13 |   |   |   end
14 |   |   |   for i := down +1 to n − 1 do
15 |   |   |   |   insert edge p_i ← p_{i+1} into E unless edge p_i → p_{i+1} exists
16 |   |   |   end
17 |   |   |   if up < down then  insert path p_up, ..., p_down into P'
18 |   |   end
19 |   end
20 end
21 return P
```

there. We are both interested in the number of paths which are not oriented correctly, i.e., which are not valley-free, and the number of customer-to-provider edges that were not inferred as such. As none of the algorithms used is capable of identifying peer-to-peer relationships we do not compare the inferred results to our known peer-to-peer edges. An exception is the Gao-algorithm as it introduces sibling-to-sibling edges which we do not want to include in the inferred results, thus paths containing a sibling-to-sibling edge are counted as invalid as well.
Table 6.6 shows the detailed results of our experiments. Recall that the edge types $\rightarrow$, $\leftarrow$ and $\leftrightarrow$ are used to represent customer-to-provider, provider-to-customer, and sibling-to-sibling relationships. The two main findings are:

– Our heuristic AHeu has the lowest number of invalid paths as well as the least number of errors when compared to the reference data.
– Using the readding strategy for AHeu can lower the number of invalid paths even more but at the cost of reliability of the resulting edge classifications.

Another aspect we are interested in is the behavior of these algorithms when having initial pre-knowledge of some of the edges. Therefore we repeated the experiment described before but provided the algorithms with a certain fraction of the edges used for comparison later. As the choice of edges provided to the algorithm has some influence on its results we averaged all results over 5 random samples of the edge set. Additionally the same samples were used for all of the algorithms. Our heuristic AHeu is the only one explicitly supporting pre-knowledge, so we used the preprocessing Algorithm 4 to augment the remaining algorithms accordingly. This should be kept in mind when comparing the results as thus our heuristic is the only one capable of "trading" edge errors, i.e., violated pre-knowledge, for violated paths.

Algorithm	**Invalid Paths**	**Misclassified → Edges for the Reliable Edge Set**	**Misclassified → Edges for the Full Edge Set**
Gao	27.366% (249 not valley-free) (547,811 with ↔ edges)	1.387% (4 as a ← edge) (34 as a ↔ edge)	6.814% (484 as a ← edge) (119 as a ↔ edge)
APX	4.483 % (89,775 not valley-free)	5.330 % (146 as a ← edge)	9.458 % (837 as a ← edge)
DPP*	0.519% (10,391 not valley-free)	0.913% (25 as a ← edge)	6.915% (612 as a ← edge)
AHeu ($W = 10$)	0.483% (9,666 not valley-free)	0.292% (8 as a ← edge)	5.073% (449 as a ← edge)
AHeu ($W = 10$) readd ($k = 10$)	0.413% (8,278 not valley-free)	0.329% (9 as a ← edge)	5.469% (484 as a ← edge)

Table 6.6. Results for inferring customer-to-provider relationships.

In the Figures 6.7–6.10 pairs of diagrams are shown visualizing the observed correlations of various parameters and the amount of pre-knowledge. The upper diagram in each pair presents the statistics for all algorithms run on the reliable edge set and the lower diagram always presents the statistics for the full edge set. The single findings are:

– Figure 6.7 gives an impression on how much of the classification is already implied by he pre-knowledge. It shows the number of edges fixed by our preprocessing step on the reliable edge set as well as one the full edge set. Obviously, in both cases a small number of known edges already determines a rather large part of the final classification while on the other hand increasing the pre-knowledge seems to have little influence.
– Figure 6.8 shows the number of conflicts occurring during the preprocessing phase. Both diagrams demonstrate that providing additional knowledge might even lead to inconsistencies when using real-world data. This holds drastically for the full edge set due to its lower data quality.
– Figure 6.9 shows the number of misclassified customer-to-provider (C2P) edges. According to these plots the performance of our heuristic is hardly influenced by the amount of pre-knowledge available which we take as an indication for the high quality of the inferred results.
– Figure 6.10 shows the number of invalid paths. The diagrams strongly confirm the inertia observed from Figure 6.9 The lower diagram based on the full edge set illustrates nicely how our AHeu keeps giving good results even in the presence of (partially) invalid pre-knowledge while the other algorithms (due to the inflexible preprocessing step) have to trust this knowledge at the cost of a huge number of invalid paths.

6.7 Summary

In this chapter we studied algorithmic solutions for the Acyclic Type-of-Relationship problem. In particular we designed a linear-time algorithm for finding an acyclic and valley-free completion of a partial orientation of a set of AS paths, that only uses customer-to-provider relations for completion whereas the partial orientation can be expressed with arbitrary types of standard AS relationships. Based on some evident assumptions on the size of ASes, acyclicity conditions are given in terms of forbidden graph patterns.

To evaluate the quality of this approach we supplement the theoretical study with practical evidence for its feasibility and accuracy in large parts. The described, heuristic algorithm AHeu turned out to be easily implementable, fast, and flexible with respect to incorporating initial pre-knowledge. On a reliable data set, it outperformed most of the state-of-the-art algorithms proposed in the literature. Moreover, the acyclicity of all customer-to-provider relationships with in the reliable data set could be confirmed. These findings suggest to integrate acyclicity notions in detailed models of AS relationships.

On the other side, we have learned that acyclicity with respect to peer-to-peer relationships is not yet fully captured. The underlying assumption that the roughly-the-same-size relation is transitive seems too much a simplification. We consider finding a more accurate problem formulation involving acyclicity and peer-to-peer relationships as the main open issue for future research.

From a theoretical point of view, the most interesting open question is whether acyclicity of the Internet hierarchy can be deduced via a game-theoretic analysis. Are acyclically oriented AS graphs Nash equilibria for certain classes of network formation games (see, e.g., [51, 3, 83])?

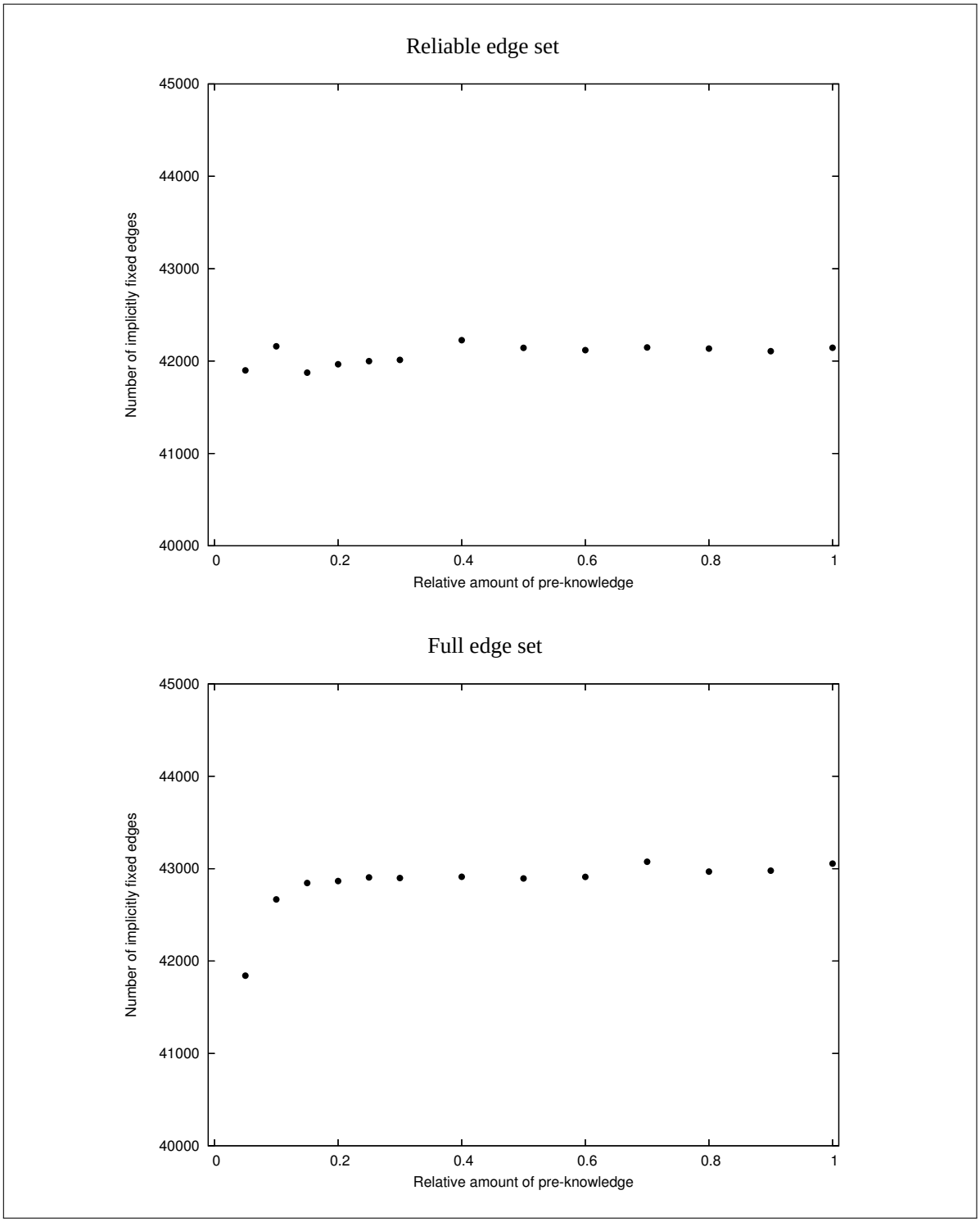


Figure 6.7. Number of edges implicitly fixed by pre-knowledge (on average over 5 random samples of the edge set)

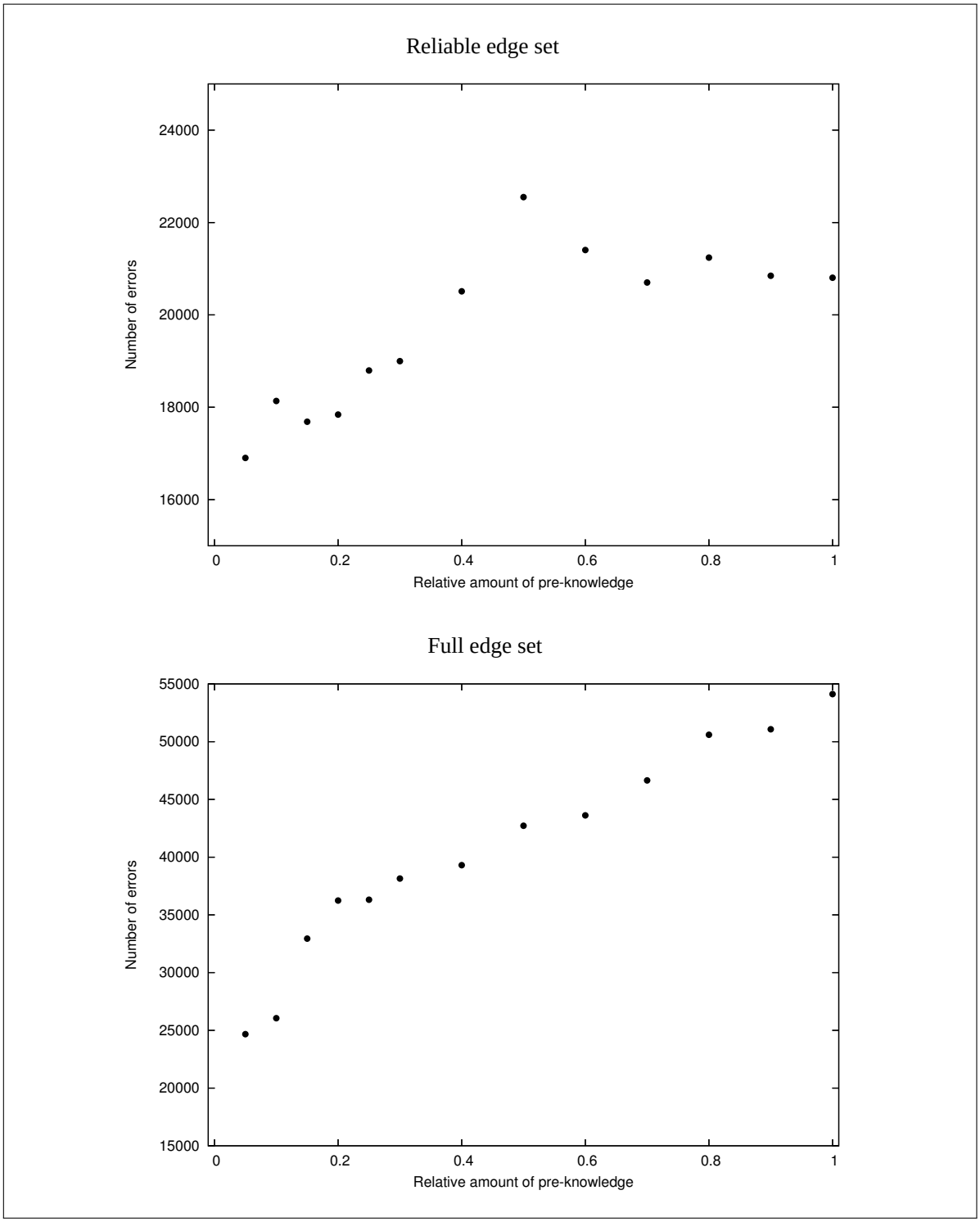


Figure 6.8. Number of errors encountered during inferring edges from pre-knowledge (on average over 5 random samples of the edge set)

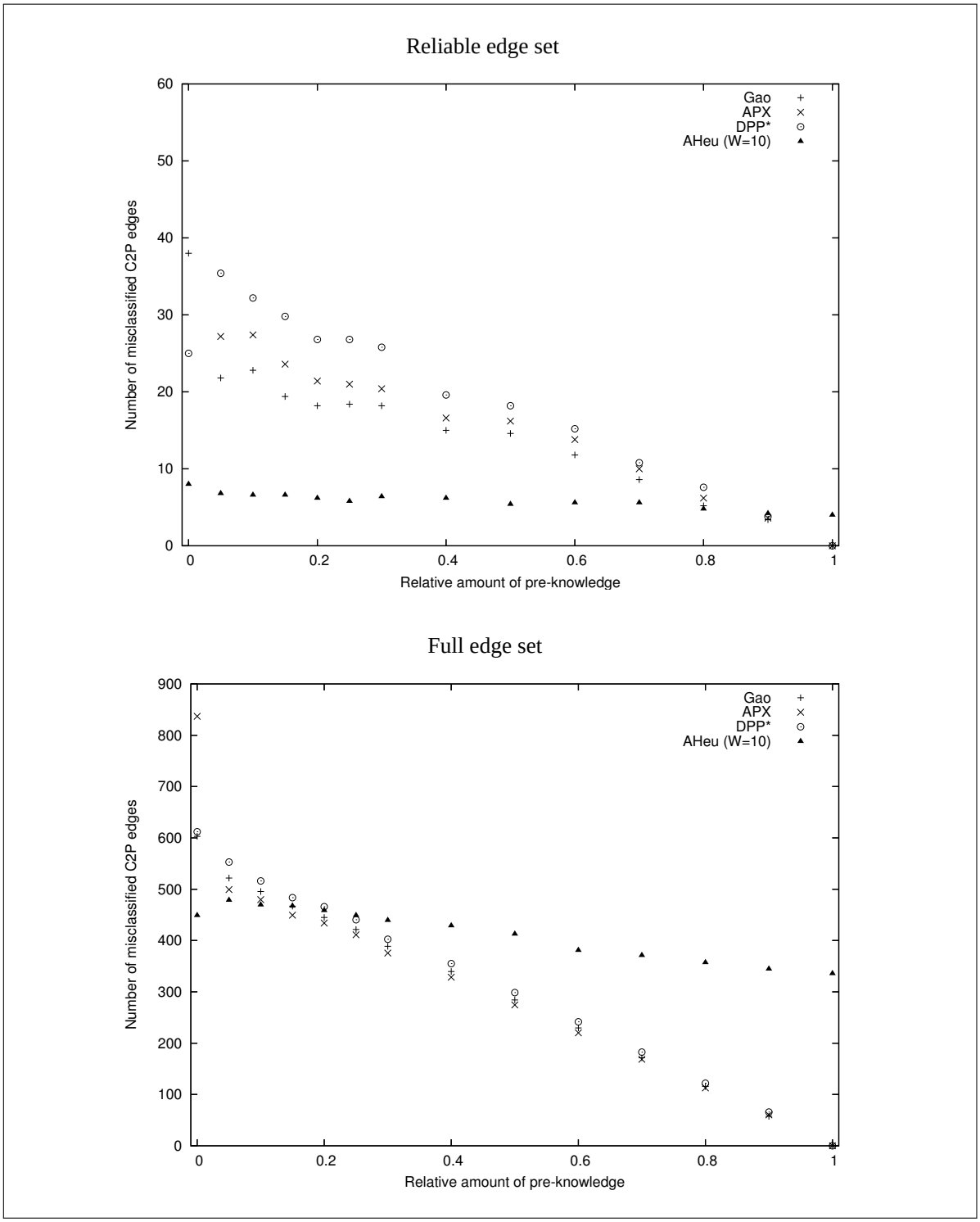


Figure 6.9. Number of misclassified edges for different amounts of pre-knowledge (on average over 5 random samples of the edge set)

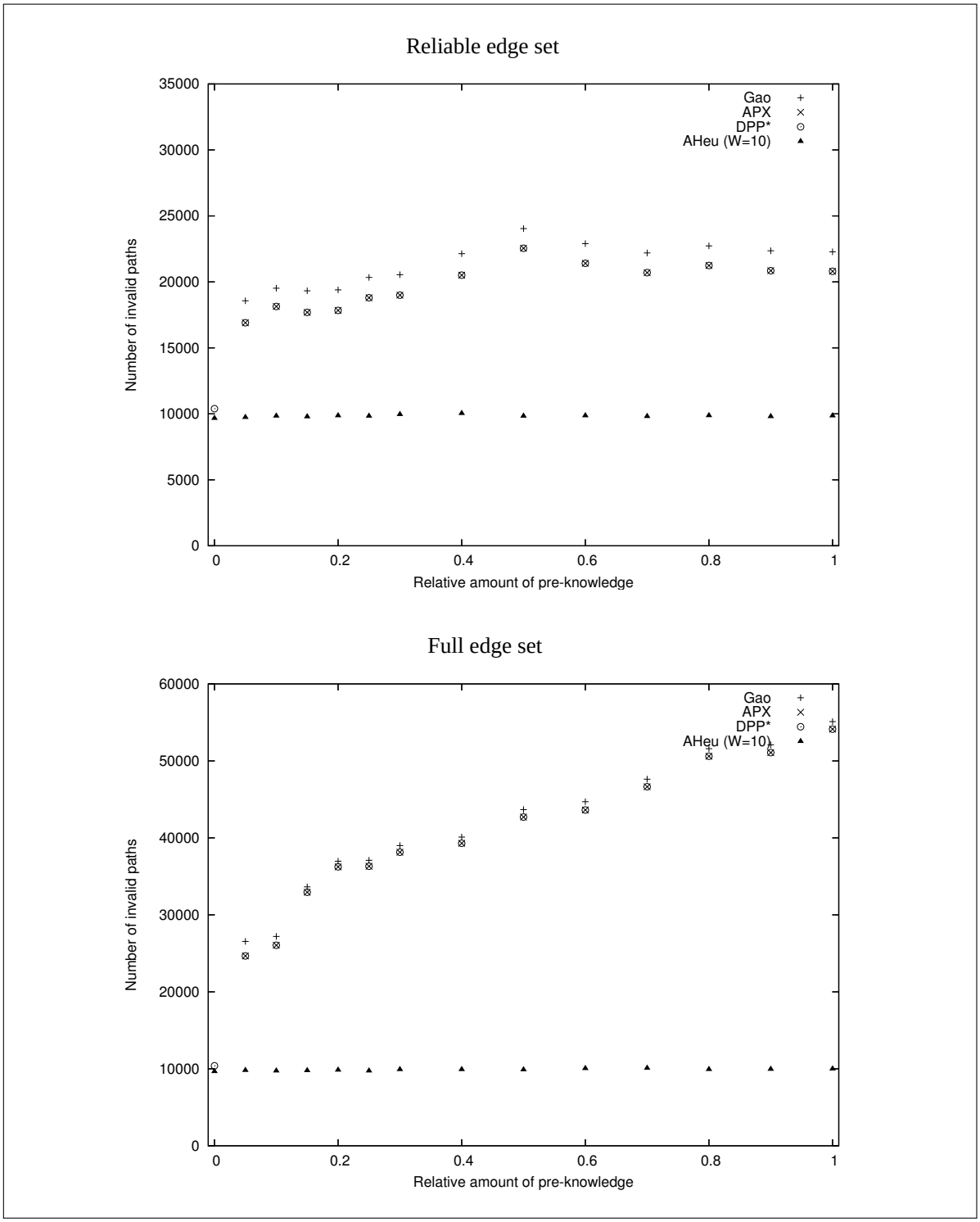


Figure 6.10. Number of invalid paths for different amounts of pre-knowledge (on average over 5 random samples of the edge set)

Bibliography

1. C. Alaettinoğlu. Scalable router configuration for the Internet. In *Proceedings of the 5th International Conference on Computer Communications and Networks (ICCCN'96)*. IEEE Computer Society Press, Washington, D.C., 1996.
2. C. Alaettinoğlu, C. Villamizar, E. Gerich, D. Kessens, D. Meyer, T. Bates, D. Karrenberg, and M. Terpstra. Routing Policy Specification Language (RPSL). RFC 2622, The Internet Society, 1999.
3. S. Albers, S. Eilts, E. Even-Dar, Y. Mansour, and L. Roditty. On Nash equilibria for a network creation game. In *Proceedings of the 17th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'06)*, pages 89–98. SIAM, Philadelphia, PA, 2006.
4. R. Albrightson, J. J. Garcia-Luna-Aceves, and J. Boyle. EIGRP - a fast routing protocol based on distance vectors. In *Proceedings of the NetWorld/Interop Engineer Conference (NetWorld/Interop'94)*, 1994.
5. S. Arnborg, D. G. Corneil, and A. Proskurowski. Complexity of finding embeddings in a k-tree. *SIAM Journal on Algebraic and Discrete Methods*, 8(2):277–284, 1987.
6. G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela, and M. Protasi. *Complexity and Approximation: Combinatorial Optimization Problems and Their Approximability Properties*. Springer-Verlag, Berlin, 1999.
7. N. A. Baas and T. Helvik. Higher order cellular automata. *Advances in Complex Systems*, 8(2–3):169–192, 2005.
8. C. L. Barrett, B. W. Bush, S. Kopp, H. S. Mortveit, and C. M. Reidys. Sequential dynamical systems and applications to simulations. In *Proceedings of the 33rd Annual Simulation Symposium (SS'00)*, pages 245–253. IEEE Computer Society Press, Washington, D.C., 2000.
9. C. L. Barrett, W. Y. C. Chen, and M. J. Zheng. Discrete dynamical systems on graphs and boolean functions. *Mathematics and Computers in Simulation*, 66(6):487–497, 2004.
10. C. L. Barrett, S. Eubank, M. V. Marathe, H. S. Mortveit, and C. M. Reidys. Science and engineering of large socio-technical simulations. In *Proceedings of the 2000 Western MultiConference on Computer Simulation (WMC'00)*. The Society for Computer Simulation International, San Diego, CA, 2000.
11. C. L. Barrett, H. B. Hunt III, M. V. Marathe, S. S. Ravi, D. J. Rosenkrantz, and R. E. Stearns. On some special classes of sequential dynamical systems. *Annals of Combinatorics*, 7(4):381–408, 2003.
12. C. L. Barrett, H. B. Hunt III, M. V. Marathe, S. S. Ravi, D. J. Rosenkrantz, and R. E. Stearns. Predecessor and permutation existence problems for sequential dynamical systems. In *Proceedings of the Conference on Discrete Models for Complex Systems*

(DMCS'03), volume AB of *Discrete Mathematics and Theoretical Computer Science Proceedings*, pages 69–80, 2003.

13. C. L. Barrett, H. B. Hunt III, M. V. Marathe, S. S. Ravi, D. J. Rosenkrantz, and R. E. Stearns. Reachability problems for sequential dynamical systems with threshold functions. *Theoretical Computer Science*, 295(1–3):41–64, 2003.
14. C. L. Barrett, H. B. Hunt III, M. V. Marathe, S. S. Ravi, D. J. Rosenkrantz, and R. E. Stearns. Complexity of reachability problems for finite discrete dynamical systems. *Journal of Computer and System Sciences*, 72(7):1317–1345, 2006.
15. C. L. Barrett, H. B. Hunt III, M. V. Marathe, S. S. Ravi, D. J. Rosenkrantz, R. E. Stearns, and P. T. Tošić. Gardens of Eden and fixed points in sequential dynamical systems. In *Proceedings of the 1st International Conference on Discrete Models: Combinatorics, Computation and Geometry (DM-CCG'01)*, volume AA of *Discrete Mathematics and Theoretical Computer Science Proceedings*, pages 241–259, 2001.
16. C. L. Barrett, H. S. Mortveit, and C. M. Reidys. Elements of a theory of computer simulation II: Sequential dynamical systems. *Applied Mathematics and Computation*, 107(2–3):121–136, 2000.
17. C. L. Barrett, H. S. Mortveit, and C. M. Reidys. Elements of a theory of computer simulation III: Equivalence of SDS. *Applied Mathematics and Computation*, 122(3):325–340, 2001.
18. C. L. Barrett, H. S. Mortveit, and C. M. Reidys. ETS IV: Sequential dynamical systems: fixed points, invertibility and equivalence. *Applied Mathematics and Computation*, 134(1):153–171, 2003.
19. C. L. Barrett and C. M. Reidys. Elements of a theory of computer simulation I: Sequential CA over random graphs. *Applied Mathematics and Computation*, 98(2–3):241–259, 1999.
20. C. Bergman, D. Juedes, and G. Slutzki. Computational complexity of term equivalence. *International Journal of Algebra and Computation*, 9(1):113–128, 1999.
21. H. L. Bodlaender. A linear-time algorithm for finding tree-decomposition of small treewidth. *SIAM Journal on Computing*, 25(6):1305–1317, 1996.
22. V. G. Bodnarchuk, L. A. Kaluzhnin, V. N. Kotov, and B. A. Romov. Galois theory for Post algebras. I. *Kibernetika*, 3:1–10, 1969. In Russian.
23. V. G. Bodnarchuk, L. A. Kaluzhnin, V. N. Kotov, and B. A. Romov. Galois theory for Post algebras. II. *Kibernetika*, 5:1–9, 1969. In Russian.
24. E. Böhler, N. Creignou, S. Reith, and H. Vollmer. Playing with Boolean blocks, part I: Post's lattice with applications to complexity theory. *ACM SIGACT News*, 34(4):38–52, 2003.
25. E. Böhler, N. Creignou, S. Reith, and H. Vollmer. Playing with Boolean blocks, part II: Constraint satisfaction problems. *ACM SIGACT News*, 35(1):22–35, 2004.
26. E. Böhler and H. Schnoor. The complexity of descriptiveness of boolean circuits over different sets of gates. *Theory of Computing Systems*, 2007. To appear.
27. U. Brandes and T. Erlebach, editors. *Network Analysis: Methodological Foundations*, volume 3418 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, 2005.
28. S. R. Buss, C. H. Papadimitriou, and J. N. Tsitsiklis. On the predictability of coupled automata: An allegory about chaos. *Complex Systems*, 5:525–539, 1991.
29. R. W. Callon. Use of OSI IS-IS for routing in TCP/IP and dual environments. RFC 1195, The Internet Society, 1990.

30. K. Cattell and M. J. Dinneen. A characterization of graphs with vertex cover up to five. In *Proceeding of the International Workshop on Orders, Algorithms, and Applications (ORDAL'94)*, volume 831 of *Lecture Notes in Computer Science*, pages 86–99. Springer-Verlag, Berlin, 1994.
31. CEER European Regulator's Group for Electricity and Gas. ERGEG final report. The lessons to be learned from the large disturbance in the European power system on the 4th of November 2006. Document Reference E06-BAG-01-06, February 2007. `http://www.ergeg.org`.
32. H. Chang, R. Govindan, S. Jamin, S. J. Shenker, and W. Willinger. Towards capturing representative AS-level Internet topologies. *Computer Networks*, 44(6):737–755, 2004.
33. G. T. Chartrand and F. Harary. Planar permutation graphs. *Annales de l'Institut Henri Poincaré, section B*, 3(4):433–438, 1967.
34. C. Chau, R. Gibbens, and T. G. Griffin. Towards a unified theory of policy-based routing. In *Proceedings of the 25th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM'06)*. IEEE Computer Society Press, Washington, D.C., 2006.
35. T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. The MIT Press, Cambridge, MA, 2nd edition, 2001.
36. H. S. M. Coxeter and W. O. J. Moser. *Generators and Relations for Discrete Groups*. Springer-Verlag, New York, NY, 4th edition, 1980.
37. N. Creignou and M. Hermann. Complexity of generalized satisfiability counting problems. *Information and Computation*, 125(1):1–12, 1996.
38. B. A. Davey and H. A. Priestley. *Introduction to Lattices and Orders*. Cambridge University Press, Cambridge, 1990.
39. S. E. Deering and R. M. Hinden. Internet Protocol, version 6 (IPv6) specification. RFC 2460, The Internet Society, 1998.
40. G. Di Battista, T. Erlebach, A. Hall, M. Patrignani, M. Pizzonia, and T. Schank. Computing the types of the relationships between autonomous systems. *IEEE/ACM Transactions on Networking*, 2007.
41. G. Di Battista, M. Patrignani, and M. Pizzonia. Computing the types of the relationships between autonomous systems. In *Proceedings of the 22nd Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM'03)*, pages 156–165. IEEE Computer Society Press, Washington, D.C., 2003.
42. G. Di Battista, T. Refice, and M. Rimondini. How to extract BGP peering information from the Internet Routing Registry. In *Proceedings of the ACM SIGCOMM 2006 Workshop on Mining Network Data (MineNet'2006)*, pages 317–322. ACM Press, New York, NY, 2006.
43. R. Diestel. *Graph Theory*. Graduate Texts in Mathematics. Springer-Verlag, Berlin, 3rd edition, 2003.
44. X. A. Dimitropoulos, D. V. Krioukov, M. Fomenkov, B. Huffaker, Y. Hyun, K. C. Claffy, and G. F. Riley. AS relationships: Inference and validation. *ACM SIGCOMM Computer Communication Review*, 37(1):31–40, 2007.
45. X. A. Dimitropoulos, D. V. Krioukov, B. Huffaker, K. C. Claffy, and G. F. Riley. Inferring AS relationships: Dead end or lively beginning? In *Proceedings of the 4th International Workshop on Experimental and Efficient Algorithms (WEA'05)*, volume

3503 of *Lecture Notes in Computer Science*, pages 113–125. Springer-Verlag, Berlin, 2005.

46. M. J. Dinneen, K. Cattell, and M. R. Fellows. Forbidden minors to graphs with small feedback sets. *Discrete Mathematics*, 230:215–252, 2001.
47. R. J. Duffin. Topology of series-parallel networks. *Journal of Mathematical Analysis and Applications*, 10(2):303–318, 1965.
48. E.ON Netz GmbH. Report on the status of the investigations of the sequences of events and causes of the failure in the continental European electricity grid on Saturday, November 4, 2006, after 22:10 hours (Investigation status as at Tuesday, November 14, 2006, 10:00 hours). `http://www.eon-energie.com`.
49. T. Erlebach, A. Hall, A. Panconesi, and D. Vukadinović. Cuts and disjoint paths in the valley-free path model of Internet BGP routing. In *Proceedings of the 1st Workshop on Combinatorial and Algorithmic Aspects of Networking (CAAN'04)*, volume 3405 of *Lecture Notes in Computer Science*, pages 49–62. Springer-Verlag, Berlin, 2004.
50. T. Erlebach, A. Hall, and T. Schank. Classifying customer-provider relationships in the Internet. In *Proceedings of the IASTED International Conference on Communications and Computer Networks (CCN'02)*, pages 538–545. ACTA Press, Calgary, 2002.
51. A. Fabrikant, A. Luthra, E. Maneva, C. H. Papadimitriou, and S. J. Shenker. On a network creation game. In *Proceeding of the 22nd ACM Symposium on Principles of Distributed Computing (PODC'03)*, pages 347–351. ACM Press, New York, NY, 2003.
52. N. Feamster, R. Johari, and H. Balakrishnan. Implications of autonomy for the expressiveness of policy routing. In *Proceedings of the ACM SIGCOMM 2005 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM'05)*, pages 25–36. ACM Press, New York, NY, 2005.
53. J. Feigenbaum, D. R. Karger, V. S. Mirrokni, and R. Sami. Subjective-cost policy routing. In *Proceedings of the 1st International Workshop on Internet and Network Economics (WINE'05)*, volume 3828 of *Lecture Notes in Computer Science*, pages 174–183. Springer-Verlag, Berlin, 2005.
54. J. Feigenbaum, D. R. Karger, V. S. Mirrokni, and R. Sami. Mechanism design for policy routing. *Distributed Computing*, 18(4):293–305, 2006.
55. A. Feldmann, O. Maennel, Z. M. Mao, A. Berger, and B. Maggs. Locating Internet routing instabilities. In *Proceedings of the ACM SIGCOMM 2004 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM'04)*, pages 205–218. ACM Press, New York, NY, 2004.
56. J. Flum and M. Grohe. The parameterized complexity of counting problems. *SIAM Journal on Computing*, 33(4):892–922, 2004.
57. S. Fortune, J. E. Hopcroft, and J. Wyllie. The directed subgraph homeomorphism problem. *Theoretical Computer Science*, 10(2):111–121, 1980.
58. E. C. Freuder. Complexity of k-tree structured constraint satisfaction problems. In *Proceedings of the 8th National Conference on Artificial Intelligence (AAAI'90)*, pages 4–9. AAAI Press/The MIT Press, Menlo Park, CA, 1990.
59. V. Fuller, T. Li, J. J. Y. Yu, and K. Varadhan. Classless Inter-Domain Routing (CIDR): An address assignment and aggregation strategy. RFC 1519, The Internet Society, 1993.

60. N. Ganguly, B. K. Sidkar, A. Deutsch, G. Canright, and P. Pal Chaudhuri. A survey on cellular automata. Technical report, Zentrum für Informationsidenste und Hochleistungsrechnen, Technische Universität Dresden, December 2003.
61. L. Gao. On inferring autonomous system relationships in the Internet. *IEEE/ACM Transactions on Networking*, 9(6):733–745, 2001.
62. L. Gao and J. Rexford. Stable Internet routing without global coordination. *IEEE/ACM Transactions on Networking*, 9(6):681–692, 2001.
63. M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. A Series of Books in the Mathematical Sciences. W. H. Freeman and Company, New York, NY, 1979.
64. M. Garzon. *Models of Massive Parallelism: Analysis of Cellular Automata and Neural Networks*. Texts in Theoretical Computer Science. An EATCS Series. Springer-Verlag, Berlin, 1995.
65. Z. Ge, D. R. Figueiredo, S. Jaiswal, and L. Gao. On the hierarchical structure of the logical Internet graph. In *Proceedings of SPIE Scalability and Traffic Control in IP Networks*, volume 4235, pages 208–222. SPIE, Bellingham, WA, 2001.
66. D. Geiger. Closed systems of functions and predicates. *Pacific Journal of Mathematics*, 27(2):228–250, 1968.
67. R. Govindan, C. Alaettinoğlu, G. Eddy, D. Kessens, S. Kumar, and W.-S. Lee. An architecture for stable, analyzable Internet routing. *IEEE Network*, 13(1):29–35, 1999.
68. R. Govindan and A. Reddy. An analysis of Internet inter-domain topology and route stability. In *Proceedings of the 16th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM'97)*, pages 850–857. IEEE Computer Society Press, Washington, D.C., 1997.
69. G. A. Grätzer. *General Lattice Theory*. Akademie-Verlag, Berlin, 1978.
70. F. Green. NP-complete problems in cellular automata. *Complex Systems*, 1(3):453–474, 1987.
71. T. G. Griffin and B. J. Premore. An experimental analysis of BGP convergence time. In *Proceedings of the 9th Annual International Conference on Network Protocols (ICNP'01)*, pages 53–61. IEEE Computer Society Press, Washington, D.C., 2001.
72. T. G. Griffin, F. B. Shepherd, and G. T. Wilfong. The stable paths problem and interdomain routing. *IEEE/ACM Transactions on Networking*, 10(2):232–243, 2002.
73. T. G. Griffin and G. T. Wilfong. An analysis of BGP convergence properties. *ACM SIGCOMM Computer Communication Review*, 29(4):277–288, 1999.
74. J. Hawkinson and T. Bates. Guidelines for creation, selection, and registration of an autonomous system (AS). RFC 1930, The Internet Society, 1996.
75. C. Hedrick. Routing Information Protocol. RFC 1058, The Internet Society, 1988.
76. L. A. Hemaspaandra and M. Ogihara. *The Complexity Theory Companion*. Texts in Theoretical Computer Science. An EATCS Series. Springer-Verlag, Berlin, 2002.
77. J. E. Hopcroft, R. Motwani, and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley Publishing Co., Reading, MA, 2nd edition, 2001.
78. J. E. Hopcroft and R. E. Tarjan. Efficient planarity testing. *Journal of the ACM*, 21(4):549–568, 1974.

79. B. Huber, S. Leinen, R. O'Dell, and R. Wattenhofer. Inferring AS relationships beyond counting edges. Technical Report 446, Institut für Pervasive Computing, Eidgenössische Technische Hochschule Zürich, May 2004.
80. B. F. Hummel. Automata-based IP packet classification, 2006. Diplomarbeit, Fachbereich Informatik, Technische Universität München, München.
81. B. F. Hummel and S. Kosub. Acyclic type-of-relationship problems on the Internet: An experimental analysis. Technical Report TUM-I0709, Fakultät für Informatik, Technische Universität München, February 2007.
82. G. Huston. Interconnection, peering and settlements—Part II. *The Internet Protocol Journal*, 2(2):2–23, 1999.
83. R. Johari, S. Mannor, and J. N. Tsitsiklis. A contract-based model for directed network formation. *Games and Economic Behavior*, 56(2):201–224, 2006.
84. R. M. Karp. Reducibility among combinatorial problems. In R. E. Miller and J. W. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press, New York, NY, 1972.
85. R. M. Karp. On the complexity of combinatorial problems. *Networks*, 5(1–3):45–68, 1975.
86. T. Kernen. traceroute.org web site. `http://www.traceroute.org`.
87. N. G. Kinnersley. Obstruction set isolation for layout permutation problems, 1989. Ph.D. Thesis, Department of Computer Science, Washington State University, Pullmann, WA.
88. S. Kosub. Dichotomy results for fixed-point existence problems for boolean dynamical systems. Technical Report TUM-I0701, Fakultät für Informatik, Technische Universität München, January 2007.
89. S. Kosub and C. M. Homan. Dichotomy results for fixed point counting in boolean dynamical systems. Technical Report TUM-I0706, Fakultät für Informatik, Technische Universität München, January 2007.
90. S. Kosub, M. G. Maaß, and H. Täubig. Acyclic type-of-relationship problems on the Internet. In *Proceedings of the 3rd Workshop on Combinatorial and Algorithmic Aspects of Networking (CAAN'06)*, volume 4235 of *Lecture Notes in Computer Science*, pages 98–111. Springer-Verlag, Berlin, 2006.
91. S. Kosub, M. G. Maaß, and H. Täubig. Acyclic type-of-relationship problems on the Internet. Technical Report TUM-I0605, Fakultät für Informatik, Technische Universität München, March 2006.
92. K. Kuratowski. Sur le probleme des courbes gauches en topologie. *Fundamenta Mathematicae*, 16:271–283, 1930.
93. J. F. Kurose and K. W. Ross. *Computer Networking: A Top-Down Approach Featuring the Internet*. Addison-Wesley Longman, Amsterdam, 3rd edition, 2004.
94. C. Labovitz, A. Ahuja, A. Bose, and F. Jahanian. Delayed Internet routing convergence. *IEEE/ACM Transactions on Networking*, 9(3):293–306, 2001.
95. C. Labovitz, A. Ahuja, R. Wattenhofer, and S. Venkatachary. The impact of Internet policy and topology on delayed routing convergence. In *Proceedings of the 20th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM'01)*, pages 537–546. IEEE Computer Society Press, Washington, D.C., 2001.
96. C. Labovitz, G. R. Malan, and F. Jahanian. Origins of Internet routing instability. In *Proceedings of the 18th Annual Joint Conference of the IEEE Computer and Com-*

munications Societies (INFOCOM'99), pages 218–226. IEEE Computer Society Press, Washington, D.C., 1999.

97. M. Lad, A. Nanavati, D. Massey, and L. Zhang. An algorithmic approach to identifying link failures. In *Proceeding of the 10th IEEE Pacific Rim International Symposium on Dependable Computing (PRDC'04)*, pages 25–34. IEEE Computer Society Press, Washington, D.C., 2004.
98. R. Laubenbacher and B. Pareigis. Update schedules of sequential dynamical systems. *Discrete Applied Mathematics*, 154(6):980–994, 2006.
99. A. Lempel, S. Even, and I. Cederbaum. An algorithm for planarity testing of graphs. In *Proceedings of an International Symposium on Theory of Graphs*, pages 215–232. Gordon & Breach, New York, NY, 1967.
100. D. Lichtenstein. Planar formulae and their uses. *SIAM Journal on Computing*, 11(2):329–343, 1982.
101. N. Linial. Hard enumeration problems in geometry and combinatorics. *SIAM Journal on Algebraic and Discrete Methods*, 7(2):331–335, 1986.
102. O. Maennel and A. Feldmann. Realistic BGP traffic for test labs. In *Proceedings of the ACM SIGCOMM 2002 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM'02)*, pages 31–44. ACM Press, New York, NY, 2002.
103. G. S. Malkin. RIP version 2 - carrying additional information. RFC 1723, The Internet Society, 1994.
104. Z. M. Mao, R. Govindan, G. Varghese, and R. H. Katz. Route flap damping exacerbates Internet routing convergence. In *Proceedings of the ACM SIGCOMM 2002 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM'02)*, pages 221–233. ACM Press, New York, NY, 2002.
105. V. V. Martynyuk. Investigation of certain classes of functions in many-valued logics. *Problemy Kibertnetiki*, 3:49–60, 1960. In Russian.
106. K. Menger. The algebra of functions: past, present, future. *Rendiconti di Matematica e delle sue Applicazioni*, 20:409–430, 1961.
107. Merit Network Inc. Internet Routing Registry. `http://www.irr.net`.
108. D. Meyer, J. Schmitz, C. Orange, M. Prior, and C. Alaettinoğlu. Using RPSL in practice. RFC 2650, The Internet Society, 1999.
109. G. A. Mihram. Simulation methodology. *Theory and Decision*, 7(1–2):67–94, 1976.
110. Miniwatts Marketing Group. Internet world stats: Usage and population statistics. `http://www.internetworldstats.com`.
111. U. Montanari. Networks of constraints: Fundamental porperties and applications to picture processing. *Information Sciences*, 7:95–132, 1974.
112. C. Moore. Unpredictability and undecidability in dynamical systems. *Physical Review Letters*, 64(20):2354–2357, 1990.
113. C. Moore. Generalized shifts: Unpredictability and undecidability in dynamical systems. *Nonlinearity*, 4(2):199–230, 1991.
114. H. S. Mortveit and C. M. Reidys. Discrete, sequential dynamical systems. *Discrete Mathematics*, 226(1–3):281–295, 2001.
115. J. Moy. The OSPF specification. RFC 1131, The Internet Society, 1989.
116. J. Moy. OSPF version 2. RFC 2178, The Internet Society, 1997.

117. W. B. Norton. Internet Service Providers and peering. Equinix White Paper, Equinix, Inc., Foster City, CA, 2001.
118. D. Oran, editor. OSI IS-IS intra-domain routing protocol. RFC 1142, The Internet Society, 1990.
119. P. Pal Chaudhuri, D. R. Chowdhury, S. Nandi, and S. Chattopadhyay. *Additive Cellular Automata: Theory and Applications*, volume I. IEEE Computer Society Press, Washington, D.C., 1997.
120. C. H. Papadimitriou and M. Yannakakis. Optimization, approximation, and complexity classes. *Journal of Computer and System Sciences*, 43:425–440, 1991.
121. R. Pastor-Satorras and A. Vespignani. *Evolution and Structure of the Internet: A Statistical Physics Approach*. Cambridge University Press, Cambridge, 2004.
122. R. Poeschel and L. A. Kaluzhnin. *Funktionen- und Relationenalgebren. Ein Kapitel der Diskreten Mathematik*, volume 67 of *Lehrbücher und Monographien aus dem Gebiete der exakten Wissenschaften: Mathematische Reihe*. Birkhäuser Verlag, Basel, 1979.
123. E. L. Post. The two-valued iterative systems of mathematical logic. *Annals of Mathematical Studies*, 5:1–122, 1941.
124. J. B. Postel, editor. Internet Protocol – DARPA Internet program protocol specification. RFC 791, Information Sciences Institute, University of Southern California, Marina del Rey, CA, 1981.
125. J. B. Postel, editor. Transmission Control Protocol – DARPA Internet program protocol specification. RFC 793, Information Sciences Institute, University of Southern California, Marina del Rey, CA, 1981.
126. J. S. Provan and M. O. Ball. The complexity of counting cuts and of computing the probability that a graph is connected. *SIAM Journal on Computing*, 12(4):777–788, 1983.
127. Y. Rekhter and P. Gross. Application of the Border Gateway Protocol in the Internet. RFC 1772, The Internet Society, 1995.
128. Y. Rekhter and T. Li. A Border Gateway Protocol 4 (BGP-4). RFC 1771, The Internet Society, 1995.
129. Y. Rekhter and T. Li, editors. An architecture for IP address allocation with CIDR. RFC 1518, The Internet Society, 1993.
130. M. Rimondini, M. Pizzonia, G. Di Battista, and M. Patrignani. Algorithms for the inference of the commercial relationships between autonomous systems: Results analysis and model validation. In *Proceedings of the 2nd International Workshop on Inter-Domain Performance and Simulation (IPS'04)*. Budapesti Mûszaki és Gazdasgtudományi Egyetem, Budapest, 2004.
131. RIPE NCC. Routing Information Service (RIS). `http://www.ripe.net/ris/`.
132. N. Robertson and P. D. Seymour. Graph minors. V. Excluding a planar graph. *Journal of Combinatorial Theory, Series B*, 41(1):92–114, 1986.
133. N. Robertson and P. D. Seymour. Graph minors. XIII. The disjoint path problem. *Journal of Combinatorial Theory, Series B*, 63(1):65–110, 1995.
134. N. Robertson and P. D. Seymour. Graph minors. XX. Wagner's conjecture. *Journal of Combinatorial Theory, Series B*, 92(2):325–357, 2004.
135. I. G. Rosenberg. Über die funktionale Vollständigkeit in den mehrwertigen Logiken. *Rozpravy ČSAV Řada Mat. Přir. Věd., Praha*, 80(4):3–93, 1970.

136. S. Savage, A. Collins, E. Hoffman, J. Snell, and T. E. Anderson. The end-to-end effects of Internet path selection. In *Proceedings of the ACM SIGCOMM 1999 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM'99)*, pages 289–299. ACM Press, New York, NY, 1999.
137. P. Schofield. Complete subsets of mappings over a finite domain. *Proceedings of the Cambridge Philosophical Society*, 62(4):597–611, 1966.
138. G. Siganos and M. Faloutsos. Analyzing BGP policies: Methodology and tool. In *Proceedings of the 23rd Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM'04)*, pages 1640–1651. IEEE Computer Society Press, Washington, D.C., 2004.
139. H. A. Simon. The architecture of complexity. *Proceedings of the American Philosophical Society*, 106:467–482, 1962.
140. L. Subramanian, S. Agarwal, J. Rexford, and R. H. Katz. Characterizing the Internet hierarchy from multiple vantage points. In *Proceedings of the 21st Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM'02)*, pages 618–627. IEEE Computer Society Press, Washington, D.C., 2002.
141. K. Sutner. On the computational complexity of finite cellular automata. *Journal of Computer and System Sciences*, 50(1):87–97, 1995.
142. A. S. Tanenbaum. *Computer Networks*. Prentice Hall, Upper Saddle River, NJ, 4th edition, 2002.
143. H. Tangmunarunkit, J. Doyle, R. Govindan, S. Jamin, S. J. Shenker, and W. Willinger. Does AS size determine degree in AS topology? *ACM SIGCOMM Computer Communication Review*, 31(5):7–10, 2001.
144. H. Tangmunarunkit, R. Govindan, S. J. Shenker, and D. Estrin. The impact of routing policy on Internet paths. In *Proceedings of the 20th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM'01)*, pages 736–742. IEEE Computer Society Press, Washington, D.C., 2001.
145. P. T. Tošić. On complexity of counting fixed point configurations in certain classes of graph automata. *Electronic Colloquium on Computational Complexity*, 12(51), 2005.
146. P. T. Tošić. Computational complexity of some enumeration problems about uniformely sparse boolean network automata. In *Proceedings of the 2nd European Conference on Complex Systems (ECCS'06)*. European Complex Systems Society, 2006.
147. P. T. Tošić. On the complexity of counting fixed points and gardens of Eden in sequential dynamical systems on planar bipartite graphs. *International Journal of Foundations of Computer Science*, 17(5):1179–1203, 2006.
148. P. T. Tošić and G. A. Agha. On computational complexity of counting fixed points in symmetric boolean graph automata. In *Proceedings of the 4th International Conference on Unconventional Computation (UC'05)*, volume 3699 of *Lecture Notes in Computer Science*, pages 191–205. Springer-Verlag, Berlin, 2005.
149. P. Traina, R. Chandra, and T. Li. BGP community attribute. RFC 1997, The Internet Society, 1996.
150. University of Oregon. Route Views project page. `http://www.routeviews.org`.
151. S. P. Vadhan. The complexity of counting in sparse, regular, and planar graphs. *SIAM Journal on Computing*, 31(2):398–427, 2002.
152. L. G. Valiant. The complexity of computing the permanent. *Theoretical Computer Science*, 8(2):189–201, 1979.

153. L. G. Valiant. The complexity of enumeration and reliability problems. *SIAM Journal on Computing*, 8(3):410–421, 1979.
154. I. van Beijnum. *BGP*. O'Reilly & Associates, Sebastopol, CA, 2002.
155. G. Varghese. *Network Algorithmics: An Interdisciplinary Approach to Designing Fast Networked Devices*. Morgan Kaufmann Publishers, San Francisco, CA, 2005.
156. C. Villamizar, R. Chandra, and R. Govindan. BGP route flap damping. RFC 2439, The Internet Society, 1998.
157. H. Vollmer. The complexity of deciding if a boolean function can be computed by circuits over a restricted basis. Manuscript, 2007.
158. J. von Neumann. *Theory of Self-Reproducing Automata*. Arthur W. Burks (ed.). University of Illinois Press, Champaign, IL, 1966.
159. G. Voutsadakis. Threshold agent networks: An approach to modelling and simulation. *Applied Mathematics and Computation*, 142(2–3):521–543, 2003.
160. K. Wagner. Über eine Erweiterung des Satzes von Kuratowski. *Deutsche Mathematik*, 2:280–285, 1937.
161. W. Weaver. Science and complexity. *American Scientist*, 36:536–544, 1948.
162. S. Wolfram. Twenty problems in the theory of cellular automata. *Physica Scripta*, T9:170–183, 1985.
163. S. Wolfram. *Cellular Automata and Complexity. Collected Papers*. Addison-Wesley Publishing Co., Reading, MA, 1994.
164. S. Wolfram. *A New Kind of Science*. Wolfram Media, Champaign, IL, 1994.
165. J. Xia and L. Gao. On the evaluation of AS relationship inferences. In *Proceedings of the 47th Annual IEEE Global Telecommunications Conference (Globecom'04)*, volume 3, pages 1373–1377. IEEE Communication Society, New York, NY, 2004.
166. S. V. Yablonskij. Functional constructions in a k-valued logic. *Trudy Matematicheskogo Instituta imeni V. A. Steklova*, 51:5–142, 1958. In Russian.