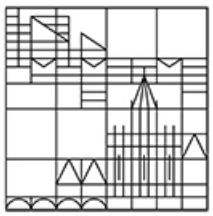
Fachbereich Informatik und
Informationswissenschaft

Universität
Konstanz



Lecture Notes

Network Dynamics

taught in Winter term 2010

by

Sven Kosub

November 3, 2010

Version v0.6

Contents

Networks 1

1.1 Networks in a Static Perspective

1.1.1 Population

An *actor* x is a variable with values in a set D called the *attribute type*. If x takes a concrete value $z \in D$, then z is the *state* of x.

Definition 1.1 *1. A* population $X = \{x_1, \ldots, x_n\}$ *is a finite set of actors* $x_1, \ldots x_n$ *with attribute types* $D_1, \ldots D_n$.

2. A population $X = \{x_1, \ldots, x_n\}$ *with attribute types* $D_1 \ldots, D_n$ *is said to be* homogeneous *with attribute type* D *if* $D_1 = \cdots = D_n = D$.

In the following we restrict ourselves to homogeneous populations. Therefore we omit the word "homogeneous."

Let $X = \{x_1, \ldots x_n\}$ be a population with attribute type D. A *configuration* (assignment, interpretation) is a mapping $I : X \to D$. A configuration I assigns a state $I(x) \in D$ to each actor $x \in X$. We also refer to a tuple $(z_1, \ldots, z_n)$ as a configuration.

1.1.2 Structure

The fundamental relation in network analysis is the *dyad*. A dyad relates two actors of a population. We use graph theory to descrive dyadic structures of populations.

Definition 1.2 *Let* $X = \{x_1, \ldots, x_n\}$ *be a population with attribute type* D.

1. A structure *is a set* $E \subseteq X \times X$.

2. The elements of a structure E *are called* dyads *(or* edges*).*

Structures can be directed, undirected, or mixed. Structures are allowed to have *annotations* (weights) with certain attribute types. That is, a structure E may be equipped with a weight function $w : E \to A$ where A is the attribute type.

1.1.3 Constraint

Constraints (conditions, invariants) limit the set of possible configurations of populations.

Definition 1.3 *Let $X = \{x_1, \ldots, x_n\}$ be a population of attribute type D. A relation $R \subseteq D^n$ is called a* constraint on X.

A configuration $I : X \to D$ is said to be *admissible* (with respect to constraint R) if and only if $(I(x_1), \ldots I(x_n)) \in R$.

Example: We discuss constraints for some populations and structures.

- Let $X = \{1, 2, 3, 4\}$ be a population of type $D = \{0, 1\}$. Let $E = C_4$ be a cycle containing the four actors. Define R to be the following constraint:

  $$R =_{\text{def}} \{\ (0,0,0,0), (0,1,0,1), (1,0,1,0), (1,1,1,1)\ \}$$

  Then, constraint R corresponds to the set of solution of the following set of equations:

  $$\begin{aligned} x_1 &= x_4 \oplus x_1 \oplus x_2 \\ x_2 &= x_1 \oplus x_2 \oplus x_3 \\ x_3 &= x_2 \oplus x_3 \oplus x_4 \\ x_4 &= x_3 \oplus x_4 \oplus x_1 \end{aligned}$$

  Here, $\oplus$ denotes XOR. Clearly, x_1 does not directly depend on x_3 and x_2 does not directly depend on x_4. All other dependencies between the variable are represented by an edge in the structure. Therefore, we call E the interdependence structure for R.
- Again, let $X = \{1, 2, 3, 4\}$ be a population of type $D = \{0, 1\}$ and let $E = C_4$ be a cycle containing the four actors. In contrast to the first example, suppose constraint R is given by the following set of equations:

  $$\begin{aligned} x_1 &= \overline{x_4} \wedge \overline{x_1} \wedge \overline{x_2} \\ x_2 &= \overline{x_1} \wedge \overline{x_2} \wedge \overline{x_3} \\ x_3 &= \overline{x_2} \wedge \overline{x_3} \wedge \overline{x_4} \\ x_4 &= \overline{x_3} \wedge \overline{x_4} \wedge \overline{x_1} \end{aligned}$$

  It is easily seen that in fact, R is empty. Thus, there is no admissible configuration.
- Let $X = \{x_1, \ldots, x_n\}$ be a population of type $D = \mathbb{R}$, and let E be unspecified. Then, R could be defined to be the convex polytope for the linear inequality constraints $A \cdot z \leq b$ such that $A \in \mathbb{R}^{n \times n}, b \in \mathbb{R}^n$ and $z \geq 0$.

1.1.4 System

Definition 1.4 *Let D be an attribute type. A (homogeneous) system S is a triple (X, E, R) such that X is a population with attribute type D, E is a structure on X, and R is a constraint on X.*

> **Example:** $(\{1,2,3,4\}, C_4, \{(0,0,0,0),(0,1,0,1),(1,0,1,0),(1,1,1,1)\})$ as in the example abve is a system.

A *state* of S is any admissible configuration $I : X \to D$.

1.2 Networks in a Dynamical Perspective

1.2.1 Process

Let $[h] = \{1, 2, \ldots, h\}$ for any $h \in \mathbb{N}_+$. We also allow h to be ∞ and define $[\infty] = \mathbb{N}_+$. The parameter h represents the observational horizon of a process.

Definition 1.5 *A* process *P is any finite or infinite sequence $(S_i)_{i\in[h]}$ of systems S_i.*

We also denote a process $P = (X_i, E_i, R_i)_{i\in[h]}$ when referring to the components of the systems.

Basically, we can identify three (non-excluding) important subtypes of processes:

- A process of type $(X_i, \emptyset, \emptyset)_{i\in[h]}$ is called a *population process.*
- A process of type $(X_i, E_i, \emptyset)_{i\in[h]}$ is called a *structure process.*
- A process of type $(X, E, R_i)_{i\in[h]}$ is called a *state process.*

In this course the focus is solely on *state processes.*

1.2.2 Trajectory

Systems running through a process can take several paths of state changes depending, e.g., on initial or environmental conditions. Such paths are called trajectories.

Definition 1.6 *Let $(X, E, R_i)_{i\in[h]}$ be any state process with a population of size n. A* trajectory *is a finite or infinite sequence $(I_j)_{j\in[h]}$ of admissible configurations, i.e., $(I_j(x_1), \ldots, I_j(x_n)) \in R_j$ for all $j \in [h]$.*

The set of possible trajectories of a process $(X, E, R_i)_{i\in[h]}$ is $\bigtimes_{i=0}^{h} R_i$ and can thus be, in the case $h = \infty$, uncountable, even for a binary attribute type. A large fraction of these trajectories is certainly not realistic.

1.2.3 Global State Dynamic

A global (deterministic) dynamic is a mechanism for selecting trajectories of a process. We focus on state dynamics.

Definition 1.7 *Let $P = (X, E, R_i)_{i\in[h]}$ be a state process with attribute type D and a population of size n.*

1. *A (global)* state dynamic *is a mapping $\mathbf{F} : D^n \times [h-1] \to D^n$.*
2. *A state dynamic f is said to be* compatible with *P if $\mathbf{F}(R_1, t) \subseteq R_{t+1}$ for all $t \in [h-1]$.*

A global state dynamic $\mathbf{F}$ produces the trajectory $(I, \mathbf{F}(I,1), \mathbf{F}(I,2), \ldots)$ depending on the initial configuration I.

Example: Let $P_1 = (X, E, R_i)_{i\in[h]}$ and $P_2 = (X, E, R_i)_{i\in[h]}$ be state processes of type $D = \{0,1\}$, population $X = \{1,2,3\}$, structure $E = K^3$, and constraints $R_i^1 = D^3$, $R_i^2 = \{(z_1,z_2,z_3) \in D^3 \mid 1 \le z_1+z_2+z_3 \le 2\ \}$. Consider the following two global state dynamics $\mathbf{F}_1$ and $\mathbf{F}_2$:

$$\mathbf{F}_1 \ : \ (z_1,z_2,z_3,t) \ \mapsto \ (1-z_1, 1-z_2, 1-z_3)$$

$$\mathbf{F}_2 \ : \ (z_1,z_2,z_3,t) \ \mapsto \ \begin{cases} (1-z_1,z_2,z_3) & \text{if } \mathrm{mod}(t-1,3) = 0 \\ (z_1,1-z_2,z_3) & \text{if } \mathrm{mod}(t-1,3) = 1 \\ (z_1,z_2,1-z_3) & \text{if } \mathrm{mod}(t-1,3) = 2 \end{cases}$$

Note that $\mathbf{F}_1$ does not depend on parameter t. For $h = 5$, it produces the following set of trajectories (the rows of the table):

(z_1,z_2,z_3)	$\mathbf{F}_1(z_1,z_2,z_3,1)$	$\mathbf{F}_1(z_1,z_2,z_3,2)$	$\mathbf{F}_1(z_1,z_2,z_3,3)$	$\mathbf{F}_1(z_1,z_2,z_3,4)$
$(0,0,0)$	$(1,1,1)$	$(1,1,1)$	$(1,1,1)$	$(1,1,1)$
$(0,0,1)$	$(1,1,0)$	$(1,1,0)$	$(1,1,0)$	$(1,1,0)$
$(0,1,0)$	$(1,0,1)$	$(1,0,1)$	$(1,0,1)$	$(1,0,1)$
$(0,1,1)$	$(1,0,0)$	$(1,0,0)$	$(1,0,0)$	$(1,0,0)$
$(1,0,0)$	$(0,1,1)$	$(0,1,1)$	$(0,1,1)$	$(0,1,1)$
$(1,0,1)$	$(0,1,0)$	$(0,1,0)$	$(0,1,0)$	$(0,1,0)$
$(1,1,0)$	$(0,0,1)$	$(0,0,1)$	$(0,0,1)$	$(0,0,1)$
$(1,1,1)$	$(0,0,0)$	$(0,0,0)$	$(0,0,0)$	$(0,0,0)$

It is obvious that $\mathbf{F}_1$ is compatible with both processes P_1 and P_2 (for each choice of h).

In contrast, $\mathbf{F}_2$ produces the following set of trajectories for $h = 5$.

(z_1, z_2, z_3)	$\mathbf{F}_2(z_1, z_2, z_3, 1)$	$\mathbf{F}_2(z_1, z_2, z_3, 2)$	$\mathbf{F}_2(z_1, z_2, z_3, 3)$	$\mathbf{F}_2(z_1, z_2, z_3, 4)$
$(0,0,0)$	$(1,0,0)$	$(0,1,0)$	$(0,0,1)$	$(1,0,0)$
$(0,0,1)$	$(1,0,1)$	$(0,1,1)$	$(0,0,0)$	$(1,0,1)$
$(0,1,0)$	$(1,1,0)$	$(0,1,0)$	$(0,1,1)$	$(1,1,0)$
$(0,1,1)$	$(1,1,1)$	$(0,1,1)$	$(0,1,0)$	$(1,1,1)$
$(1,0,0)$	$(0,1,1)$	$(1,0,0)$	$(1,0,1)$	$(0,0,0)$
$(1,0,1)$	$(0,0,1)$	$(1,0,1)$	$(1,0,0)$	$(0,0,1)$
$(1,1,0)$	$(0,1,0)$	$(1,0,0)$	$(1,1,1)$	$(0,1,0)$
$(1,1,1)$	$(0,1,1)$	$(1,0,1)$	$(1,1,0)$	$(0,1,1)$

We observe that $\mathbf{F}_2$ is compatible with P_1 but is compatible with P_2, since, e.g., $\mathbf{F}_2(0,1,1,1) = (1,1,1) \notin R_2^2$ although $(0,1,1) \in R_1^2$.

1.2.4 Local State Dynamic

Local dynamics separate the state-change information for each actor and the environmental information of global dynamics.

Definition 1.8 *Let $P = (X, E, R_i)_{i\in[h]}$ be a state process with a population of type D and size n.*

1. *A* local transition function on X *(or* local transition*) is a mapping $f : D^n \to D$.*
2. *An* update schedule on X *is a mapping $\alpha : [h-1] \to \mathcal{P}(X)$.*
3. *A* local state dynamic on X *is a pair (F, α) such that $F = \{f_1, \ldots, f_n\}$ is a set of local transitions on X, where f_i is associated with actor $x_i \in X$, and α is an update schedule on X.*

Example: Let P be the process given by $X = \{1,2,3\}$ with $D = \{0,1\}$, $E = K^3$ (the interdepence structure), and $R_i = D^3$ for all $i \in [\infty] = \mathbb{N}_+$. We consider the following two local state dynamics (F, α_1) and (F, α_2):

- $F = \{f_1, f_2, f_3\}$ consists of the local transitions f_i defined for $i \in \{1,2,3\}$ and $x_1, x_2, x_3 \in \{0,1\}$ by

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

- update schedule $\alpha_1 : \mathbb{N}_+ \to \mathcal{P}(X) : t \mapsto \{1,2,3\}$

- update schedule $\alpha_2 : \mathbb{N}_+ \to \mathcal{P}(X)$ is defined as follows:

$$\alpha_2 : t \mapsto \begin{cases} \{2\} & \text{if } \mathrm{mod}(t-1,3) = 0 \\ \{1\} & \text{if } \mathrm{mod}(t-1,3) = 1 \\ \{3\} & \text{if } \mathrm{mod}(t-1,3) = 2 \end{cases}$$

The update schedule α_1 is an example of a synchronous update schedule; α_2 is an example of a sequential update schedule.

Local state dynamics induce global state dynamics in the following way.

Definition 1.9 *Let $P = (X, E, R_i)_{i\in[h]}$ be a state process with a population of type D and size n. Let (F, α) be a local state dynamic on X.*

1. *For each actor $x_i \in X$ and for a each subset $U \subseteq X$ of actors, the* activity function *$\varphi_i[U]$ is defined for a configuration $\vec{z} = (z_1, \ldots, z_n) \in D^n$ by*

$$\varphi_i[U](\vec{z}) =_{\mathrm{def}} \begin{cases} f_i(z_1, \ldots, z_n) & \text{if } x_i \in U \\ z_i & \text{if } x_i \notin U \end{cases}$$

2. *For a set $U \subseteq X$ the* global transition function *$\mathbf{F}[U] : D^n \to D^n$ is defined by*

$$\mathbf{F}_F[U](\vec{z}) =_{\mathrm{def}} (\varphi_1[U](\vec{z}), \ldots, \varphi_n[U](\vec{z})).$$

3. *The global state dynamic $\mathbf{F}_{(F,\alpha)} : D^n \to D^n$ induced by the local state dynamic (F, α) is defined by*

$$\mathbf{F}_{(F,\alpha)}(\cdot, t) =_{\mathrm{def}} \prod_{k=1}^{t} \mathbf{F}_F[\alpha(k)],$$

i.e., by the composition of global transition functions specified by the update schedule.

Note that our usage of $f \cdot g$ is $(f \cdot g)(x) = g(f(x))$.

Note also that activity functions and global transition functions do not depend on schedules.

Example: We continue the example of local state dynamics discussed above. Let $U_1 = \{1, 2\}$ and $U_2 = \{1, 2, 3\}$ be subsets of populations. Then we obtain the following activity functions:

- $\varphi_1[U_1] = f_1$, $\varphi_2[U_1] = f_2$, and $\varphi_3[U_1] = \mathrm{id}$;
- $\varphi_1[U_2] = f_1$, $\varphi_2[U_2] = f_2$, and $\varphi_3[U_2] = f_3$.

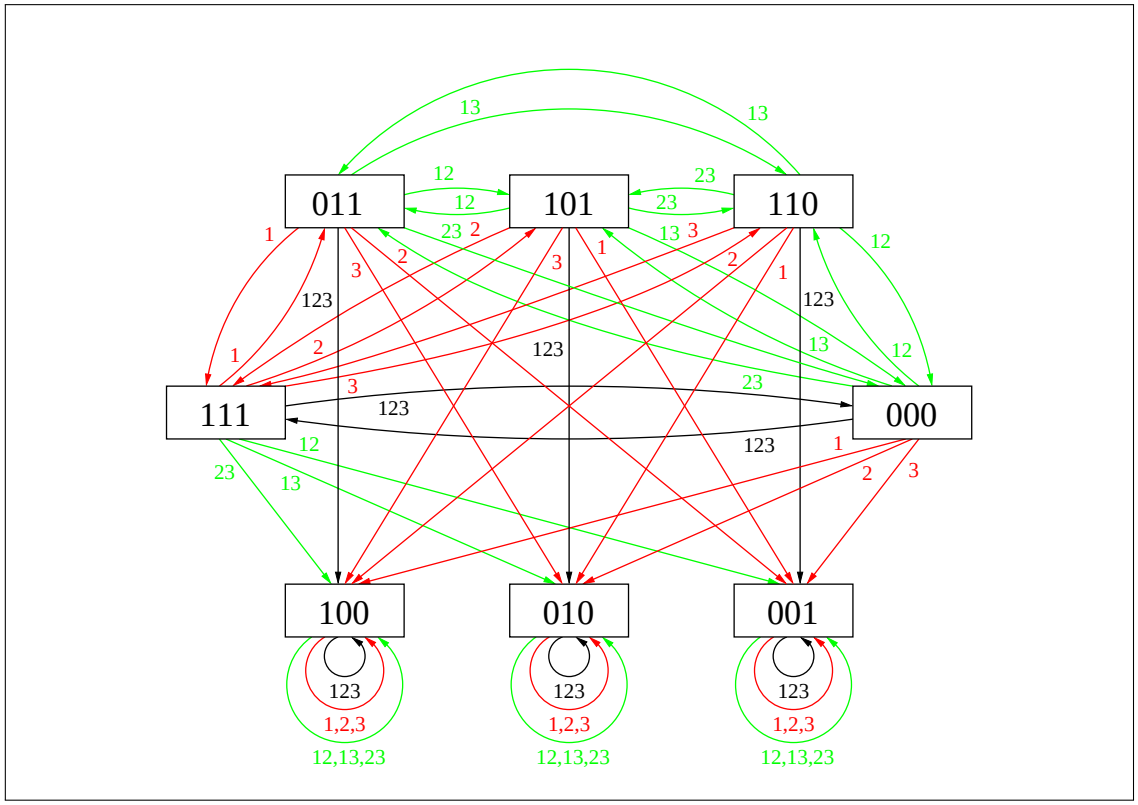


Figure 1.1: Global transition function

The global transition function looks as follows:

- $\mathbf{F}_F[U_1](z_1, z_2, z_3) = (f_1(z_1, z_2, z_3), f_2(z_1, z_2, z_3), z_3)$; concrete function values are, e.g.,

$$\begin{aligned} \mathbf{F}_F[U_1](1,1,1) &= (0,0,1) \\ \mathbf{F}_F[U_1](1,0,1) &= (0,1,1) \\ \mathbf{F}_F[U_1](0,0,1) &= (0,0,1) \end{aligned}$$

- $\mathbf{F}_F[U_2](z_1, z_2, z_3) = (f_1(z_1, z_2, z_3), f_2(z_1, z_2, z_3), f_3(z_1, z_2, z_3)$; concrete values are, e.g.,

$$\begin{aligned} \mathbf{F}_F[U_2](1,1,1) &= (0,0,0) \\ \mathbf{F}_F[U_2](0,0,0) &= (1,1,1) \end{aligned}$$

Figure 1.1 visualizes the global transition functions for $F = \{f_1, f_2, f_3\}$ completely. The global state dynamic induced by (F, α_1) is as follows:

(z_1, z_2, z_3)	$\{1,2,3\}$	$\{1,2,3\}$	$\{1,2,3\}$	$\{1,2,3\}$	...
(0,0,0)	(1,1,1)	(0,0,0)	(1,1,1)	(0,0,0)	...
(0,0,1)	(0,0,1)	(0,0,1)	(0,0,1)	(0,0,1)	...
(0,1,0)	(0,1,0)	(0,1,0)	(0,1,0)	(0,1,0)	...
(0,1,1)	(1,0,0)	(1,0,0)	(1,0,0)	(1,0,0)	...
(1,0,0)	(1,0,0)	(1,0,0)	(1,0,0)	(1,0,0)	...
(1,0,1)	(0,1,0)	(0,1,0)	(0,1,0)	(0,1,0)	...
(1,1,0)	(0,0,1)	(0,0,1)	(0,0,1)	(0,0,1)	...
(1,1,1)	(0,0,0)	(1,1,1)	(0,0,0)	(0,0,0)	...

That is, (F, α_1) can generate oscillating trajectories.

The global state dynamic induced by (F, α_2) is as follows:

(z_1, z_2, z_3)	$\{2\}$	$\{1\}$	$\{3\}$	$\{2\}$	...
(0,0,0)	(0,1,0)	(0,1,0)	(0,1,0)	(0,1,0)	...
(0,0,1)	(0,0,1)	(0,0,1)	(0,0,1)	(0,0,1)	...
(0,1,0)	(0,1,0)	(0,1,0)	(0,1,0)	(0,1,0)	...
(0,1,1)	(0,0,1)	(0,0,1)	(0,0,1)	(0,0,1)	...
(1,0,0)	(1,0,0)	(1,0,0)	(1,0,0)	(1,0,0)	...
(1,0,1)	(1,1,1)	(0,1,1)	(0,1,0)	(0,1,0)	...
(1,1,0)	(1,0,0)	(1,0,0)	(1,0,0)	(1,0,0)	...
(1,1,1)	(1,0,1)	(0,0,1)	(0,0,1)	(0,0,1)	...

We observe that any further update does not change the resulting configurations. Thus, (F, α_2) is highly stable as each trajectory reaches a fixed point.

Mathematical Tools A

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

A.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 $(f' \circ f)(x) \quad =_{\text{def}} \quad f'(f(x))$. 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., [Grä78, DP90]) 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., [HMU01]).

A.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.

A.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 [Val79a, Val79b] 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., [HMU01, HO02]). 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 [Val79b]. 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.

Bibliography

[Bar93] Yaneer Bar-Yam. *Dynamics of Complex Systems.* The Advanced Book Program, Addison Wesley, Reading, MA, 1997.

[Kau93] Stuart A. Kauffman. *The Origins of Order. Self-Organization and Selection in Evolution.* Oxford University Press, Oxford, 1993.

[MR08] Henning S. Mortveit and Christian M. Reidys. *An Introduction to Sequential Dynamical Systems.* Springer, New York, NY, 2008.

[Sch93] Heinz Georg Schuster. *Deterministic Chaos.* VCH, Weinheim, 1994.

[Wol02] Stephen Wolfram. *A New Kind of Science.* Wolfram Media, Champaign, IL, 2002.

[Grä78] George A. Grätzer. *General Lattice Theory.* Akademie-Verlag, Berlin, 1978.

[DP90] Brian A. Davey and Hilary A. Priestley. *Introduction to Lattices and Orders.* Cambridge University Press, Cambridge, 1990.

[HO02] Lane A. Hemaspaandra and Mitsunori Ogihara. *The Complexity Theory Companion.* An EATCS series. Springer-Verlag, Berlin, 2002.

[HMU01] John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation* 2nd edition. Addison Wesley, Reading, MA, 2001.

[Val79a] Leslie G. Valiant. The Complexity of Enumeration and Reliability Problems. *SIAM Journal on Computing*, **8**(3):410–421, 1979.

[Val79b] Leslie G. Valiant. The Complexity of Computing the Permanent. *Theoretical Computer Science*, **8**(2):189–201, 1979