

Constraint Satisfaction Problems

Mathematical Background: Sets, Relations, and Graphs

Bernhard Nebel, Julien Hué, and Stefan Wöfl

Albert-Ludwigs-Universität Freiburg

April 23, 25, and 2012; May 2, 2012

Constraint Satisfaction Problems

April 23, 25, and 2012; May 2, 2012 — Mathematical Background: Sets, Relations, and Graphs

1 Set-theoretical notions

2 Relations

3 Graphs

4 Computational Complexity

Constraints, sets, relations, graphs

- ▶ Formal definition of CSP uses **sets** and **constraints**
- ▶ Constraints are specific **relations** that restrict possible solutions
- ▶ CSP solving techniques use operations that manipulate sets and relations
- ▶ CSP instances can also be represented by various kinds of **graphs**
- ▶ Graph-theoretical notions can be used to describe, e.g., **structural properties** of constraint networks
- ▶ Complexity for solving CSP instances can depend on both the relations used in the constraints and properties of the constraint graphs

1 Set-theoretical notions

- Set-theoretical principles
- Sets and Boolean algebras

Sets

Sets:

Naive understanding:

a set is a “well-defined” collection of objects.

Principles/Set-theoretical axioms (ZF):

Axioms that describe which objects count as **sets** and which operations can be used to form new sets . . .

Set theory

Some set-theoretical axioms (ZF):

- ▶ **Extensionality:** Two sets are equal if and only if they contain the same elements.
- ▶ **Empty set:** There is a set, \emptyset , with no elements.
- ▶ **Pairs:** For any pair of sets x, y , $\{x, y\}$ is a set.
- ▶ **Union:** For any set x , there exists a set, $\bigcup x$, whose elements are precisely the elements of the elements of x .
- ▶ **Separation:** For any set x and any property $F(y)$, there is a subset of x , $\{y \in x : F(y)\}$, containing precisely the elements y of x for which $F(y)$ holds.
- ▶ **Power set:** For any set x there exists a set 2^x such that the elements of 2^x are precisely the subsets of x .
- ▶ **Axiom of choice:** Given a set x of pairwise disjoint nonempty sets, there is a set y that contains exactly one element from each member of x .

Set-theoretical notations

Usually, we argue naïvely by using the following notations ...

Boolean operations on sets:

$$A \cup B := \{x : x \in A \text{ or } x \in B\}$$

$$A \cap B := \{x \in A : x \in B\}$$

$$A \setminus B := \{x \in A : x \notin B\}$$

Subset relation: $A \subseteq B$, $A \subsetneq B$, etc., are defined as usual.

Power set: $2^A := \{B : B \subseteq A\}$

(Ordered) pairs:

$$(x, y) := \{\{x\}, \{x, y\}\}$$

$$(x_1, \dots, x_n) := ((x_1, \dots, x_{n-1}), x_n)$$

Product: $A \times B := \{(a, b) : a \in A \text{ and } b \in B\}$

Boolean algebra

Definition

A **Boolean algebra (with complements)** is a set A with

- ▶ two binary operations \sqcap , \sqcup ,
- ▶ a unary operation $-$, and
- ▶ two distinct elements 0 and 1

such that for all elements a , b and c of A :

$a \sqcup (b \sqcup c) = (a \sqcup b) \sqcup c$	$a \sqcap (b \sqcap c) = (a \sqcap b) \sqcap c$	Ass
$a \sqcup b = b \sqcup a$	$a \sqcap b = b \sqcap a$	Com
$a \sqcup (a \sqcap b) = a$	$a \sqcap (a \sqcup b) = a$	Abs
$a \sqcup (b \sqcap c) = (a \sqcup b) \sqcap (a \sqcup c)$	$a \sqcap (b \sqcup c) = (a \sqcap b) \sqcup (a \sqcap c)$	Dis
$a \sqcup -a = 1$	$a \sqcap -a = 0$	Compl

Set algebras

Definition

A **set algebra** on a set X is a non-empty subset \mathcal{F} of 2^X that is closed under unions, intersections, and complements.

$\langle X, \mathcal{F} \rangle$ is called a **field of sets**.

Notice: a set algebra on X contains X and \emptyset as elements.

Lemma

- (a) *The power set of any set forms a set algebra.*
- (b) *Each set algebra defines a Boolean algebra.*
- (c) *A finite Boolean algebra can always be represented as a power set, . . .*
- (d) *more generally, each Boolean algebra is isomorphic to a field of sets (Stone's representation theorem).*

Boolean algebras vs set algebras I

Proof of the lemma.

- (a) By applying complement, union, or intersection on subsets of a given set X , we again obtain subsets of X .
- (b) A set algebra \mathcal{F} on X contains \emptyset and X . $\bar{A} := X \setminus A$ is a unary operation on \mathcal{F} ; \cap and \cup are binary operations. Hence, $\langle \mathcal{F}, \cap, \cup, \bar{\cdot}, \emptyset, X \rangle$ is a structure that obviously satisfies all properties of a Boolean algebra.
- (c) One has to show: given a finite Boolean algebra $B = \langle A, \cap, \cup, \bar{\cdot}, 0, 1 \rangle$ there exists a set X such that ...

(next slide ...)

Boolean algebras vs set algebras II

Proof of the lemma (cont'd):

... B and 2^X are **isomorphic** (as Boolean algebras).

1. Define a partial order on B :

$$a \leq b : \iff b \sqcap a = a \quad (\iff b \sqcup a = b \iff a \sqcap -b = 0)$$

$$a < b : \iff a \leq b \wedge a \neq b.$$

The set of **atoms** (i.e., non-zero minimal element of B) is def. by:

$$\text{At}_B := \{a \in A : 0 \leq a \wedge \text{there is no } b \in A \text{ s.t. } 0 < b < a\}.$$

2. Define a map $f : A \rightarrow 2^{\text{At}_B}$, $x \mapsto \{a \in \text{At}_B : a \leq x\}$.

Obviously $f(a) = \{a\}$ for each $a \in \text{At}_B$.

3. f is an **homomorphism** of Boolean algebras, i.e., it preserves Boolean operations: $f(0) = \emptyset$, $f(1) = X$, $f(-x) = \overline{f(x)}$, $f(x \sqcap y) = f(x) \cap f(y)$, and $f(x \sqcup y) = f(x) \cup f(y)$.

4. f is a **bijection**, i.e., it is injective (“one-to-one”) and surjective (“onto”). \square

2 Relations

- Relations
- Binary Relations
- Relations over Variables

Relations

Definition

A **relation over** sets X_1, \dots, X_n is a subset

$$R \subseteq X_1 \times \dots \times X_n =: \prod_{1 \leq i \leq n} X_i.$$

The number n is referred to as **arity** of R .

An **n -ary relation on** a set X is a subset

$$R \subseteq X^n := X \times \dots \times X \quad (n \text{ times}).$$

Since relations are sets, set-theoretical operations (union, intersection, complement) can be applied to relations as well.

Binary relations

For binary relations on a set X we have some special operations:

Definition

Let R, S be binary (2-ary) relations on X .

The **converse** of relation R is defined by:

$$R^{-1} := \{(x, y) \in X^2 : (y, x) \in R\}.$$

The **composition** of relations R and S is defined by:

$$R \circ S := \{(x, z) \in X^2 : \exists y \in X \text{ s.t. } (x, y) \in R \text{ and } (y, z) \in S\}.$$

The **identity relation** is:

$$\Delta_X := \{(x, y) \in X^2 : x = y\}.$$

Operating on binary relations

Lemma

Let X be a non-empty set. Let $\mathcal{R}(X)$ be the set of all binary relations on X . Then:

- (a) $\mathcal{R}(X)$ is a set algebra on $X \times X$.
 (b) For all relations $R, S, T \in \mathcal{R}(X)$:

$$R \circ (S \circ T) = (R \circ S) \circ T$$

$$R \circ (S \cup T) = (R \circ S) \cup (R \circ T)$$

$$\Delta_X \circ R = R \circ \Delta_X = R$$

$$(R^{-1})^{-1} = R \text{ and } (-R)^{-1} = -(R^{-1})$$

$$(R \cup S)^{-1} = R^{-1} \cup S^{-1}$$

$$(R \circ S)^{-1} = S^{-1} \circ R^{-1}$$

$$(R \circ S) \cap T^{-1} = \emptyset \text{ if and only if } (S \circ T) \cap R^{-1} = \emptyset$$

Constraints, relations, and variables

Constraints can be expressed by relations that restrict value assignments to variables.

Consider variables x_1, x_2, x_3 and relations B, C defined by:

$$B = \{(x, y, z) \in [0..3]^3 : x < y < z\}$$

$$C = \{(x, y, z) \in [0..3]^3 : x > y > z\}.$$

- ▶ “ (x_1, x_2, x_3) satisfies B ” and “ (x_3, x_2, x_1) satisfies B ” express **different** constraints, while ...
- ▶ “ (x_3, x_2, x_1) satisfies B ” and “ (x_1, x_2, x_3) satisfies C ” essentially express the **same** constraint.

x_1	x_2	x_3		x_3	x_2	x_1		x_1	x_2	x_3
0	1	2		0	1	2		2	1	0
0	1	3	\neq	0	1	3	\equiv	3	1	0
0	2	3		0	2	3		3	2	0
1	2	3		1	2	3		3	2	1

Relations over variables

Let V be a set of variables. For $v \in V$, let $\text{dom}(v)$ be a non-empty set (of values), called the **domain of v** .

Definition

A **relation** over (pairwise distinct) variables $v_1, \dots, v_n \in V$ is a pair

$$R_{v_1, \dots, v_n} := ((v_1, \dots, v_n), R)$$

where R is a relation over $\text{dom}(v_1), \dots, \text{dom}(v_n)$.

The sequence (v_1, \dots, v_n) is referred to as the **scheme** (or: **range**), the set $\{v_1, \dots, v_n\}$ as the **scope**, and R as the **graph** of R_{v_1, \dots, v_n} .

We will not always distinguish between a relation over variables and its graph (and between scope and scheme), e. g., we write

$$R_{v_1, \dots, v_n} \subseteq \text{dom}(v_1) \times \dots \times \text{dom}(v_n).$$

Selections, ...

Let $R_{\bar{v}} = (\bar{v}, R)$ be a relation over variables $\bar{v} = (v_1, \dots, v_n)$.

Definition

For any fixed values $a_1 \in \text{dom}(v_{i_1}), \dots, a_k \in \text{dom}(v_{i_k})$, define

$$\sigma_{v_{i_1}=a_1, \dots, v_{i_k}=a_k}(\bar{v}, R) := (\bar{v}, R')$$

with

$$R' := \{(x_1, \dots, x_n) \in R : x_{i_j} = a_j, \text{ for each } 1 \leq j \leq k\}.$$

The (unary) operation $\sigma_{v_{i_1}=a_1, \dots, v_{i_k}=a_k}$ is called **selection** or **restriction**.

... Projections, ...

Let (i_1, \dots, i_k) be a k -tuple of pairwise distinct elements of $\{1, \dots, n\}$ ($k \leq n$).

Definition

Given a relation (\bar{v}, R) over \bar{v} ,

$$\pi_{v_{i_1}, \dots, v_{i_k}}(\bar{v}, R) := ((v_{i_1}, \dots, v_{i_k}), R')$$

with

$$R' := \left\{ \bar{y} \in \prod_{1 \leq j \leq k} \text{dom}(v_{i_j}) : \bar{y} = (x_{i_1}, \dots, x_{i_k}), \right. \\ \left. \text{for some } (x_1, \dots, x_n) \in R \right\}$$

is a relation over $(v_{i_1}, \dots, v_{i_k})$, called the **projection** of (\bar{v}, R) on $(v_{i_1}, \dots, v_{i_k})$.

Note: Each permutation of the scheme \bar{v} defines a projection.

For binary relations $R = R_{x,y}$, $R^{-1} = \pi_{y,x}(R_{x,y})$.

... Joins

Definition

Consider pairwise distinct variables v_1, \dots, v_n .

Let (\bar{v}, R) and (\bar{v}', S) be relations over variables $\bar{v} = (v_{i_1}, \dots, v_{i_k})$ and $\bar{v}' = (v_{j_1}, \dots, v_{j_l})$, resp., such that

$\{v_{i_1}, \dots, v_{i_k}\} \cup \{v_{j_1}, \dots, v_{j_l}\} = \{v_1, \dots, v_n\}$. Then

$$(\bar{v}, R) \bowtie (\bar{v}', S) := ((v_1, \dots, v_n), T)$$

with

$$T = \left\{ \bar{x} \in \prod_{1 \leq i \leq n} \text{dom}(v_i) : (x_{i_1}, \dots, x_{i_k}) \in R \text{ and } (x_{j_1}, \dots, x_{j_l}) \in S \right\}$$

is a relation over (v_1, \dots, v_n) , the **join** of (\bar{v}, R) and (\bar{v}', S) .

For binary relations $R = R_{x,y}$ and $S = S_{y,z}$ on the same set,

$$R \circ S = \pi_{x,z}(R_{x,y} \bowtie S_{y,z}).$$

Examples

Consider relations $R := R_{x_1, x_2, x_3}$ and $S := S_{x_2, x_3, x_4}$ defined by:

x_1	x_2	x_3
b	b	c
c	b	c
c	n	n

x_2	x_3	x_4
a	a	1
b	c	2
b	c	3

Then $\sigma_{x_3=c}(R)$, $\pi_{x_2, x_3}(R)$, $\pi_{x_2, x_1}(R)$, and $R \bowtie S$ are:

x_1	x_2	x_3
b	b	c
c	b	c

x_2	x_3
b	c
b	c
n	n

x_2	x_1
b	b
b	c
n	c

x_1	x_2	x_3	x_4
b	b	c	2
b	b	c	3
c	b	c	2
c	b	c	3

3 Graphs

- Undirected Graphs
- Directed Graphs
- Labeled Graphs
- Hypergraphs

Undirected graph

Definition

An (undirected, simple) graph is an ordered pair

$$G := \langle V, E \rangle$$

where:

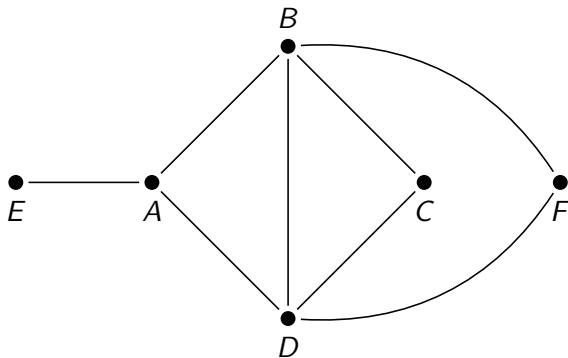
- ▶ V is a non-empty set (of vertices, nodes);
- ▶ E is a set of two-element subsets $X \subseteq V$ (elements of E are called edges).

Usually, we assume that the graph (i.e., $|V|$) is finite.

In undirected, simple graphs edges are often written as $[u, v]$.

Sometimes, one allows E to also contain singleton subsets of V (loops), written as $[v, v]$. But simple graphs are always loopless.

A simple undirected graph



Undirected multi-graph

Often we allow for multiple edges between the same set of end vertices.

Definition

An **(undirected, multi-) graph** is an ordered triple

$$G := \langle V, E, \gamma \rangle$$

where:

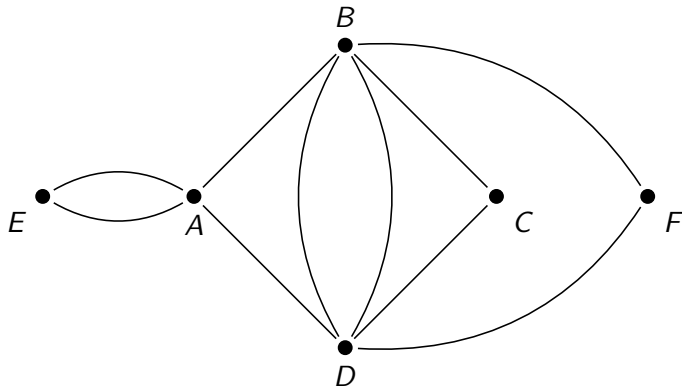
- ▶ V is non-empty set (of **vertices, nodes**);
- ▶ $\gamma : E \rightarrow \{X \in 2^V : 1 \leq |X| \leq 2\}$.

The elements of E are called **edges**.

We always assume: $V \cap E = \emptyset$.

The **order** of a graph is the number of vertices $|V|$. Often, $|E|$ is referred to as the **size** of G , but often we specify both $n := |V|$ and $m := |E|$.

An undirected multi-graph



Graphs: Some definitions

Definition

Let $G = \langle V, E, \gamma \rangle$ be an undirected graph.

- (a) If $\gamma(e) = \{u, v\}$ for some $e \in E$, then u and v are called **adjacent** (or: **connected** by e).
- (b) A **path** (or: **walk**) in G is a sequence

$$(v_0, e_1, v_1, \dots, e_k, v_k)$$

such that $e_1, \dots, e_k \in E$ and $\gamma(e_i) = \{v_{i-1}, v_i\}$ (for each $1 \leq i \leq k$).
 k is referred to as **length**, v_0 as **start vertex**, and v_k as **end vertex** of the path.

- (c) A **cycle** is a path (v_0, \dots, e_k, v_k) with $v_0 = v_k$ and $k \geq 1$.
- (d) A path (v_0, \dots, e_k, v_k) is **simple** if $e_i \neq e_j$ for all $i \neq j$.
- (e) A path (v_0, \dots, e_k, v_k) is **elementary** if $v_i \neq v_j$ for $0 \leq i \neq j \leq k$ (but $v_0 = v_k$ is allowed).

Paths: An example

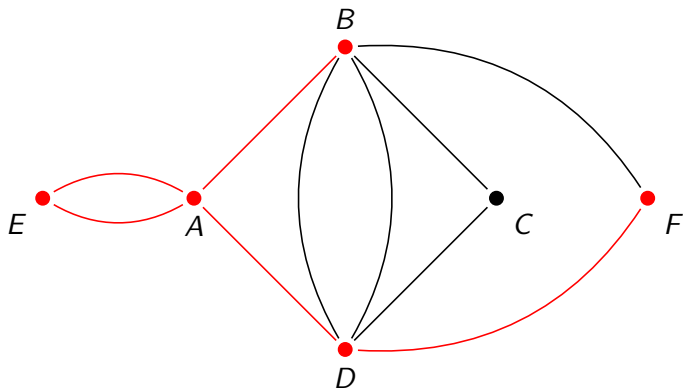


Figure: A simple path visiting the nodes B, A, E, D, F

Graph-theoretical notions

Let $G = \langle V, E, \gamma \rangle$ be an undirected graph.

Definition

- (a) G is **connected** if for each pair of vertices u and v , there exists a path from u to v .
- (b) G is **complete** if any pair of vertices is connected by an edge.
- (c) G is a **forest** if G is cycle-free.
- (d) G is a **tree** if G is cycle-free and connected.

Examples

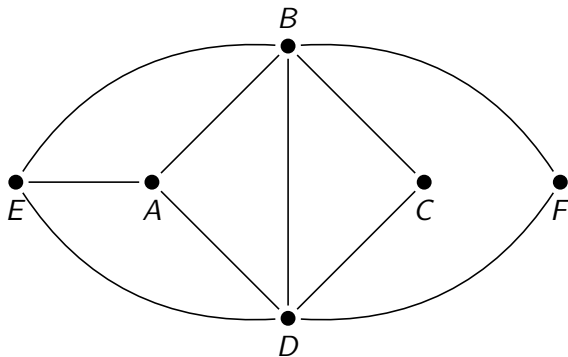


Figure: Connected, but not complete

Examples

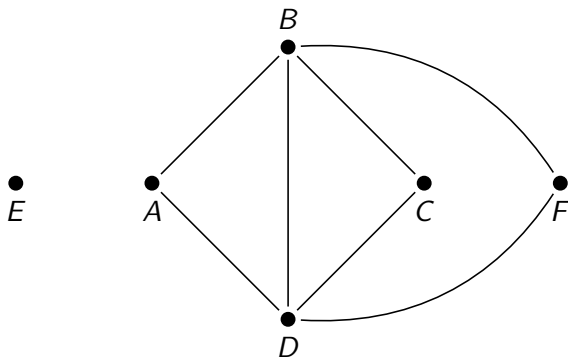


Figure: A not connected graph

Examples

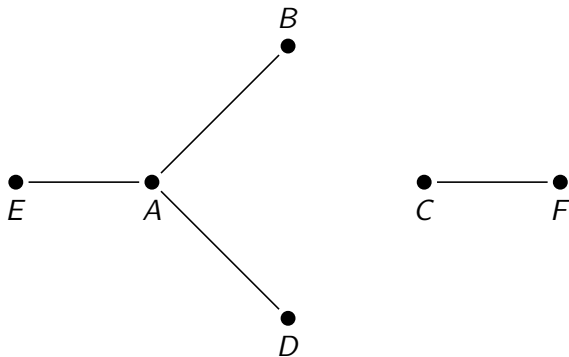


Figure: A forest

Examples

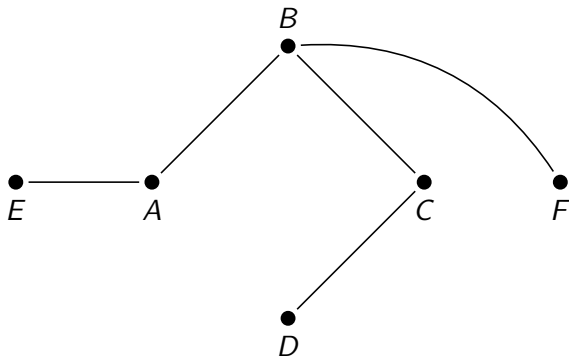


Figure: A tree

Graph-theoretical notions

Let $G = \langle V, E, \gamma \rangle$ be an undirected graph.

Definition

Let V' be a non-empty subset of V . Then $G[V'] = \langle V', E', \gamma' \rangle$ with:

$$E' := \{e \in E : \gamma(e) \subseteq V'\} \text{ and } \gamma' := \gamma|_{E'}$$

is called the **subgraph** induced by V' .

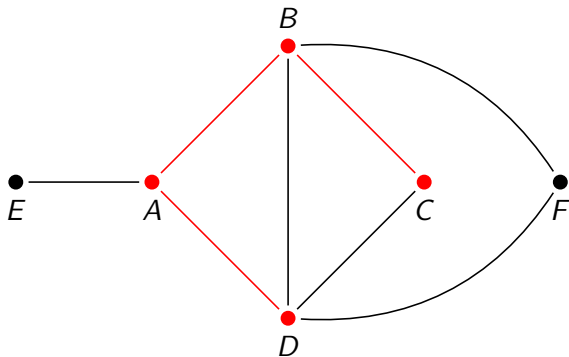
Definition

Let E' be a subset of E . Then $G[E'] = \langle V', E', \gamma|_{E'} \rangle$ is called the **partial graph** induced by E' .

Definition

A **clique** in a graph G is a complete subgraph of G .

Examples



Directed Graph

Definition

A **directed (multi-) graph** (or: **digraph**) is an ordered tuple

$$G := \langle V, A, \alpha, \omega \rangle$$

where:

- ▶ V is a non-empty set (of **vertices** or **nodes**),
- ▶ A is a set (elements of A are called **arcs**, **edges**, or **arrows**),
- ▶ $\alpha, \omega : A \rightarrow V$ are functions.

$\alpha(a)$ is called the **start vertex** of a , $\omega(a)$ the **end vertex** of a .

If G has no parallel arcs ($a, a' \in A$ with $\alpha(a) = \alpha(a')$ and $\omega(a) = \omega(a')$), we can write A as a set of tuples:

$$\{(\alpha(a), \omega(a)) \in V^2 : a \in A\}.$$

In that case we use the notation $\langle V, A \rangle$ instead of $\langle V, A, \alpha, \omega \rangle$.

Digraphs: Some notions

Most notions introduced for undirected graphs can easily be adapted for directed graphs. For example:

Definition

A **path** in G is a sequence $(v_0, a_1, v_1, \dots, a_k, v_k)$ such that $a_1, \dots, a_k \in A$ and for each $1 \leq i \leq k$, $\alpha(a_i) = v_{i-1}$ and $\omega(a_i) = v_i$.

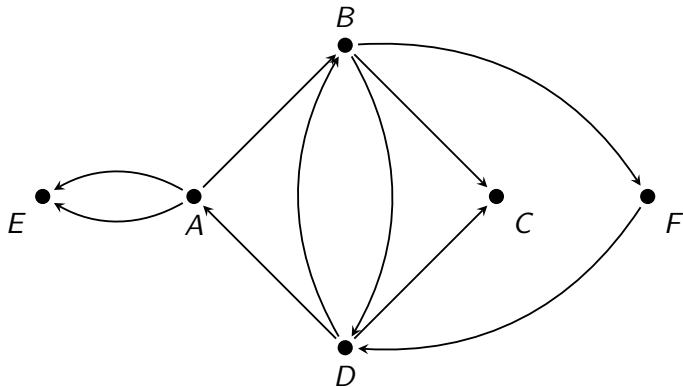
$g^+(v)$: the **outdegree** of v , the number of arcs that start from v

$g^-(v)$: the **indegree** of v , the number of arcs that end in v

parents of v : nodes with an arc to v

childs of v : nodes with an arc from v

A directed multi-graph



A directed multi-graph

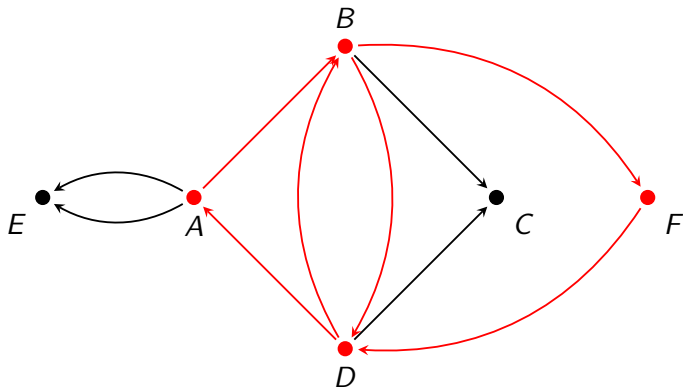


Figure: A directed graph with a (strongly) connected subgraph

Labeled graphs

Often graphs $G = \langle V, E/A, \dots \rangle$ are equipped with labeling functions.

Let L be a not-empty set of labels.

Vertex labeling: a function $l : V \rightarrow L$ that assigns to each v a vertex label $l(v) \in L$.

Edge labeling: a function $l : E \rightarrow L$ that assigns to each $e \in E$ a label $l(e) \in L$.

Example: In route planning, one can represent street networks as digraphs with an arc labeling (expressing travelling distance between places/nodes).

The label set may be equipped with further structures. In the route planning example, the labeling function is understood as a distance function (metric space).

Hypergraph

Graphs can be used to represent binary relations between nodes.
For relations of higher arity we need:

Definition

A **hypergraph** is a pair $H := \langle V, E \rangle$, where

- ▶ V is a set (of **nodes**, **vertices**),
- ▶ E is a set of non-empty subsets of V (called **hyperedges**), i.e.,
 $E \subseteq 2^V \setminus \{\emptyset\}$.

Notice: Hyperedges may contain arbitrarily many nodes.

k -uniform hypergraph: each hyperedge contains exactly k vertices.

Hypergraphs: An example

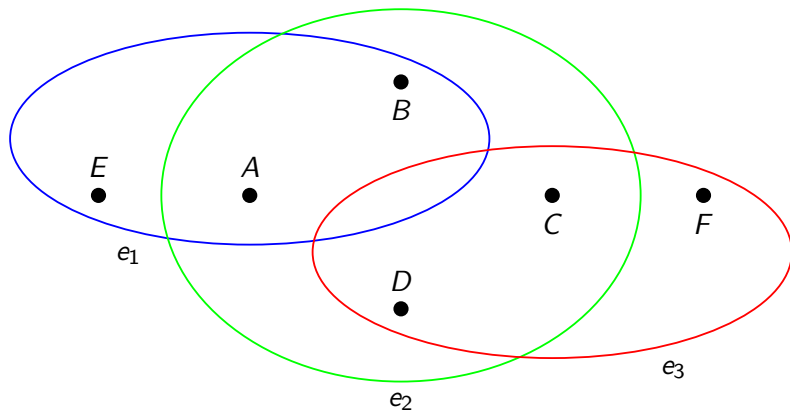


Figure: A hypergraph

4 Computational Complexity

- \mathcal{O} , Ω , etc.
- Computational Problems
- NP

Model of computation

- ▶ In the lecture we do not use a specific model of computation: any Turing-complete abstract machine (Turing machine, (unit cost) RAM, ...) suffices
- ▶ When analyzing algorithms, we use a **uniform cost model**: constant costs are assumed for every machine operation (regardless of the size of its input)

Landau symbols

Let M be the set of all functions $f : \mathbb{N} \rightarrow \mathbb{R}$, $g \in M$.

$$\mathcal{O}(g) = \{f \in M : \exists c \in \mathbb{R} \exists n_0 \in \mathbb{N} \forall n > n_0 : f(n) \leq c \cdot g(n)\}$$

$$\Omega(g) = \{f \in M : \exists c \in \mathbb{R} \exists n_0 \in \mathbb{N} \forall n > n_0 : f(n) \geq c \cdot g(n)\}$$

$$\Theta(g) = \mathcal{O}(g) \cap \Omega(g)$$

Data structures

- ▶ Runtime depends on used data structures
- ▶ For example: basic operations on a graph depend on how the graph is represented (e.g., as an **adjacency matrix** or an **adjacency list**).

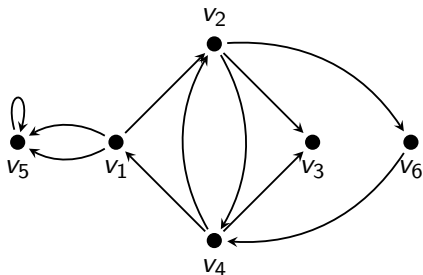
Let $G = \langle V, A, \alpha, \omega \rangle$ be a digraph.

Adjacency matrix: $n \times n$ matrix $(a_{ij})_{1 \leq i, j \leq n}$ such that a_{ij} is the number of arcs from vertex v_i to vertex v_j .

Adjacency list: an array of lists, namely, for each vertex v , the list of v 's children (in undirected graphs: neighbors = adjacent vertices)

Adjacency matrix

Graph:

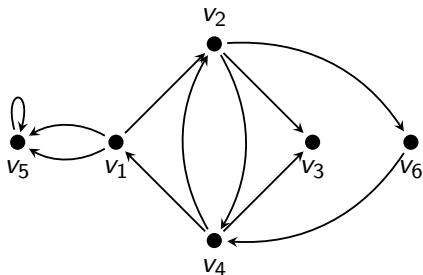


Adjacency matrix:

$$\begin{pmatrix} 0 & 1 & 0 & 0 & 2 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}$$

Adjacency list

Graph:



Adjacency list:

1 \rightarrow (2, 5, 5)

2 \rightarrow (3, 4, 6)

3 \rightarrow ()

4 \rightarrow (1, 2)

5 \rightarrow (5)

6 \rightarrow (4)

Comparing basic operations

Consider the following operations on a digraph (without parallel arcs):

- ▶ **Arc:** Check whether there is an arc from v to w ($(v, w) \in E?$);
- ▶ **Deg⁺:** Determine the outdegree of v ($g^+(v) = ?$);
- ▶ **Root:** Check whether there exists a v with $g^-(v) = 0$.

Data structure	Memory	Arc	Deg ⁺	Root
Adjacency matrix	$\Theta(n^2)$	$\mathcal{O}(1)$	$\mathcal{O}(n)$	$\mathcal{O}(n^2)$
Adjacency list	$\Theta(n + m)$	$\mathcal{O}(g^+(v))$	$\mathcal{O}(g^+(v))$	$\mathcal{O}(n + m)$

n : number of vertices; m : number of arcs/edges

Computational problems

In the lecture we will study three types of computational problems:

- ▶ Decision problems
Expected output: YES/No
- ▶ Search problems
Expected output: a solution
- ▶ Optimization problems
Expected output: an optimal solution

Decision problem

Let P be a set of problem instances and F be a unary property defined on P .

Then the decision problem “ x satisfies F ?” is defined as follows:

- ▶ **Given:** A problem instance $x \in P$
- ▶ **Question:** Does x satisfy condition F ?

Example

- ▶ **Given:** A digraph $G = \langle V, E \rangle$, vertices $v_1, v_2 \in V$.
- ▶ **Question:** Does there exist a path from v_1 to v_2 in G ?

Search problem

Let P be a set of problem instances, S be the set of solutions, and R be a binary relation $R \subseteq P \times S$.

Then the search problem “Find a solution of x ?” is defined as follows:

- ▶ **Given:** A problem instance $x \in P$
- ▶ **Asked:** A solution $s \in S$ with $(x, s) \in R$

Example

- ▶ **Given:** A digraph $G = \langle V, E \rangle$, vertices $v_1, v_2 \in V$.
- ▶ **Asked:** **Find** a path from v_1 to v_2 in G (if there exists one; otherwise “failure”)!

Optimization problem

Let P be a set of problem instances, S be the set of solutions, R be a binary relation $R \subseteq P \times S$, and $f : S \rightarrow \mathbb{R}$ be an **objective function**.

The optimization problem “Find an optimal solution of x ?” is defined as follows:

- ▶ **Given:** A problem instance $x \in P$
- ▶ **Asked:** A solution $s \in S$ with $(x, s) \in R$ that maximizes/minimizes f , i.e., $f(s)$ is maximal/minimal among all s with $(x, s) \in R$.

Example

- ▶ **Given:** A weighted digraph $G = \langle V, E \rangle$, vertices $v_1, v_2 \in V$.
- ▶ **Asked:** Find a **shortest** path from v_1 to v_2 in G (if there exists one; otherwise “failure”)!

P, NP

P: class of decision problems that can be solved by a deterministic Turing machine in polynomial time

NP: class of decision problems that can be solved by a non-deterministic Turing machine in polynomial time

Alternative characterization of NP:

NP: class of decision problems P such that there exists a polynomial **verifier** for P .

A **verifier** for a decision problem P is a procedure that, given a problem instance x and a candidate solution s (called *certificate*), verifies that s is a solution of x .

A verifier is **polynomial** if it verifies $(x, s) \in R$ in polynomial time (it need not run in polynomial time on input $(x, s) \notin R$)

NP-completeness

Consider decision problems P and P' encoded as formal languages L, L' over alphabets Σ, Σ' .

Polynomial reduction: L' is **polynomially reducible** to L , $L' \leq_p L$, if there exists a total and polynomial time-computable function $f : \Sigma' \rightarrow \Sigma$ such that $x \in L' \iff f(x) \in L$.

Definition

- ▶ A decision problem L is **NP-hard** if for each decision problem L' in NP, it holds $L' \leq_p L$.
- ▶ A decision problem L is **NP-complete** if it is both in NP and NP-hard.

SAT, 3SAT

Theorem (Cook)

The Boolean satisfiability problem, i.e., the problem of deciding whether a propositional logic formula φ is satisfiable, is NP-complete.

3CNF-SAT formula: a propositional logic formula φ that is in conjunctive normal form such that each clause contains at most 3 literals.

Theorem (3CNF-SAT)

The problem of deciding whether a 3CNF-SAT formula is satisfiable is NP-complete.

3-COLORABILITY

The problem **3-COLORABILITY** is defined as follows:

Given an undirected, simple graph $G = \langle V, E \rangle$, is there a vertex coloring $c : V \rightarrow \{1, 2, 3\}$ such that for each pair of adjacent vertices v, v' in G , $c(v) \neq c(v')$.

Theorem

3-COLORABILITY is NP-complete.

Proof.

Obviously, **3-COLORABILITY** is in NP: we only need to guess the coloring c . Then we check whether this coloring assigns different colors to adjacent vertices. This can be done in polynomial time.

We now show that **3-COLORABILITY** is NP-hard by a polynomial reduction from **3CNF-SAT**. Since **3CNF-SAT** is NP-complete, each problem in NP can be reduced to **3CNF-SAT** and via $\text{3CNF-SAT} \leq_p \text{3-COLORABILITY}$, each problem in NP can also be reduced to **3-COLORABILITY**.

...

(next slide ...)

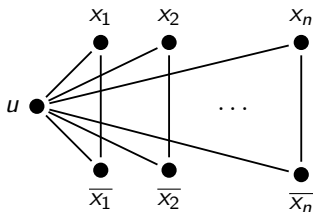
3-COLORABILITY

We construct a function that assigns to each 3CNF-SAT formula

$\varphi = C_1 \wedge \dots \wedge C_m$ a graph G_φ such that

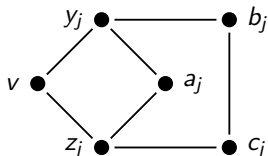
φ is satisfiable $\iff G_\varphi$ has a coloring with colors {red, blue, green}.

We assume (w.l.o.g.) that each clause C_j consists of exactly three literals, i.e., $C_j = (l_{j1} \vee l_{j2} \vee l_{j3})$. Let x_1, \dots, x_n be the set of propositional variables that occur in φ . G_φ will contain the following subgraph G_T (with $2n + 1$ vertices):



3-COLORABILITY

For each clause C_j ($1 \leq j \leq m$) we add a subgraph G_j (clause gadget) with new vertices a_j, b_j, c_j, y_j, z_j and a vertex v which is the same in each of the clause gadgets:

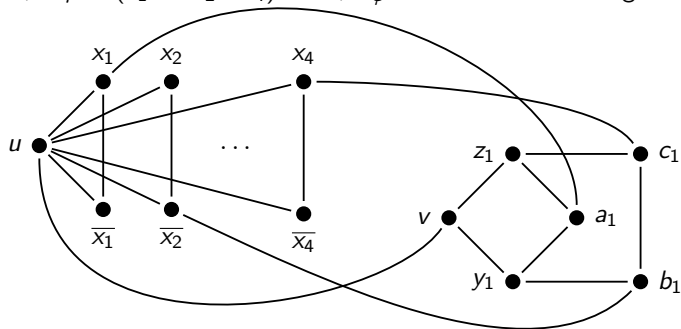


Vertices in G_j are connected by an edge to vertices in G_T as follows:

- ▶ an edge $\{u, v\}$
- ▶ edges $\{a_j, l_{j1}\}, \{b_j, l_{j2}\}, \{c_j, l_{j3}\}$ ($1 \leq j \leq m$)

3-COLORABILITY

For example, if $\varphi = (x_1 \vee \neg x_2 \vee x_4) \wedge \dots$, G_φ contains the following subgraph G_1 :



For example: if $V(x_1) = 1, V(x_2) = 1, V(x_4) = 0, \dots$,

3-COLORABILITY

Proof (summary).

Thus we have constructed a function f that assigns to each 3CNF-SAT formula $\varphi = C_1 \wedge \cdots \wedge C_m$ a graph G_φ such that

$$\varphi \text{ is satisfiable} \iff G_\varphi \text{ has a coloring with colors } \{\text{red, blue, green}\}.$$

Since the constructed graph G_φ has $2n + 5m + 2$ vertices, f can be computed in polynomial time. □

Notice:

- ▶ Actually, what we have proven is: $3\text{CNF-SAT} \leq_p k\text{-COLORABILITY}$, for $k \geq 3$.
- ▶ The corresponding search problem “Given a graph, find a 3-coloring . . .” is in the complexity class **Function NP (FNP)**.

Summary

- ▶ Short reminder on set-theoretical notions and operations
- ▶ Even more operations can be defined for relations
- ▶ Distinguish relations (as sets) and relations over variables
- ▶ Very basic reminder of graph-theoretical notions
- ▶ ... and complexity theory
- ▶ Example: k -colorability is an NP-complete decision problem
- ▶ ... for $k \geq 3$; for $k = 2$ it is **tractable**

Literature



Rina Dechter.
Constraint Processing,
Chapter 1 and 2, Morgan Kaufmann, 2003



Sven Oliver Krumke and Hartmut Noltemeier.
Graphentheoretische Konzepte und Algorithmen,
Vieweg+Teubner, 2009



Uwe Schöning.
Theoretische Informatik – kurzgefasst,
Spektrum, 2001



Wikipedia contributors,
Graph theory, Graph (mathematics), Boolean Algebra, Relational Algebra,
(2007, April),
In Wikipedia, The Free Encyclopedia. Wikipedia.