Constraint Satisfaction Problems

Mathematical Background: Sets, Relations, and Graphs

Albert-Ludwigs-Universität Freiburg

UNI FREIBURG

Stefan Wölfl, **Christian Becker-Asano**, and **Bernhard Nebel**

October 23 and 27, 2014

Constraints, sets, relations, graphs

- Formal definition of CSP uses sets and constraints
- Constraints are relational statements that restrict possible solutions
- CSP solving techniques use operations thatmanipulate 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

Set-Theoretical Basics and Relations

Set-theoretical notations

Usually, we use sets in a naïve way. The following notations are all standard ...

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:

$$(x,y) := \{\{x\},\{x,y\}\}$$
$$(x_1,\ldots,x_n) := ((x_1,\ldots,x_{n-1}),x_n)$$

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

Relations

Definition

A relation over sets $X_1, \ldots, X_n$ is a subset

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

The number n is referred to as arity of R.
An n-ary relation on a set X is a subset

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

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:

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

The composition of relations R and S is defined by:

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

The identity relation is:

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

Operating on binary relations

Lemma

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

(a) *$\mathscr{R}(X)$ forms a Boolean algebra on $X \times X$.*

(b) *For all relations $R, S, T \in \mathscr{R}(X)$:*

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

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

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

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

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

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

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

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:

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

- "(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
0	1	2
0	1	3
0	2	3

$\not\equiv$

x_3	x_2	x_1
0	1	2
0	1	3
0	2	3

$\equiv$

x_1	x_2	x_3
2	1	0
3	1	0
3	2	0

Relations over variables

Let V be a set of variables. For $v_i \in V$, let $\mathrm{dom}(v_i)$ be a set (of values), called the domain of v_i.

Definition

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

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

where R is a relation over $\mathrm{dom}(v_1), \ldots, \mathrm{dom}(v_n)$.
The sequence $(v_1, \ldots, v_n)$ is referred to as the scheme (or: range), the set $\{v_1, \ldots, v_n\}$ as the scope, and R as the graph of $R_{v_1,\ldots,v_n}$.

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

$$R_{v_1,\ldots,v_n} \subseteq \mathrm{dom}(v_1) \times \cdots \times \mathrm{dom}(v_n).$$

Selections, ...

Let $R_v = (v, R)$ be a relation over variables $v = (v_1, \ldots, v_n)$.

Definition

For any fixed value $a_i \in \mathrm{dom}(v_i)$, define

$$\sigma_{v_i=a_i}(v, R) := (v, R')$$

with

$$R' := \{(x_1, \ldots, x_n) \in R : x_i = a_i\}.$$

The (unary) operation $\sigma_{v_i=a_i}$ is called selection or restriction.

A multiple selection operation $\sigma_{v_{i_1}=a_1,\ldots,v_{i_k}=a_k}$ can be defined in a similar way.

… 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 (v, R) over $v = (v_1, \dots, v_n)$,

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

with

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

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

Note: Each permutation of the scheme v defines a projection. For binary relations R, ${R_{x,y}}^{-1} = \pi_{y,x}(R_{x,y})$.

… Joins

Definition

Consider a tuple of pairwise distinct variables $v = (v_1, \dots, v_n)$. Let (v', R) and (v'', S) be relations over variables $v' = (v_{i_1}, \dots, v_{i_k})$ and $v'' = (v_{j_1}, \dots, v_{j_l})$, resp., s. t. $\{v_{i_1}, \dots, v_{i_k}\} \cup \{v_{j_1}, \dots, v_{j_l}\} = \{v_1, \dots, v_n\}$. Then

$$(v', R) \bowtie (v'', S) := (v, T)$$

with

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

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

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

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

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

Graphs

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.

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.

Undirected graph

Definition

An (undirected, simple) graph is an ordered pair

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

where:

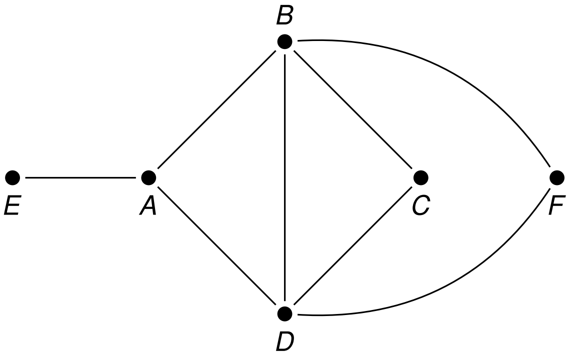
- V is a non-empty set (of vertices, nodes);
- E is a set of two-element subsets $X \subseteq V$ (elements of E are called edges).

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 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);
- $\gamma : E \rightarrow \left\{ X \in 2^V : 1 \leq |X| \leq 2 \right\}$.

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 $m := |E|$.

Undirected multi-graph

Often we allow for multiple edges between the same vertices.

Definition

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

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

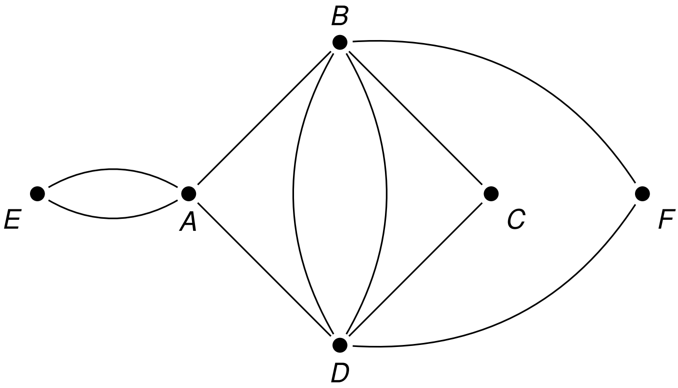
where:

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

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: Walks, cycles, etc.

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

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

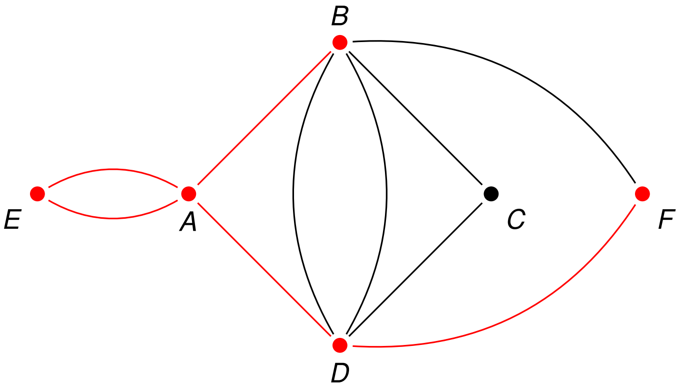
such that $e_1, \dots, e_k \in E$ and $\gamma(e_i) = \{v_{i-1}, v_i\}$ (for each $1 \leq i \leq k$). k is referred to as length, v_0 as start vertex, and v_k as end vertex of the path.

(c) A cycle is a path $(v_0, \dots, e_k, v_k)$ with $v_0 = v_k$ and $k \geq 1$.

(d) A walk $(v_0, \dots, e_k, v_k)$ is simple if $e_i \neq e_j$ for all $i \neq j$.

(e) A walk $(v_0, \dots, e_k, v_k)$ is elementary if $v_1 \neq v_j$ for $0 \leq i \neq j \leq k$ (but $v_0 = v_k$ is allowed).

Paths: An example



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

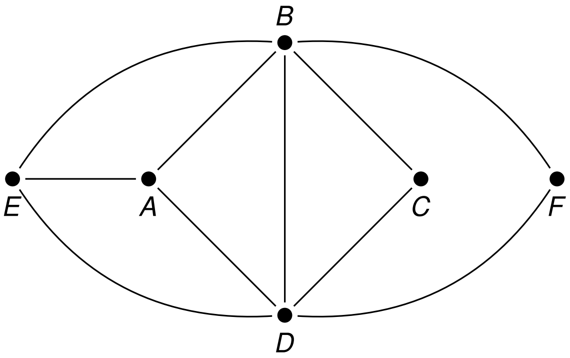
(a) G is connected if for each pair of vertices u and v, there exists a path from u to v.

(b) G is complete if any pair of vertices is connected by an edge.

(c) G is a forest if G is cycle-free.

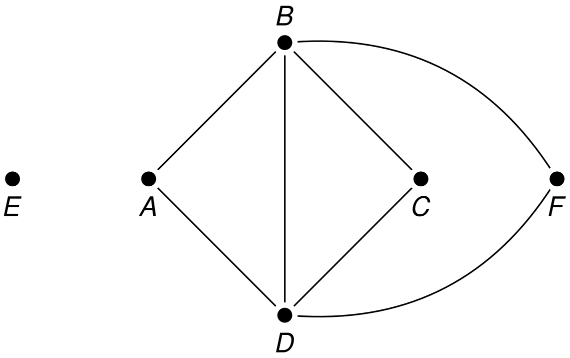
(d) G is a tree if G is cycle-free and connected.

Examples



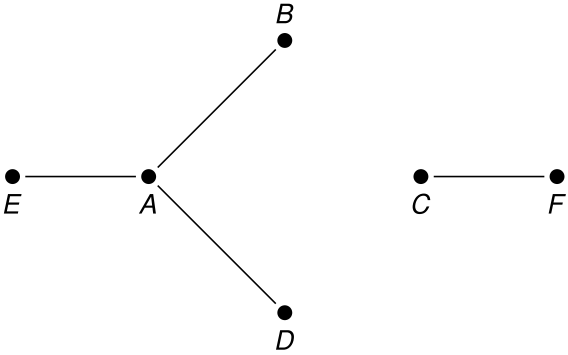
Connected, but not complete

Examples



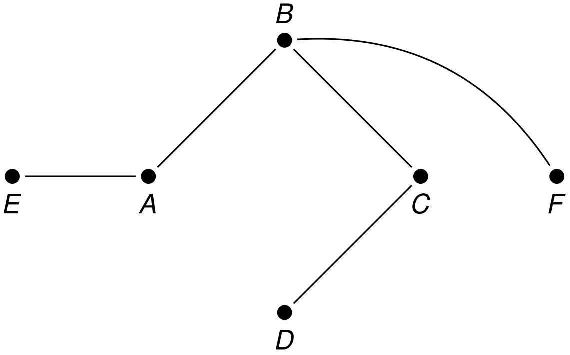
A not connected graph

Examples



A forest

Examples



A tree

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:

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

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.

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.

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:

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

is called the subgraph induced by V'.

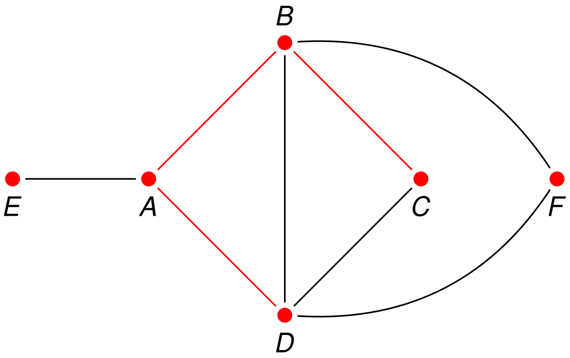
Definition

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

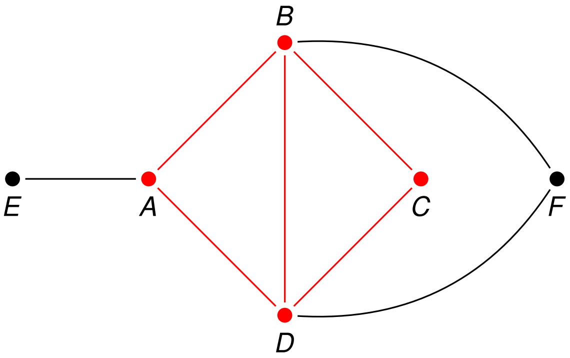
A clique in a graph G is a complete subgraph of G.

Examples



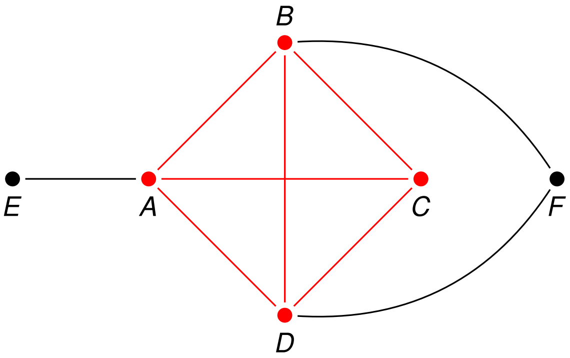
A partial graph

Examples



A subgraph

Examples



A clique

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),
- $\alpha, \omega : A \to V$ are functions.

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

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

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

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),
- $\alpha, \omega : A \to V$ are functions.

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

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

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

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 $(v_0, a_1, v_1, \ldots, a_k, v_k)$ such that $a_1, \ldots, a_k \in A$ 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

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

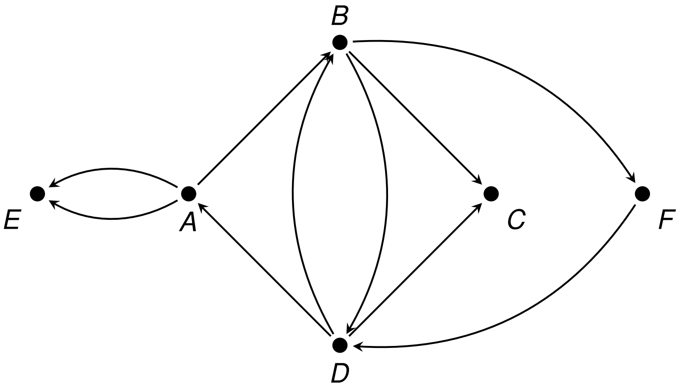
Definition

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

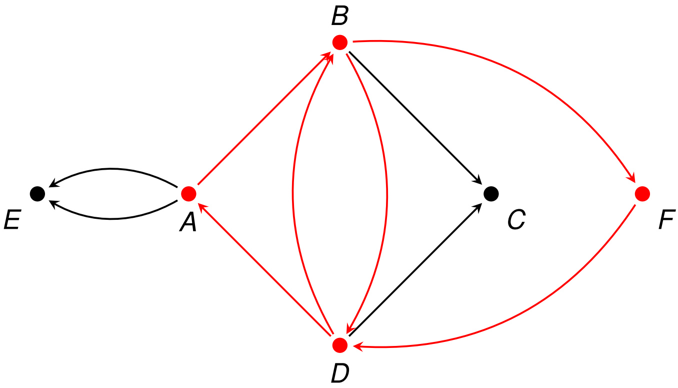
$g^+(v)$: the outdegree of v, the number of arcs that start from v
$g^-(v)$: the indegree of v, the number of arcs that end in v

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



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

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

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).

Labeled graphs

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

Let L be a not-empty set of labels.
Vertex labeling: a function $l : V \to L$ that assigns to each v a vertex label $l(v) \in L$.

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

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).

Labeled graphs

Often graphs $G = \langle V, E/A, \ldots \rangle$ 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 $l(v) \in L$.

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).

Labeled graphs

Often graphs $G = \langle V, E/A, \ldots \rangle$ 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 $l(v) \in L$.

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.

Hypergraph

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

Definition

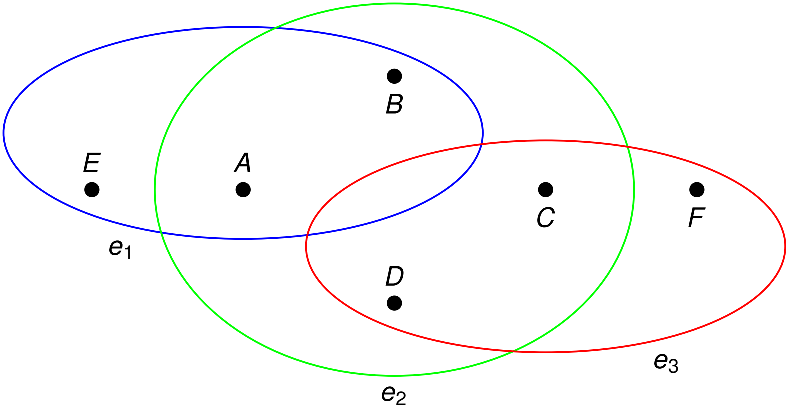
A hypergraph is a pair $H = \langle V, E \rangle$, where

- 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



A hypergraph

Computational Complexity

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 $f : \mathbb{N} \to \mathbb{R}$, $g \in M$.

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

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 $f : \mathbb{N} \to \mathbb{R}$, $g \in M$.

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

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

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

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)

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).

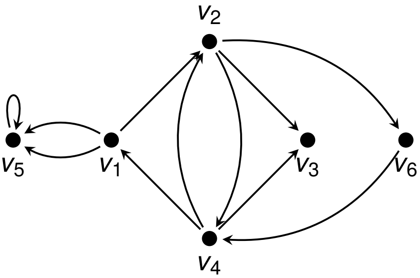
Let $G = \langle V, A, \alpha, \omega \rangle$ be a digraph.

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:

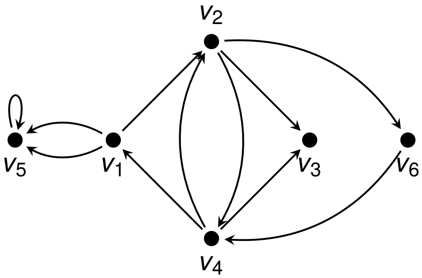


Adjacency matrix:

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

Adjacency list

Graph:



Adjacency list:

$1 \to 2 \to 5 \to 5$

$2 \to 6 \to 3 \to 4$

3

$4 \to 1 \to 2$

$5 \to 5$

$6 \to 4$

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

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

- Given: A problem instance $x \in X$
- 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?

Decision problem

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

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

- Given: A problem instance $x \in X$
- 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 X be a set of problem instances, S be the set of solution candidates, and R be a binary relation $R \subseteq X \times S$.

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

- Given: A problem instance $x \in X$
- 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")!

Search problem

Let X be a set of problem instances, S be the set of solution candidates, and R be a binary relation $R \subseteq X \times S$.

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

- Given: A problem instance $x \in X$
- 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 X be a set of problem instances, S be the set of solution candidates, R be a binary relation $R \subseteq X \times S$, and $f : S \to \mathbb{R}$ be an objective function.

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

- Given: A problem instance $x \in X$
- 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")!

Optimization problem

Let X be a set of problem instances, S be the set of solution candidates, R be a binary relation $R \subseteq X \times S$, and $f : S \to \mathbb{R}$ be an objective function.

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

- Given: A problem instance $x \in X$
- 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 (DTM) in polynomial time

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

Alternative characterization of NP

NP: class of decision problems X such that there exists a polynomial-time verifier for X.

A polynomial-time verifier for X is a polynomial-time DTM M such that there exists a polynomial $p : \mathbb{N} \to \mathbb{N}$ with:

a given x is a YES-instance of X, there exists a witness (or: certificate) s with $|s| \leq p(|x|)$ such that M accepts (x,s), and

b given x is a NO-instance of X, for any candidate s with $|s| \leq p(|x|)$, M rejects (x,s).

NP-completeness

Consider decision problems X and X' 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 $f : \Sigma' \to \Sigma$ such that $w \in L' \iff f(w) \in L$.

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 proposional 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 $c : V \to \{1, 2, 3\}$ 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 3CNF-SAT $\leq_p$ 3-COLORABILITY, each problem in NP can also be reduced to 3-COLORABILITY.

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 $c : V \to \{1,2,3\}$ 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.

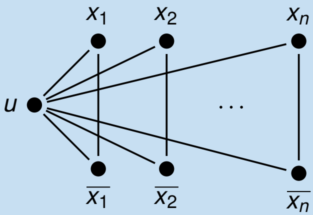
We now show that 3-COLORABILITY is NP-hard by a polynomial reduction from 3CNF-SAT. Since 3CNF-SAT is NP-complete, each problem in NP can be reduced to 3CNF-SAT and via 3CNF-SAT $\leq_p$ 3-COLORABILITY, each problem in NP can also be reduced to 3-COLORABILITY.

3-Colorability

We construct a function that assigns to each 3CNF-SAT formula $\varphi = C_1 \wedge \cdots \wedge C_m$ a graph G_φ such that

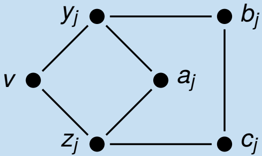
$$\varphi \text{ is satisfiable } \iff G_\varphi \text{ has a coloring with colors } \{\text{red}, \text{blue}, \text{green}\}.$$

We assume (w.l.o.g.) that each clause C_j consists of exactly three literals, i.e., $C_j = (l_{1j} \vee l_{j2} \vee l_{j3})$. Let $x_1, \ldots, x_n$ be the set of proposional 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:

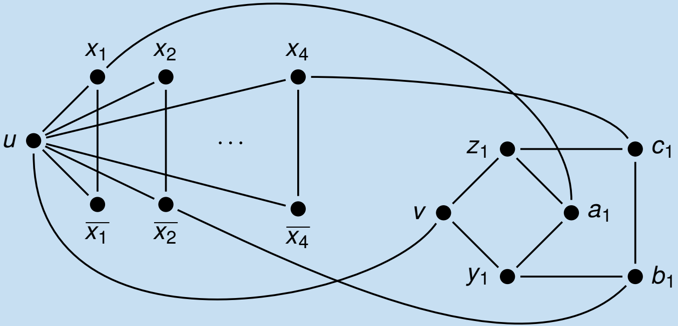


Vertices in G_j are connected by an edge to vertices in G_T as follows:

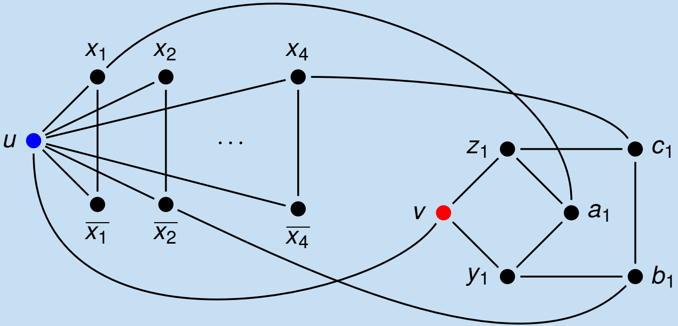
- an edge $\{u, v\}$
- edges $\{a_j, l_{j1}\}$, $\{b_j, l_{j2}\}$, $\{c_j, l_{j3}\}$ ($1 \leq j \leq m$)

3-Colorability

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



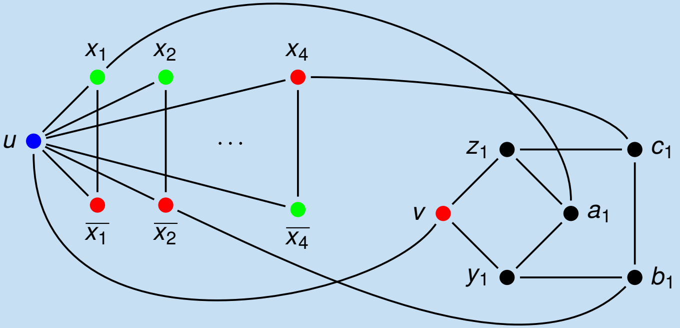
For example, if $\varphi = (x_1 \vee \neg x_2 \vee x_4) \wedge \ldots$, G_φ contains the following subgraph G_1:



Assume now that φ is satisfied by a truth function V. Define:
$c(u)$ = blue, $c(v)$ = red, $c(x_i)$ = green and $c(\overline{x_i})$ = red, if $V(x_i) = 1$, and $c(x_i)$ = red and $c(\overline{x_i})$ = green, if $V(x_i) = 0$.

3-Colorability

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

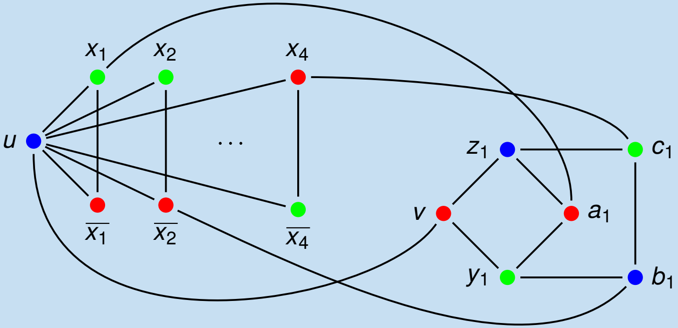


Assume now that φ is satisfied by a truth function V. Define:
$c(u)$ = blue, $c(v)$ = red, $c(x_i)$ = green and $c(\overline{x_i})$ = red, if $V(x_i) = 1$, and $c(x_i)$ = red and $c(\overline{x_i})$ = green, if $V(x_i) = 0$.

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

3-COLORABILITY

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

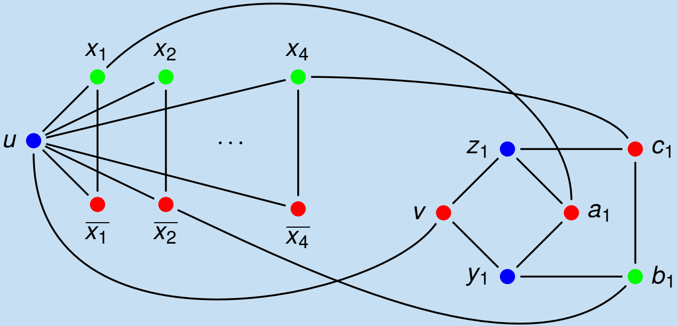


Assume now that φ is satisfied by a truth function V. Define:
$c(u)$ = blue, $c(v)$ = red, $c(x_i)$ = green and $c(\overline{x_i})$ = red, if $V(x_i) = 1$, and $c(x_i)$ = red and $c(\overline{x_i})$ = green, if $V(x_i) = 0$.

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

3-Colorability

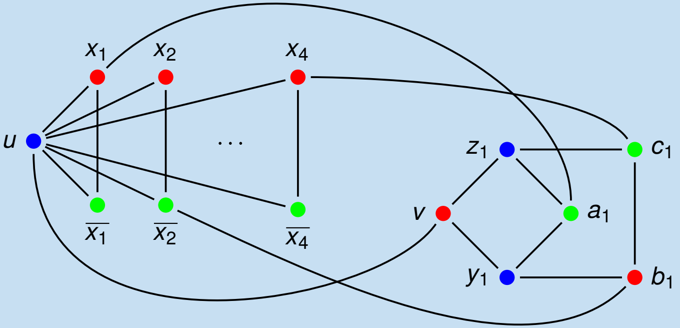
For example, if $\varphi = (x_1 \vee \neg x_2 \vee x_4) \wedge \ldots$, G_φ contains the following subgraph G_1:



Assume now that φ is satisfied by a truth function V. Define:
$c(u)$ = blue, $c(v)$ = red, $c(x_i)$ = green and $c(\overline{x_i})$ = red, if $V(x_i) = 1$, and $c(x_i)$ = red and $c(\overline{x_i})$ = green, if $V(x_i) = 0$.

For $V(x_1) = 1, V(x_2) = 1, V(x_4) = 1, \ldots, G_1$ can also be colored …

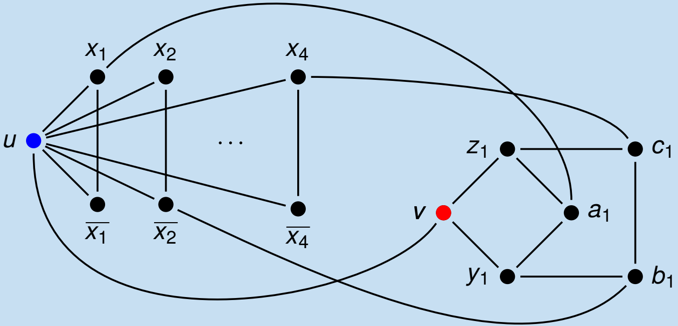
For example, if $\varphi = (x_1 \vee \neg x_2 \vee x_4) \wedge \ldots$, G_φ contains the following subgraph G_1:



Assume now that φ is satisfied by a truth function V. Define:
$c(u)$ = blue, $c(v)$ = red, $c(x_i)$ = green and $c(\overline{x_i})$ = red, if $V(x_i) = 1$, and $c(x_i)$ = red and $c(\overline{x_i})$ = green, if $V(x_i) = 0$.

...also for $V(x_1) = 0$, $V(x_2) = 0$, $V(x_4) = 0$ etc.

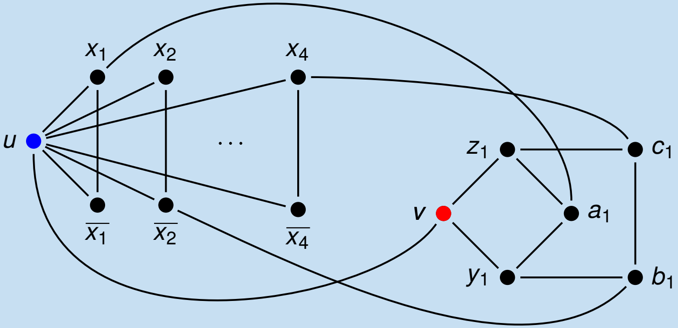
For example, if $\varphi = (x_1 \lor \neg x_2 \lor x_4) \land \ldots$, G_φ contains the following subgraph G_1:



For the other direction, assume that G_φ has a coloring c (w.l.o.g, $c(u)$ = blue and $c(v)$ = red). Define $V(x_i) = 1$ if $c(x_i)$ = green, and $V(x_i) = 0$ if $c(x_i)$ = red.

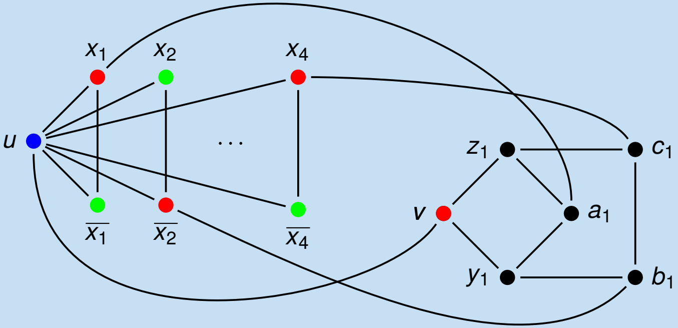
3-Colorability

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



For the other direction, assume that G_φ has a coloring c (w.l.o.g, $c(u) =$ blue and $c(v) =$ red). Define $V(x_i) = 1$ if $c(x_i) =$ green, and $V(x_i) = 0$ if $c(x_i) =$ red. This is a truth function $V\colon \{x_1, \ldots, x_n\} \to \{0, 1\}$, since all x_i-nodes are red or green (because u is colored blue).

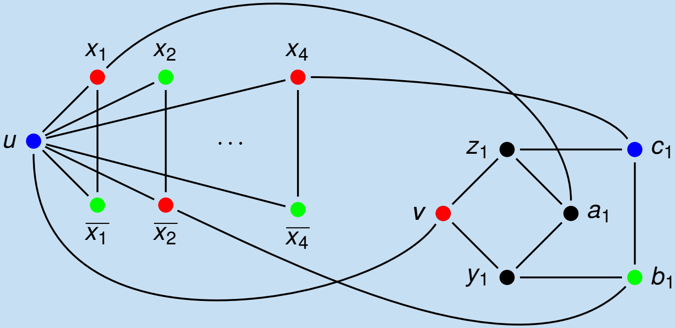
For example, if $\varphi = (x_1 \vee \neg x_2 \vee x_4) \wedge \ldots$, G_φ contains the following subgraph G_1:



. . . Assume V does not satisfy φ. Then there is clause, say C_1, with $V \not\models C_1$, i.e., all literals in C_1 are false.

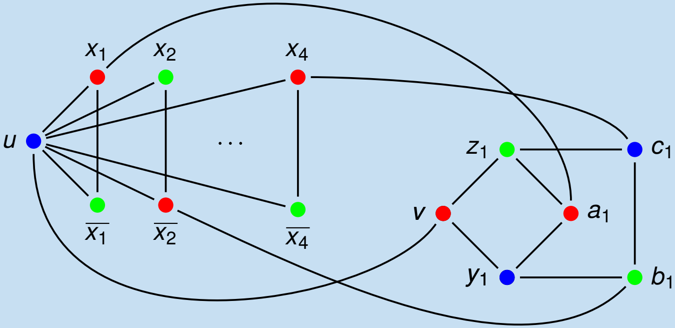
3-Colorability

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



... Assume V does not satisfy φ. Then there is clause, say C_1, with $V \not\models C_1$, i.e., all literals in C_1 are false. Then a_1, b_1, c_1 must be colored blue or green. If, w.l.o.g., $c(b_1)$ = green and $c(c_1)$ = blue, ...

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



... Assume V does not satisfy φ. Then there is clause, say C_1, with $V \not\models C_1$, i.e., all literals in C_1 are false. Then a_1, b_1, c_1 must be colored blue or green. If, w.l.o.g., $c(b_1)$ = green and $c(c_1)$ = blue, ... then a_1 must be colored red, a contradiction.

UNI FREIBURG

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

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

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: 3CNF-SAT $\leq_p$ k-COLORABILITY, for $k \geq 3$.
- The corresponding search problem "Given a graph, find a 3-coloring ..." is in the complexity class Function NP (FNP).

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

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

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: 3CNF-SAT $\leq_p$ k-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

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.