Network Models

Ufuk Bahçeci

v0.23.10.02

Network Models

MIT License

Network Models
Author

- Ufuk Bahçeci, Ph. D. (Industrial Engineering, University of Galatasaray)

Table of Contents

Graph

Definition

Graph

Given a list of locations, a graph is a structured representation of the locations and the relationships between them.

Network Flow

Definition

Network flow

Network flow is the sending of a certain amount of assets from one location to another on the graph.

Mathematical Programming

Definition

Mathematical programming

Mathematical programming is the optimization of problems formulated as minimization (or maximization) of an objective function subject to a set of constraints.

Combinatorial Optimization

Definition

Combinatorial optimization

Combinatorial optimization is a class of mathematical programming, where optimization is performed over a discrete set of feasible solutions.

Network Flow Problem

Definition

Network flow problem

Network flow problems are mathematical programming problems that can be converted into combinatorial optimization problems dealing with network flows.

- Linear programming
  - Simplex algorithm
  - Duality
- Decomposition methods
  - Dantzig-Wolfe (complicating constraints, column(extreme point) generation, duality gap between upper and lower bounds)
  - Benders (complicating variables, cut generation, duality gap between upper and lower bounds)
- Mixed-integer programming
  - Branch-and-bound (BaB)
  - BaB + Cutting planes = Branch-and-cut
  - BaB + Column(variable for pricing, extreme point for decomposition) generation = Branch-and-price
  - BaB + Cutting planes + Column generation = Branch-price-and-cut

Mathematical Optimization

- Constraint programming
  - Constraint propagation
  - Domain reduction
- Combinatorial optimization
  - Some problems are easy to solve
    - Special fast algorithms
  - Some problems are hard to solve
    - Mixed-integer programming
    - Heuristics

Motivations

Network Flow Problems

- Network flow problems
  - Combinatorial optimization
  - Wide application area in Operations Research
  - Special fast algorithms suitable for large problem instances
  - Network flow problem as an embedded subproblem

Graph

Definition

Graph [1]

A graph $G(V, E)$ consists of a set of vertices V and edges E. Edges are used to model the relationship between vertices.

Graph

Definition

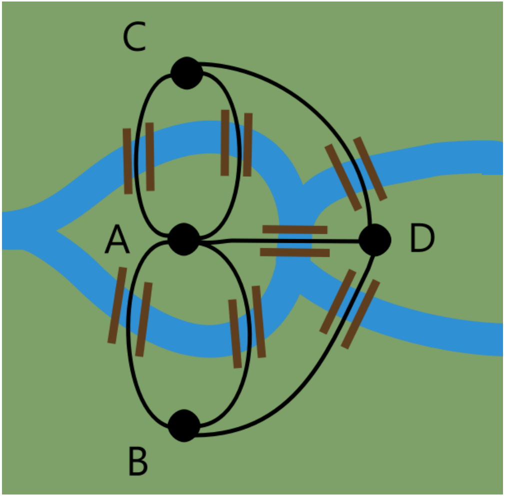
Graph [2]

A graph $G(N, A)$ consists of a set of nodes N and arcs A. Arcs are used to model the relationship between nodes.

Subgraph

A graph $G'(V', E')$ is a subgraph of $G(V, E)$ if $V' \subset V$ and $E' \subset E$.

Graph
Example



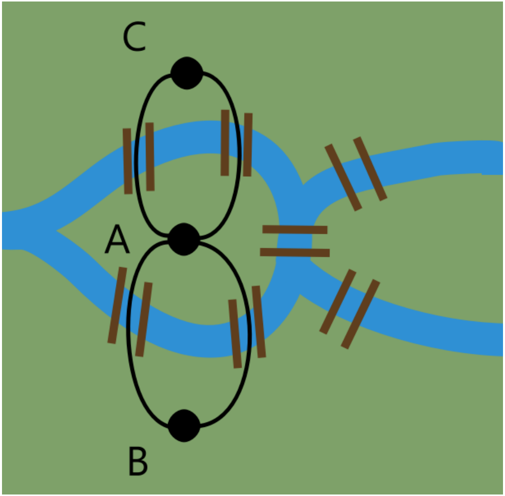
Graph

The Euler's problem

- Is it possible to start from a vertex, move along all edges, traversing every edge only once, and finally return to the starting vertex?

Graph

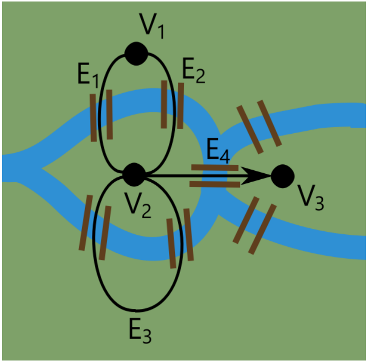
Example



- Is it possible to start from a vertex, visit each of all vertices exactly once, and finally return to the starting vertex?

Graph

Directed edges, multiple edges and loops



- E_1 and E_2 are multiple edges
- E_3 is a loop
- E_4 is a directed edge
- V_2(tail) and V_3(head) are the endpoints of the edge(arc) E_4.

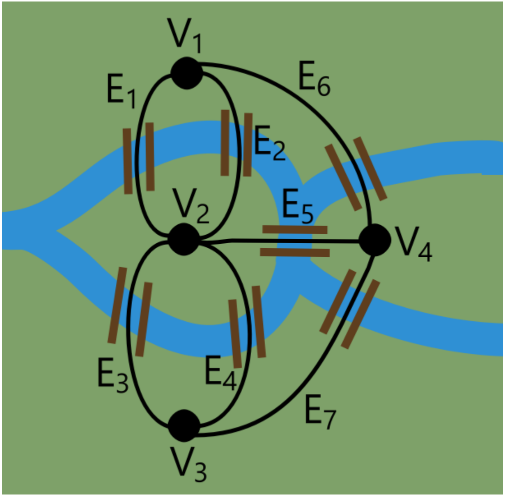
Graph

Graph types [1]

Type	Edges	Multiple edges	Loops
Simple graph	Undirected	✘	✘
Multigraph	Undirected	✔	✘
Pseudograph	Undirected	✔	✔
Simple directed graph	Directed	✘	✘
Directed multigraph	Directed	✔	✔
Mixed graph	Directed and undirected	✔	✔

Graph

A multigraph



Graph

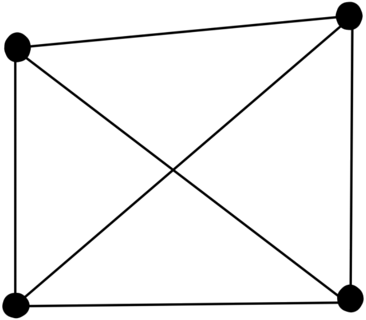
Definitions

Complete graph [1]

Complete graph is a simple graph where each pairs of distinct vertices are connected.

Graph

A complete graph



Graph

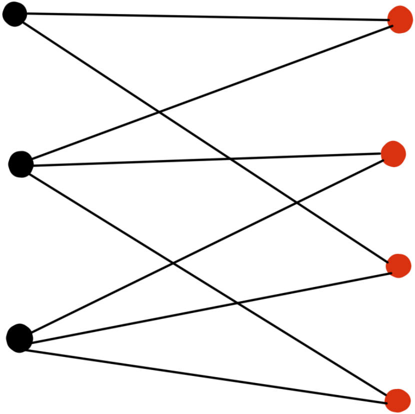
Definitions

Bipartite simple graph [1]

A simple graph $G(V, E)$ is bipartite if $\exists V_1, V_2 : V_1 \cap V_2 = \emptyset$ and $V_1 \cup V_2 = V$ such that every edge in E connects a vertex in V_1 to a vertex in V_2.

Graph

A bipartite simple graph



Matching [1]

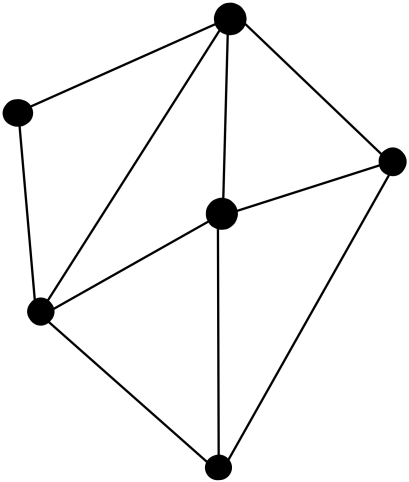
A matching M in a simple graph $G(V, E)$ is a subset of E, i.e. $M \subseteq E$ such that $\forall m, m' \in M$, all the endpoints of m and m' are distinct vertices.

Maximal matching

The maximal matching of G is the matching with the largest $|M|$.

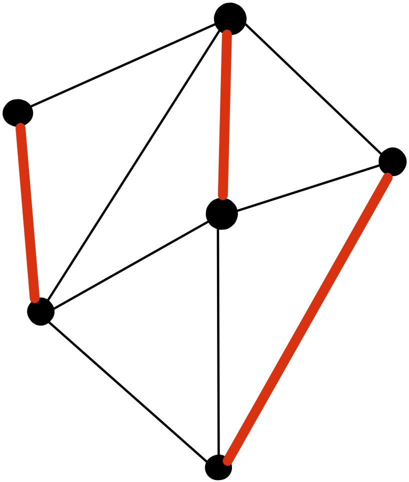
Graph

A simple graph



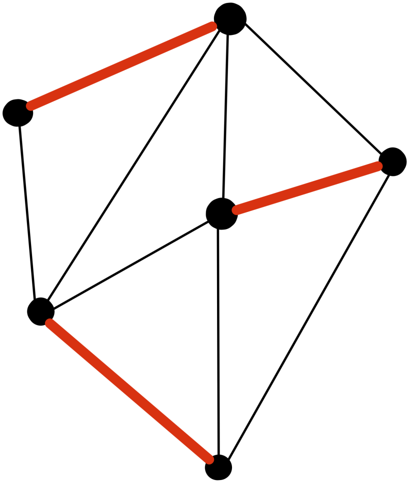
Graph

A maximal matching



Graph

Another maximal matching



Graph
Definitions

Adjacent vertices in an undirected graph

Two vertices are adjacent in an undirected graph G if they are endpoints of an edge in G.

Adjacent vertices in a directed graph

In a directed graph G, the vertex v_1 is adjacent to the vertex v_2 if they are endpoints of a directed edge $E(v_1, v_2)$ in G.

An edge of an undirected graph G is incident with the vertices that are endpoints of this edge.

Graph
Definitions

Degree of a vertex in an undirected graph [1]

The degree of a vertex v in an undirected graph G, $deg(v)$ is equal to the number of edges incident with the vertex v, where a loop is equivalent to two edges.

Graph

Definitions

Given an undirected graph $G(V, E)$

$\sum_{v \in V} deg(v) = 2|E|$

Degree of a vertex in a directed graph [1]

The indegree(outdegree) of a vertex v in a directed graph G, $deg^{-}(v)(deg^{+}(v))$ is equal to the number of edges with v as their terminal(initial) vertex.

Graph

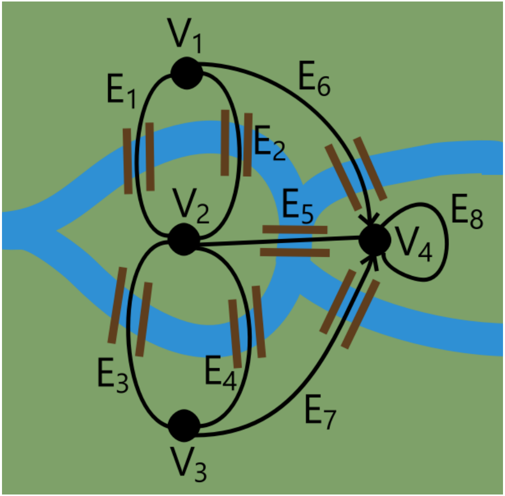
Definitions

Given a directed graph $G(V, E)$

$$\sum_{v \in V} deg^{-}(v) = \sum_{v \in V} deg^{+}(v) = |E|$$

Graph

A mixed graph

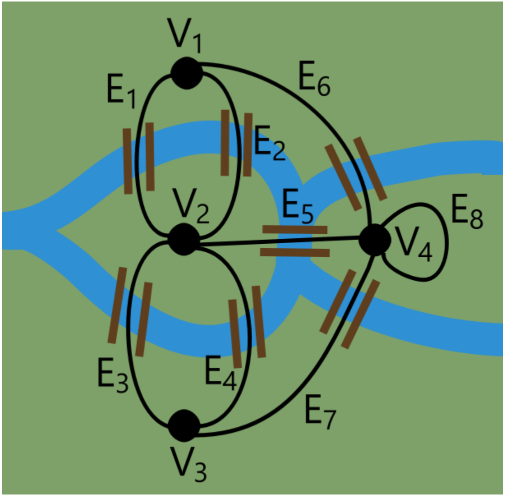


Graph
Adjacency matrix

	V_1	V_2	V_3	V_4
V_1	0	2	0	1
V_2	2	0	2	1
V_3	0	2	0	1
V_4	0	1	0	1

Graph

A pseudograph



Graph

Incidence matrix

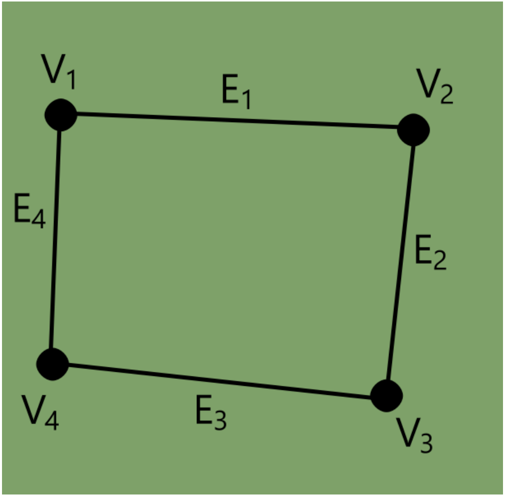
	E_1	E_2	E_3	E_4	E_5	E_6	E_7	E_8
V_1	1	1	0	0	0	1	0	0
V_2	1	1	1	1	1	0	0	0
V_3	0	0	1	1	0	0	1	0
V_4	0	0	0	0	1	1	1	1

Isomorphism of graphs [1]

Two simple graphs $G(V, E)$ and $G'(V', E')$ are isomorphic if and only if there exists a permutation of V', denoted as V'^p, leading to $G'^p(V'^p, E')$, where G and G'^p have the same adjacency matrix.

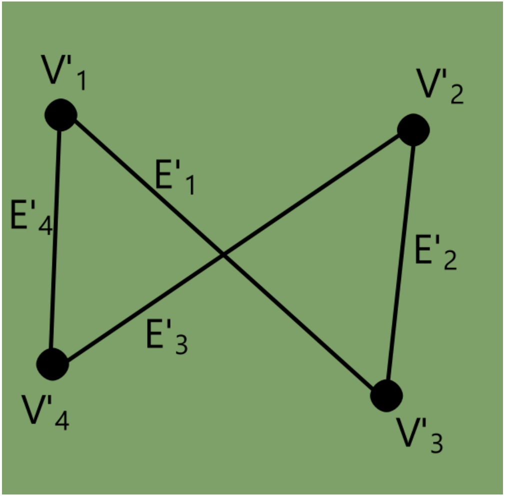
Graph

$G(V, E)$



Graph

$G'(V', E')$



Walk [2]

A walk is a series of vertices that are connected to each other by means of edges.

Simple walk (trail) [1]

A simple walk (trail) is a walk that does not contain the same edge more than once.

Directed walk [2]

A directed walk is a series of vertices that are connected to each other by means of edges in a way that respects the edge directions.

Path [2], [1]

A path is a walk that visits each vertex in the walk only once. A path is also a trail.

Directed path [2]

A directed path is a directed walk that visits each vertex in the directed walk only once.

Circuit [2], [1]

A circuit(closed walk) is a walk of length strictly positive that starts and ends at the same vertex. A simple circuit does not contain the same edge more than once.

Cycle [2]

A cycle is a closed path.

Directed circuit

A directed circuit(closed directed walk) is a directed walk of length strictly positive that starts and ends at the same vertex. A simple directed circuit does not contain the same edge more than once.

Directed cycle [2]

A directed cycle is a directed closed path.

Graph
Definitions

Connected [1]

An undirected graph $G(V, E)$ is connected when a walk exists between each pair of vertices $v, v' \in V^2$ and $v \neq v'$.

Graph

Definitions

Connected [1]

An directed graph $G(V, E)$ is strongly connected when a directed walk exists between each pair of vertices $v, v' \in V^2$ and $v \neq v'$. Let $G'(V', E')$ be the underlying undirected graph. G is weakly connected if G' is connected.

Network [2]

A network is a graph where vertices and edges have associated properties in the form of numerical values.

Graph

Definitions

The length of a walk [1]

The length of a walk is equal to the sum of the weights of its edges.

The number of walks [1]

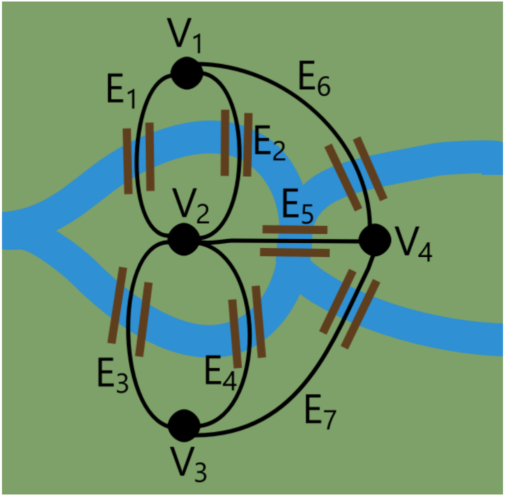
Let A be the adjacency matrix of a graph $G(V, E)$, then the cell with index (i, j) of the matrix A^d is equal to the number of walks of length $d \in \mathbb{Z}^+$ from v_i to v_j, where v_i,$v_j \in V^2$.

Euler walk and circuit [1]

A simple circuit traversing all edges of a graph G is an Euler circuit.
Similarly, a simple walk traversing all edges of a graph G is an Euler walk.

Graph

Can you find an Euler circuit in this multigraph?



An Euler circuit exists..[1]

An Euler circuit exists in a connected multigraph $G(V, E)$ with $|V| \geq 2$ if and only if $\forall v \in V,\ deg(v) \equiv 0\,(mod\,2)$.

Graph

Definitions

An Euler walk exists..[1]

An Euler walk but not an Euler circuit exists in a connected multigraph $G(V, E)$ if and only if $\exists\, v', v'' \in V^2$, $v' \neq v''$, $deg(v') \equiv 1\,(mod\, 2)$, $deg(v'') \equiv 1\,(mod\, 2)$, and $\forall v \in V \setminus \{v', v''\}$, $deg(v) \equiv 0\,(mod\, 2)$.

Chinese postman (route inspection) problem

Chinese postman problem looks for the shortest circuit traversing every edge of a connected multigraph at least once.

Chinese postman problem

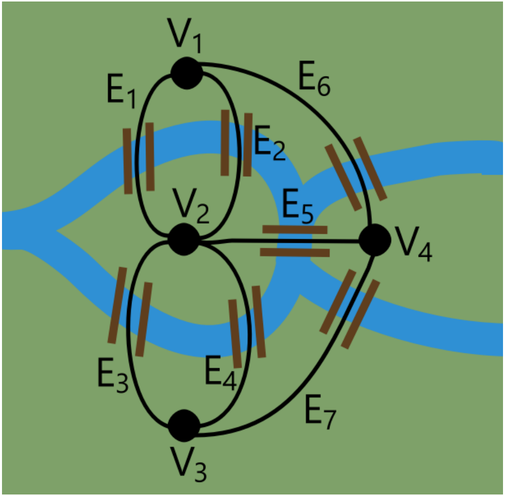
What if an Euler circuit exists in a connected multigraph?

Hamilton path and cycle [1]

A simple circuit visiting every vertex of a graph G exactly once is an Hamilton cycle. Similarly, a simple walk visiting every vertex of a graph G exactly once is an Hamilton path.

Graph

Can you find an Hamilton cycle in this multigraph?



Graph

Definitions (Dirac's theorem)

An Hamilton cycle exists..[1]

An Hamilton cycle exists in a graph $G(V, E)$ if G is a simple graph with $|V| \geq 3$ and $\forall v \in V$, $deg(v) \geq \frac{|V|}{2}$.

Traveling salesman problem

Traveling salesman problem looks for the shortest circuit visiting every vertex of a connected graph exactly once.

Traveling salesman problem

What about the feasible solutions of a traveling salesman problem if it is defined on a complete simple graph with more than 3 vertices? Is this problem feasible?

Graph

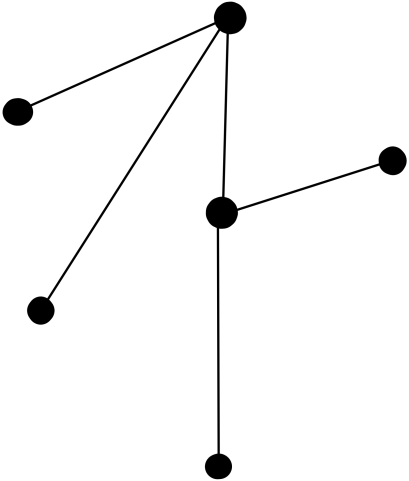
Definitions

Tree [2]

A connected graph that contains no cycle is called tree.

Graph

A tree

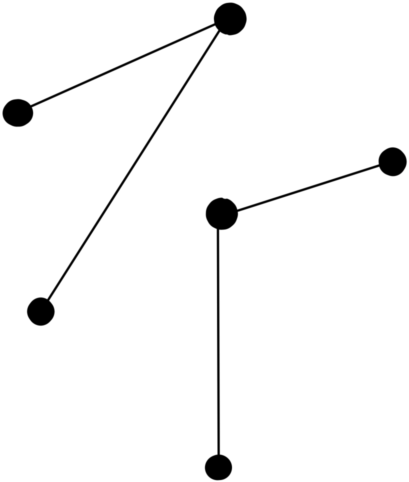


Forest [2]

A collection of trees is called forest.

Graph

A forest



The number of edges in a tree

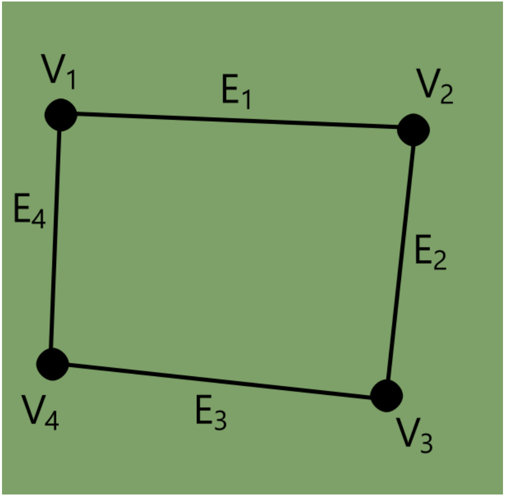
If the graph $G(V, E)$ is a tree than $|E| = |V| - 1$

Planar graph [1]

A planar graph can be drawn in two dimensions without any edges intersecting each other.

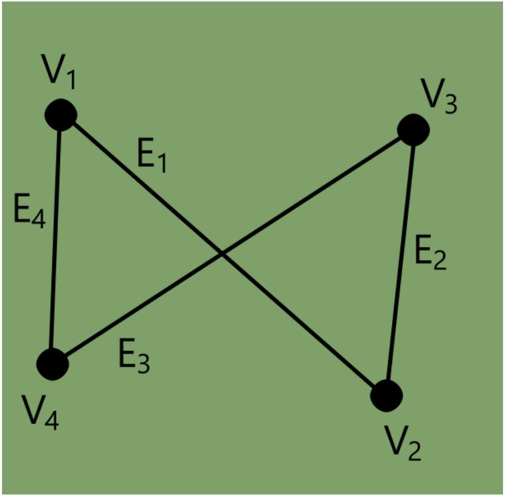
Graph

Planar representation of a planar graph



Graph

Non-planar representation of a planar graph



Graph

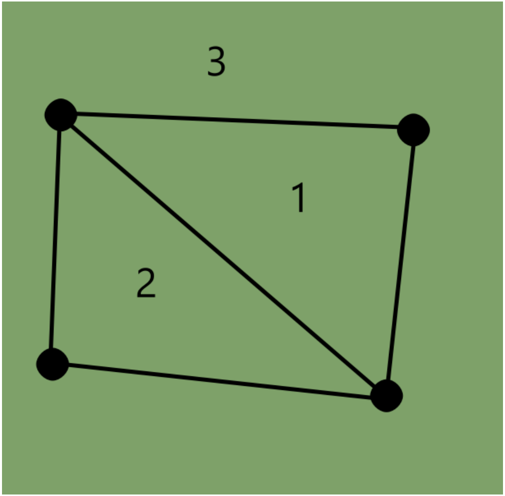
Definitions

Euler's formula [1]

A connected planar simple graph $G(V, E)$ has $|E| - |V| + 2$ regions.

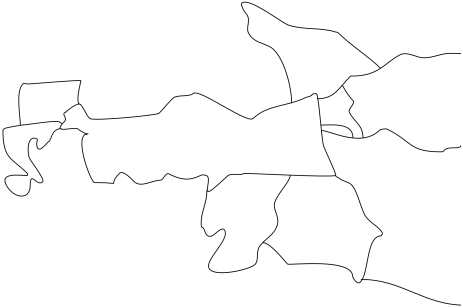
Graph

$3(= 5 - 4 + 2)$ regions of a planar graph



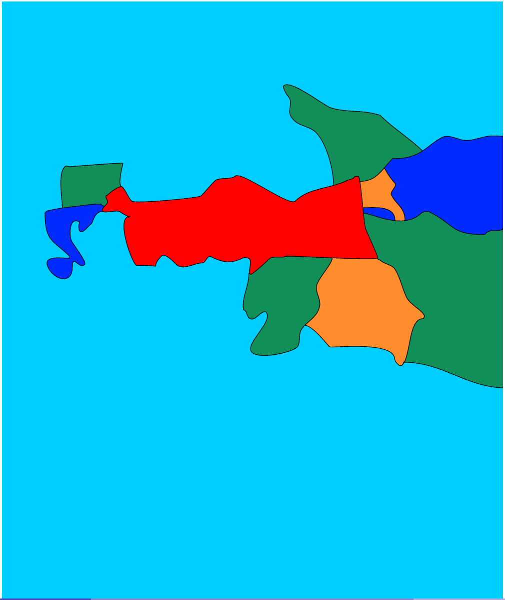
Graph

Map coloring example



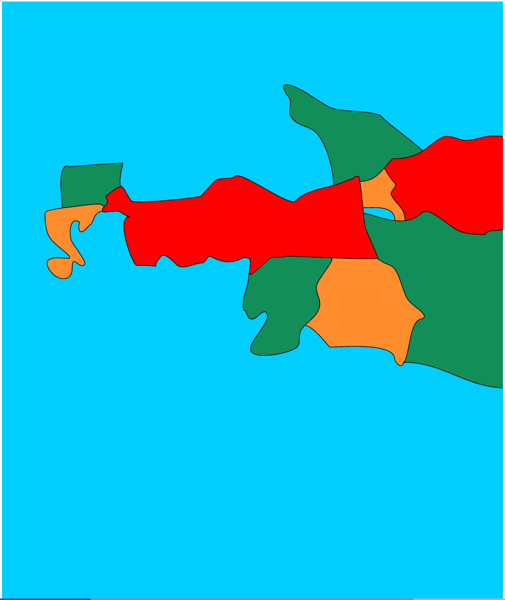
Graph

Map coloring example I (5 colors)



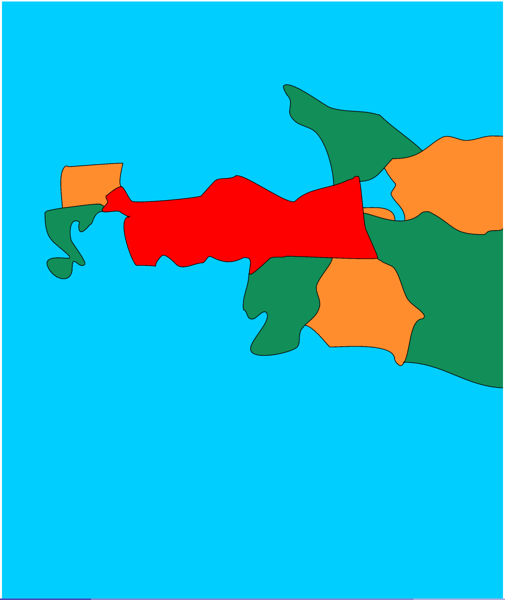
Graph

Map coloring example II (4 colors)



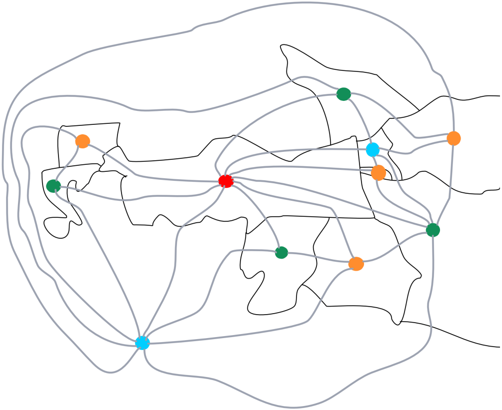
Graph

Map coloring example III (4 colors)



Graph

Dual graph (III) (4 colors)

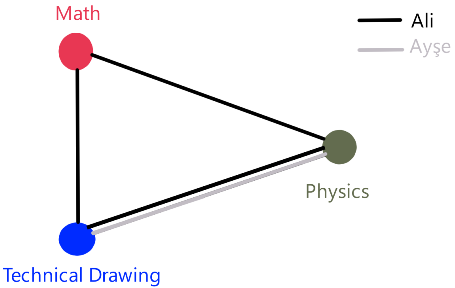


The four color theorem [1]

The chromatic number (minimum number of colors) of a planar simple graph ≤ 4.

Graph

Graph coloring example



Minimum cost flow problem

Let $G(V, E)$ be a directed graph with costs $c_{vv'}$ and capacities $u_{vv'}$ defined on edges $vv' = e \in E$, where $v \neq v'$, v and $v' \in V$. Let $b_v > 0$ be the supply and $b_v < 0$ be the demand associated with each vertex $v \in V$. Moreover, $x_{vv'}$ denotes the amount of flow from a vertex v to another vertex v'. Then, minimum cost flow problem minimizes the total cost incurred from all flows in G satisfying both flow conservation constraints and flow limits.

Minimum cost flow problem

$$
\begin{aligned}
& min \sum_{vv' \in E} c_{vv'} x_{vv'} \\
& s.t. \sum_{v' : vv' \in E} x_{vv'} - \sum_{v' : v'v \in E} x_{v'v} = b_v && \forall v \in V \\
& \qquad 0 \leq x_{vv'} \leq u_{vv'} && \forall vv' \in E
\end{aligned}
$$

Network Problems

Minimum cost flow problem

Assumptions [2]

- $\forall e \in E,\ c_e \in \mathcal{Z}_0^+$
- $\forall v \in V,\ b_v \in \mathcal{Z}$ and $\sum_v b_v = 0$
- $\forall e \in E,\ u_e \in \mathcal{Z}_0^+$
- $\forall v, v' \in V^2$, $\exists$ an uncapaciated directed path from v to v'

Network Problems

Definitions

Polynomial time algorithm

A polynomial time algorithm has a running time polynomial in the length (number of bits) of the input.

Pseudo-polynomial time algorithm

A pseudo-polynomial time algorithm has a running time polynomial in the numeric value (largest value) of the input.

Pseudo-polynomial time algorithms [2]

- Cycle-canceling with $\mathcal{O}(|E|CU)$ iterations
- Successive shortest path with $\mathcal{O}(|V|U)$ iterations
- Primal-dual algorithm with $\mathcal{O}(min(|V|U, |V|C))$ iterations
- Out-of-kilter with $\mathcal{O}(|V|U)$ iterations
- Relaxation

where, $c_e \leq C, \forall e \in E$ and $u_e \leq U, \forall e \in E$

Network Problems

Minimum cost flow problem

Complexity of some minimum cost flow algorithms [3]

- Ford and Fulkerson, $\mathcal{O}(|V|^4 CU)$
- Out-of-kilter, $\mathcal{O}(|E|^3 U)$
- Successive shortest path, $\mathcal{O}(|V|^2 |E| U)$
- Cycle-cancelling, $\mathcal{O}(|V||E|^2 CU)$
- Cost-scaling (generic), $\mathcal{O}(|V|^2 |E| log(|V| C))$
- Cancel-and-tighten (dynamic trees), $\mathcal{O}(|V||E| log(|V|) min(log(|V| C, |E| log(|V|))))$
- Primal network simplex (dynamic trees), $\mathcal{O}(|V||E| log(|V|) min(log(|V| C, |E| log(|V|))))$
- Dual network simplex (Orlin), $\mathcal{O}(|E|(|E| + |V| log |V|) min(log(|E| U), |E| log(|V|)))$
- Dual network simplex (Armstrong and Jin), $\mathcal{O}(|V||E| log |V| (|E| + |V| log |V|))$

Study of minimum cost flow algorithms [3]

Cost-scaling and primal network simplex were both efficient and robust.

Study of seven state-of-the-art algorithms [4]

- Simple cycle canceling
- Minimum mean cycle canceling
- Cancel and tighten
- Successive shortest path
- Capacity scaling
- Network simplex
- Cost scaling

where, network simplex was the fastest algorithm in $\approx 75\%$ of the studied cases

Maximum flow problem

Let $G(V, E)$ be a directed graph with capacities $u_{vv'} \geq 0$ defined on edges $vv' = e \in E$, where $v \neq v'$, v and $v' \in V$. Let $b_v > 0$ be the supply and $b_v < 0$ be the demand associated with each vertex $v \in V$. Moreover, $x_{vv'}$ denotes the amount of flow from a vertex v to another vertex v'. Then, maximum flow problem maximizes the amount of flow from the source vertex $s \in V$ to the sink vertex $t \in V$, $s \neq t$, and all flows in G satisfy both flow conservation constraints and flow limits.

Maximum flow problem

$$
\begin{aligned}
& max \quad \alpha \\
& s.t. \quad \sum_{v':vv' \in E} x_{vv'} - \sum_{v':v'v \in E} x_{v'v} = \begin{cases} \alpha & for \quad v = s \\ 0 & \forall v \in V \setminus \{s, t\} \\ -\alpha & for \quad v = t \end{cases} \\
& \qquad 0 \leq x_{vv'} \leq u_{vv'} \qquad \forall vv' \in E
\end{aligned}
$$

Network Problems

Maximum flow problem

Special case of minimum cost flow problem

- Maximum flow problem from s to t on $G(V, E)$
- Add $b_v = 0, \forall v \in V$
- Add $c_e = 0, \forall e \in E$
- Add a new edge ts with $c_{ts} = -1$ and $u_{ts} = \infty$
- $E' = E \cup \{ts\}$
- Minimum cost flow problem on $G'(V, E') \equiv$ Maximum flow problem on $G(V, E)$

Network Problems

Maximum flow problem

Assumptions [2]

- $\forall e \in E$, $u_e \in \mathcal{Z}_0^+$
- $\nexists$ an uncapaciated directed path from s to t
- If $vv' \in E$ than $v'v \in E$
- No multiple edges

Running times of maximum flow algorithms [2]

- Labeling, $\mathcal{O}(|V||E|U)$
- Capacity scaling, $\mathcal{O}(|V||E|log(U))$
- Successive shortest path, $\mathcal{O}(|V|^2|E|)$
- Generic preflow-push, $\mathcal{O}(|V|^2|E|)$
- FIFO preflow-push, $\mathcal{O}(|V|^3)$
- Highest-label preflow-push, $\mathcal{O}(|V|^2\sqrt{|E|})$
- Excess scaling, $\mathcal{O}(|V||E| + |V|^2 log(U))$

Network Problems

Minimum cost flow and maximum flow problems

Running time of an almost linear time algorithm [5] for minimum cost flows and maximum flows

- Demands, costs and capacities are bounded polynomially
- Demands, costs and capacities are integral
- Runs in $m^{1+\mathcal{O}(1)}$ time

Feasible flow problem

$$
\begin{aligned}
&\sum_{v':vv'\in E} x_{vv'} - \sum_{v':v'v\in E} x_{v'v} = b_v && \forall v \\
&0 \leq x_{vv'} \leq u_{vv'} && \forall vv' \in E
\end{aligned}
$$

Network Problems

Maximum flow problem

Procedure to create a transformed network $G'(V', E')$ [2]

- Add the vertex s
- $\forall v \in V$ with $b_v > 0$, add the edges sv with $u_{sv} = b_v$
- Add the vertex t
- $\forall v \in V$ with $b_v < 0$, add the edges vt with $u_{vt} = -b_v$
- $V' = V \cup \{s, t\}$
- $E' = E \cup \{sv : v \in V,\ b_v > 0\} \cup \{vt : v \in V,\ b_v < 0\}$

Network Problems

Maximum flow problem

Maximum flow problem on the transformed network $G'(V', E')$

$$\max \quad \alpha$$

$$s.t. \quad \sum_{v':vv' \in E'} x_{vv'} - \sum_{v':v'v \in E'} x_{v'v} = \begin{cases} \alpha & \text{for} \quad v = s \\ 0 & \forall v \in V' \setminus \{s, t\} \\ -\alpha & \text{for} \quad v = t \end{cases}$$

$$0 \leq x_{vv'} \leq u_{vv'} \qquad \forall vv' \in E'$$

Feasible flow problem

If α^* of the maximum flow problem on the transformed network $G'(V', E')$ is equal to $\sum\limits_{v \in V,\, b_v > 0} b_v$ than the flow problem is feasible.

Network Problems

Maximum flow problem

Maximum flow problem with lower bounds on $G(V, E)$

$$
\begin{aligned}
& max \quad \alpha \\
& s.t. \quad \sum_{v':vv' \in E} x_{vv'} - \sum_{v':v'v \in E} x_{v'v} = \begin{cases} \alpha & for \quad v = s \\ 0 & \forall v \in V \setminus \{s, t\} \\ -\alpha & for \quad v = t \end{cases} \\
& \qquad l_{vv'} \leq x_{vv'} \leq u_{vv'} \qquad \forall vv' \in E
\end{aligned}
$$

Procedure to create a circulation network $G^c(V, E^c)$ [2]

- Add the edge ts with $u_{ts} = \infty$
- $E^c = E \cup \{ts\}$

so that it is possible to send the flow from s to t back to s from t by using the edge ts with $u_{ts} = \infty$.

Circulation problem (a feasible flow of the maximum flow problem with lower bounds) [2]

$$\sum_{v':vv'\in E^c} x_{vv'} - \sum_{v':v'v\in E^c} x_{v'v} = 0 \qquad \forall v \in V$$

$$l_{vv'} \leq x_{vv'} \leq u_{vv'} \qquad \forall vv' \in E^c$$

Network Problems

Maximum flow problem

Transformed ($x_{vv'} = x'_{vv'} + l_{vv'}$) circulation problem [2]

$$\sum_{v':vv' \in E^c} x'_{vv'} - \sum_{v':v'v \in E^c} x'_{v'v} = b_v \qquad \forall v \in V$$

$$0 \leq x'_{vv'} \leq u_{vv'} - l_{vv'} \qquad \forall vv' \in E^c$$

where $b_v = \sum_{v':v'v \in E^c} l_{v'v} - \sum_{v':vv' \in E^c} l_{vv'} \qquad \forall v \in V$

Feasible flow problem

A feasible flow can be found by solving a maximum flow problem on the transformed network $G^{c\prime}(V', E^{c\prime})$.

Network Problems

Maximum flow problem

Residual capacities on $G(V, E)$ [2]

A residual capacity of an edge vv' is denoted as
$r_{vv'} = (u_{vv'} - x_{vv'}) + (x_{v'v} - l_{v'v})$, where $x_{vv'}$'s and $x_{v'v}$'s are the feasible flows found in the previous step.

Maximum flow problem with residual capacities on $G(V, E)$

Solve the maximum flow problem with residual capacities on $G(V, E)$. Note that the residual capacity $r_{vv'}$ denotes the maximum possible increase in flow for the edge vv'.

Find the solution of the maximum flow problem with lower bounds

Find the solution of the maximum flow problem with lower bounds on $G(V, E)$ by increasing feasible flows found in the feasible flow problem by values from the maximum flow problem with residual capacities.

Minimum value problem [2] with lower bounds on $G(V, E)$

$$
\begin{aligned}
&min \quad \alpha \\
&s.t. \quad \sum_{v': vv' \in E} x_{vv'} - \sum_{v': v'v \in E} x_{v'v} = \begin{cases} \alpha & for \quad v = s \\ 0 & \forall v \in V \setminus \{s, t\} \\ -\alpha & for \quad v = t \end{cases} \\
&\qquad l_{vv'} \leq x_{vv'} \leq u_{vv'} \qquad \forall vv' \in E
\end{aligned}
$$

Solution method for minimum value problem

First find a feasible flow. Than solve the maximum flow problem, where capacities $r^{inv}_{vv'}$ are equal to $(x_{vv'} - l_{vv'}) + (u_{v'v} - x_{v'v})$. Note that the capacity $r^{inv}_{vv'}$ denotes the maximum possible decrease in flow for the edge vv'. Finally, the solution of the minimum value problem with lower bounds on $G(V, E)$ can be found by decreasing feasible flows by values from the maximum flow problem with capacities $r^{inv}_{vv'}$.

Network Problems

A few libraries

LEMON Graph Library (C++)

Library for Efficient Modeling and Optimization in Networks

NetworkX

A Python library for graphs and networks

Compressed sparse graph routines (scipy.sparse.csgraph)

Fast graph algorithms

References I

[1] K. Rosen, *Discrete Mathematics and Its Applications*. McGraw-Hill, 2007.

[2] R. Ahuja, T. Magnanti, and J. Orlin, *Network Flows: Theory, Algorithms, and Applications*. Prentice Hall, 1993.

[3] P. Kovács, "Minimum-cost flow algorithms: An experimental evaluation," *Optimization Methods and Software*, vol. 30, no. 1, pp. 94–127, 2015. DOI: https://doi.org/10.1080/10556788.2014.895828.

[4] P. Herrmann, A. Meyer, S. Ruzika, L. E. Schäfer, and F. von der Warth, "A machine learning based algorithm selection method to solve the minimum cost flow problem,", 2022. arXiv: 2210.02195 [cs.LG].

References II

[5] L. Chen, R. Kyng, Y. P. Liu, R. Peng, M. P. Gutenberg, and S. Sachdeva, "Maximum flow and minimum-cost flow in almost-linear time," in *2022 IEEE 63rd Annual Symposium on Foundations of Computer Science (FOCS)*, 2022, pp. 612–623. DOI: https://doi.org/10.1109/FOCS54457.2022.00064.