# Introduction to Combinatorial Algorithms

Lucia Moura

Winter 2021

# Introduction to the course

**What are :**

- Combinatorial Structures?
- Combinatorial Algorithms?
- Combinatorial Problems?

# Combinatorial Structures

Combinatorial structures are *collections* of $k$-subsets/$k$-tuple/permutations from a parent set (finite).

- **Undirected Graphs:**
  Collections of $2$-subsets (edges) of a parent set (vertices).
  $$V = \{1, 2, 3, 4\} \quad E = \{\{1, 2\}, \{1, 3\}, \{1, 4\}, \{3, 4\}\}$$
- **Directed Graphs:**
  Collections of $2$-tuples (directed edges) of a parent set (vertices).
  $$V = \{1, 2, 3, 4\} \quad E = \{(2, 1), (3, 1), (1, 4), (3, 4)\}$$
- **Hypergraphs or Set Systems:**
  Similar to graphs, but hyper-edges are sets with possibly more than two elements.
  $$V = \{1, 2, 3, 4\} \quad E = \{\{1, 3\}, \{1, 2, 4\}, \{3, 4\}\}$$

# Building blocks: finite sets, finite lists (tuples)

- **Finite Set**
  $X = \{1, 2, 3, 5\}$
  - undordered structure, no repeats
    $\{1, 2, 3, 5\} = \{2, 1, 5, 3\} = \{2, 1, 1, 5, 3\}$
  - cardinality (size) = number of elements, $|X| = 4$.

  A $k$-subset of a finite set $X$ is a set $S \subseteq X$, $|S| = k$.
  For example: $\{1, 3\}$ is a 2-subset of $X$.

- **Finite List (or Tuple)**
  $L = [1, 5, 2, 1, 3]$
  - ordered structure, repeats allowed
    $[1, 5, 2, 1, 3] \neq [1, 1, 2, 3, 5] \neq [1, 2, 3, 5]$
  - length = number of items, length of $L$ is 5.

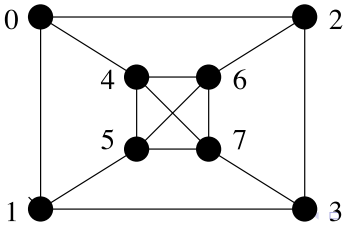
  An $n$-tuple is a list of length $n$.
  A permutation of an $n$-set $X$ is a list of length $n$ such that every element of $X$ occurs exactly once.

# Graphs

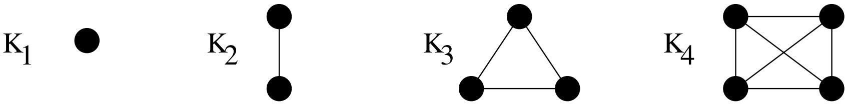
## Definition

*A graph is a pair* $(V, E)$ *where:*
$V$ *is a finite set (of vertices).*
$E$ *is a finite set of* $2$*-subsets (called edges) of* $V$.

$G_1 = (V, E)$
$V = \{0, 1, 2, 3, 4, 5, 6, 7\}$ $\quad E = \{\{0,4\}, \{0,1\}, \{0,2\}, \{2,3\}, \{2,6\}, \{1,3\}, \{1,5\}, \{3,7\}, \{4,5\}, \{4,6\}, \{4,7\}, \{5,6\}, \{5,7\}, \{6,7\}\}$



Complete graphs are graphs with all possible edges.

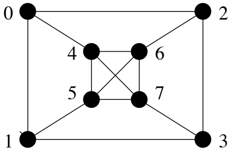


# Substructures of a graph: hamiltonian cycle

## Definition

*A hamiltonian cycle is a closed path that passes through each vertex once.*

The list $[0, 1, 5, 4, 6, 7, 3, 2]$ describes a hamiltonian cycle in the graph:
(Note that different lists may describe the same cycle.)



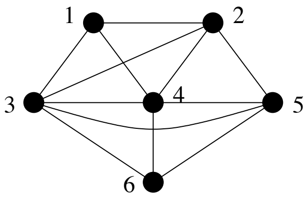

## Problem (Traveling Salesman Problem)

*Given a weight/cost function $w : E \to R$ on the edges of $G$, find a smallest weight hamiltonian cycle in $G$.*

# Substructures of a graph: cliques

## Definition

*A clique in a graph $G = (V, E)$ is a subset $C \subseteq V$ such that $\{x, y\} \in E$, for all $x, y \in C$ with $x \neq y$.*
*(Or equivalently: the subgraph induced by $C$ is complete).*




- Some cliques: $\{1, 2, 3\}$, $\{2, 4, 5\}$, $\{4, 6\}$, $\{1\}$, $\emptyset$
- Maximum cliques (largest): $\{1, 2, 3, 4\}$, $\{3, 4, 5, 6\}$, $\{2, 3, 4, 5\}$

# Famous problems involving cliques

**Problem (Maximum clique problem)**

*Find a clique of maximum cardinality in a graph.*

**Problem (All cliques problem)**

*Find all cliques in a graph without repetition.*

# Set systems/Hypergraphs

## Definition

*A set system (or hypergraph) is a pair* $(X, \mathcal{B})$ *where:*
$X$ *is a finite set (of points/vertices).*
$\mathcal{B}$ *is a finite set of subsets of* $X$ *(blocks/hyperedges).*

# Set systems/Hypergraphs

## Definition

*A set system (or hypergraph) is a pair $(X, \mathcal{B})$ where:*
*$X$ is a finite set (of points/vertices).*
*$\mathcal{B}$ is a finite set of subsets of $X$ (blocks/hyperedges).*

- **Graph:** A graph is a set system with every block with cardinality 2.

# Set systems/Hypergraphs

## Definition

*A set system (or hypergraph) is a pair $(X, \mathcal{B})$ where:*
*$X$ is a finite set (of points/vertices).*
*$\mathcal{B}$ is a finite set of subsets of $X$ (blocks/hyperedges).*

- **Graph:** A graph is a set system with every block with cardinality 2.
- **Partition of a finite set:**
  A partition is a set system $(X, \mathcal{B})$ such that
  $B_1 \cap B_2 = \emptyset$ for all $B_1, B_2 \in \mathcal{B}, B_1 \neq B_2$, and $\cup_{B \in \mathcal{B}} B = X$.

# Set systems/Hypergraphs

## Definition

*A set system (or hypergraph) is a pair $(X, \mathcal{B})$ where:*
*$X$ is a finite set (of points/vertices).*
*$\mathcal{B}$ is a finite set of subsets of $X$ (blocks/hyperedges).*

- **Graph:** A graph is a set system with every block with cardinality 2.
- **Partition of a finite set:**
  A partition is a set system $(X, \mathcal{B})$ such that
  $B_1 \cap B_2 = \emptyset$ for all $B_1, B_2 \in \mathcal{B}, B_1 \neq B_2$, and $\cup_{B \in \mathcal{B}} B = X$.
- **Steiner triple system (a type of combinatorial designs):**
  $\mathcal{B}$ is a set of 3-subsets of $X$ such that for each $x, y \in X, x \neq y$, there exists eactly one block $B \in \mathcal{B}$ with $\{x, y\} \subseteq B$.
  $X = \{0, 1, 2, 3, 4, 5, 6\}$
  $\mathcal{B} = \{\{0, 1, 2\}, \{0, 3, 4\}, \{0, 5, 6\}, \{1, 3, 5\}, \{1, 4, 6\}, \{2, 3, 6\}, \{2, 4, 5\}\}$

# Combinatorial algorithms

Combinatorial algorithms are algorithms for investigating combinatorial structures.

- **Generation**
  **Construct all** combinatorial structures of a particular type.
- **Enumeration**
  **Compute the number** of all different structures of a particular type.
- **Search**
  **Find at least one** example of a combinatorial structures of a particular type (if one exists).
  **Optimization problems** can be seen as a type of search problem.

- **Generation**
  **Construct all** combinatorial structures of a particular type.
  - Generate all subsets/permutations/partitions of a set.
  - Generate all cliques of a graph.
  - Generate all maximum cliques of a graph.
  - Generate all Steiner triple systems of a finite set.

- **Generation**
  **Construct all** combinatorial structures of a particular type.
  - Generate all subsets/permutations/partitions of a set.
  - Generate all cliques of a graph.
  - Generate all maximum cliques of a graph.
  - Generate all Steiner triple systems of a finite set.

- **Enumeration**
  **Compute the number** of all different structures of a particular type.
  - Compute the number of subsets/permutat./partitions of a set.
  - Compute the number of cliques of a graph.
  - Compute the number of maximum cliques of a graph.
  - Compute the number of Steiner triple systems of a finite set.

- **Search**
  **Find at least one** example of a combinatorial structures of a particular type (if one exists).
  **Optimization problems** can be seen as a type of search problem.
  - Find a Steiner triple system on a finite set. (feasibility)
  - Find a maximum clique of a graph. (optimization)
  - Find a hamiltonian cycle in a graph. (feasibility)
  - Find a smallest weight hamiltonian cycle in a graph. (optimization)

# Hardness of Search and Optimization

- Many search and optimization problems are **NP-hard** or their corresponding "decision problems" are **NP-complete**.

# Hardness of Search and Optimization

- Many search and optimization problems are **NP-hard** or their corresponding "decision problems" are **NP-complete**.
- **P** = class of decision problems that can be **solved** in polynomial time. (e.g. Shortest path in a graph is in **P**)
  **NP** = class of decision problems that can be **verified** in polynomial time. (e.g. Hamiltonian path in a graph is in **NP**)
  Therefore, **P** $\subseteq$ **NP**.
  **NP-complete** are problems in **NP** that are at least "as hard as" any other problem in **NP**.

# Hardness of Search and Optimization

- Many search and optimization problems are **NP-hard** or their corresponding "decision problems" are **NP-complete**.
- **P** = class of decision problems that can be **solved** in polynomial time. (e.g. Shortest path in a graph is in **P**)
  **NP** = class of decision problems that can be **verified** in polynomial time. (e.g. Hamiltonian path in a graph is in **NP**)
  Therefore, **P** $\subseteq$ **NP**.
  **NP-complete** are problems in **NP** that are at least "as hard as" any other problem in **NP**.
- An important unsolved complexity question is the **P=NP** question. One million dollars offered for its solution!

# Hardness of Search and Optimization

- Many search and optimization problems are **NP-hard** or their corresponding "decision problems" are **NP-complete**.
- **P** = class of decision problems that can be **solved** in polynomial time. (e.g. Shortest path in a graph is in **P**)
  **NP** = class of decision problems that can be **verified** in polynomial time. (e.g. Hamiltonian path in a graph is in **NP**)
  Therefore, **P** $\subseteq$ **NP**.
  **NP-complete** are problems in **NP** that are at least "as hard as" any other problem in **NP**.
- An important unsolved complexity question is the **P=NP** question. One million dollars offered for its solution!
- It is believed that **P**$\neq$**NP** which, if true, would mean that there exist no polynomial-time algorithm to solve an **NP-hard** problem.

# Hardness of Search and Optimization

- Many search and optimization problems are **NP-hard** or their corresponding "decision problems" are **NP-complete**.
- **P** = class of decision problems that can be **solved** in polynomial time. (e.g. Shortest path in a graph is in **P**)
  **NP** = class of decision problems that can be **verified** in polynomial time. (e.g. Hamiltonian path in a graph is in **NP**)
  Therefore, **P** $\subseteq$ **NP**.
  **NP-complete** are problems in **NP** that are at least "as hard as" any other problem in **NP**.
- An important unsolved complexity question is the **P=NP** question. One million dollars offered for its solution!
- It is believed that **P**$\neq$**NP** which, if true, would mean that there exist no polynomial-time algorithm to solve an **NP-hard** problem.
- There are several approaches to deal with **NP-hard** problems.

# Approaches for dealing with NP-hard problems

- **Exhaustive Search**
  - exponential-time algorithms.
  - solves the problem exactly

  (Backtracking and Branch-and-Bound)
- **Heuristic Search/Metaheuristics**
  - algorithms that explore a search space to find a feasible solution that is hopefully "close to" optimal, within a time limit
  - approximates a solution to the problem

  (Hill-climbing, Simulated annealing, Tabu-Search, Genetic Algs, etc.)
- **Approximation Algorithms**
  - polynomial time algorithm
  - we have a provable guarantee that the solution found is "close to" optimal.

  (Approximation algorithms not covered in this course)

# Types of Search Problems

1) **Decision Problem**:
A yes/no problem
**Problem 1:** Clique (decision)
Instance: graph $G = (V, E)$,
target size $k$
**Question:**
Does there exist a clique $C$
of $G$ with $|C| = k$?

2) **Search Problem**
Find the guy.
**Problem 2:** Clique (search)
Instance: graph $G = (V, E)$,
target size $k$
**Find:**
a clique $C$ of $G$
with $|C| = k$, if one exists.

3) **Optimal Value**:
Find the largest target size.
**Problem 3:** Clique (optimal value)
Instance: graph $G = (V, E)$,
**Find:**
the maximum value of $|C|$,
where $C$ is a clique

4) **Optimization**:
Find an optimal guy.
**Problem 4:** Clique (optimization)
Instance: graph $G = (V, E)$,
**Find:**
a clique $C$ such that
$|C|$ is maximize (max. clique)

# Topics for the Course

"text" refers to: Kreher&Stinson, *Combinatorial Algorithms: generation, enumeration& search*

1. **Generating elementary combinatorial objects** [text Ch 2,3&Knuth's TAOCP 4A]
   Sequential generation (successor), rank, unrank.
   Algorithms for subsets, $k$-subsets, permutations, other objects.
2. **Exhaustive Generation and Exhaustive Search**
   [text Chap4+ Kaski&Ostergard's book Chap 4.1]]
   Backtracking algorithms (exhaustive generation, exhaustive search, optimiz.)
   Branch-and-bound (exhaustive search, optimization)
3. **Heuristic Search** [text Chap 5 + Gendreau&Potvin's Handbook of Metaheuristics Vol.2]
   Hill-climbing, Simulated annealing, Tabu-Search, Genetic Algs, etc.
4. **Computing Isomorphism and Isomorph-free Exhaustive Generation** [text Chap 7 + Kaski&Ostergard's book Chap 3,4]
   Graph isomorphism, isomorphism of other structures.
   Computing invariants. Computing certificates.
   Isomorph-free exhaustive generation.

# Course evaluation

- **45% Assignments**
  3 assignments, 15% each
  Assignments covering: theory, algorithms, implementation
- **11% Student presentation (topic can be related to project)**
  10% 15-20 minute talk
  1% participation (attending talks/discussion/other method)
  Talk teaching your peers on a topic not covered in lectures.
- **44% Project: individual, chosen by student**
  3% Project proposal (1-2 pages)
  11% Project presentation (15-20 minute talk) 10%, particip. 1%
  30% Project paper (10-15 page)
  - research (reading papers related to course topics),
  - original work (involving one or more of: modelling, application, algorithm design, implementation, experimentation, analysis)