Graphs: Introduction

Ali Shokoufandeh,

Department of Computer Science, Drexel University

Overview of this talk

Introduction:

Notations and Definitions

Graphs and Modeling

Algorithmic Graph Theory and Combinatorial Optimization

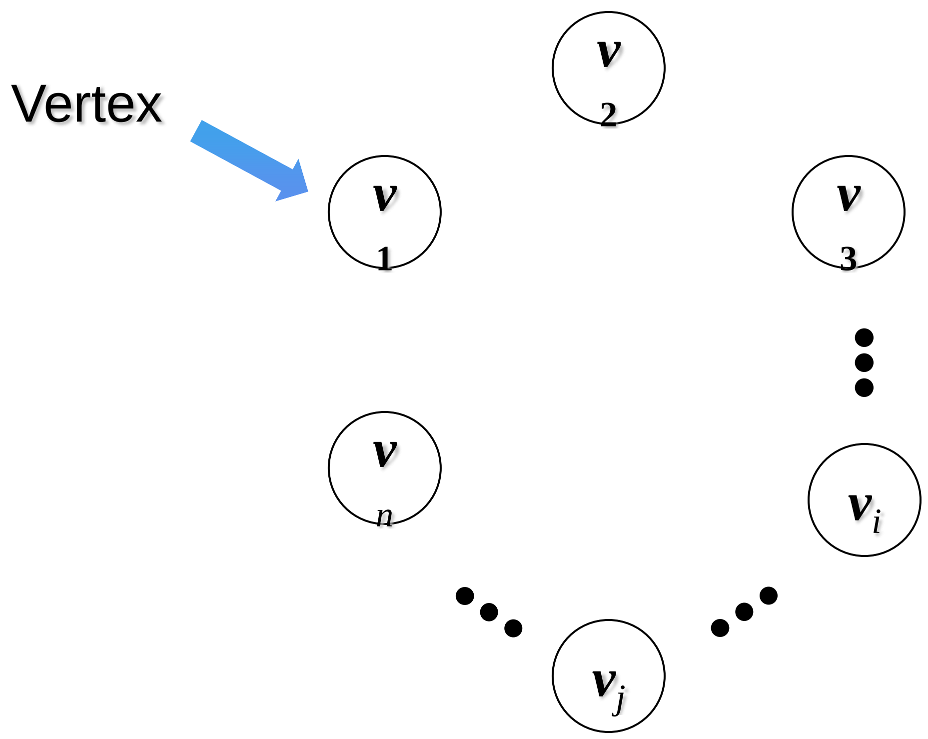
Overview of this talk

Introduction:

- **Notations and Definitions**
- Graphs and Modeling
- Algorithmic Graph Theory and Combinatorial Optimization

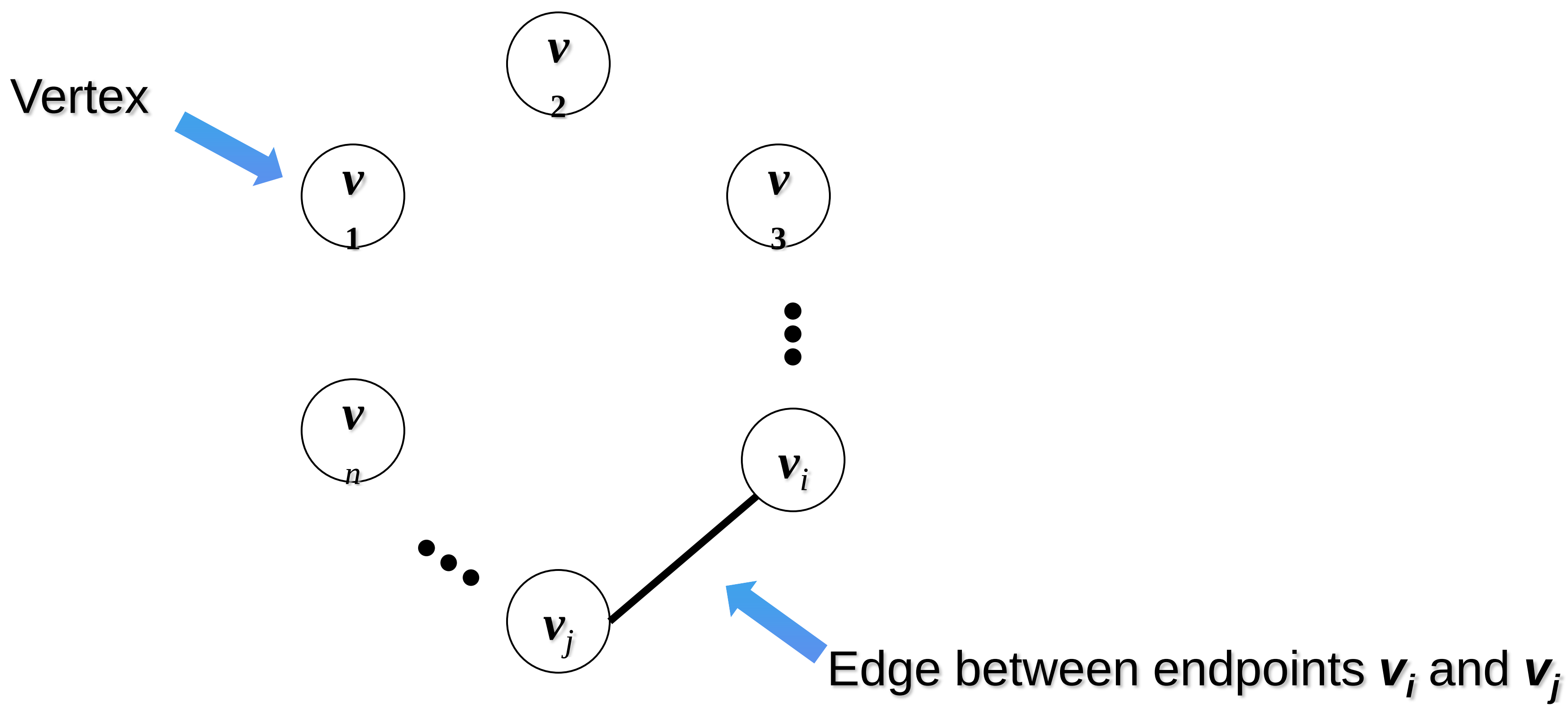
Notations:

A graph G is a triple consisting of a ***vertex set*** $V(G)$, an ***edge set*** $E(G)$, and a ***relation (weight function)*** $W(G)$ associates with each edge. The two vertices of each edge will be called its ***endpoints*** (not necessarily distinct).



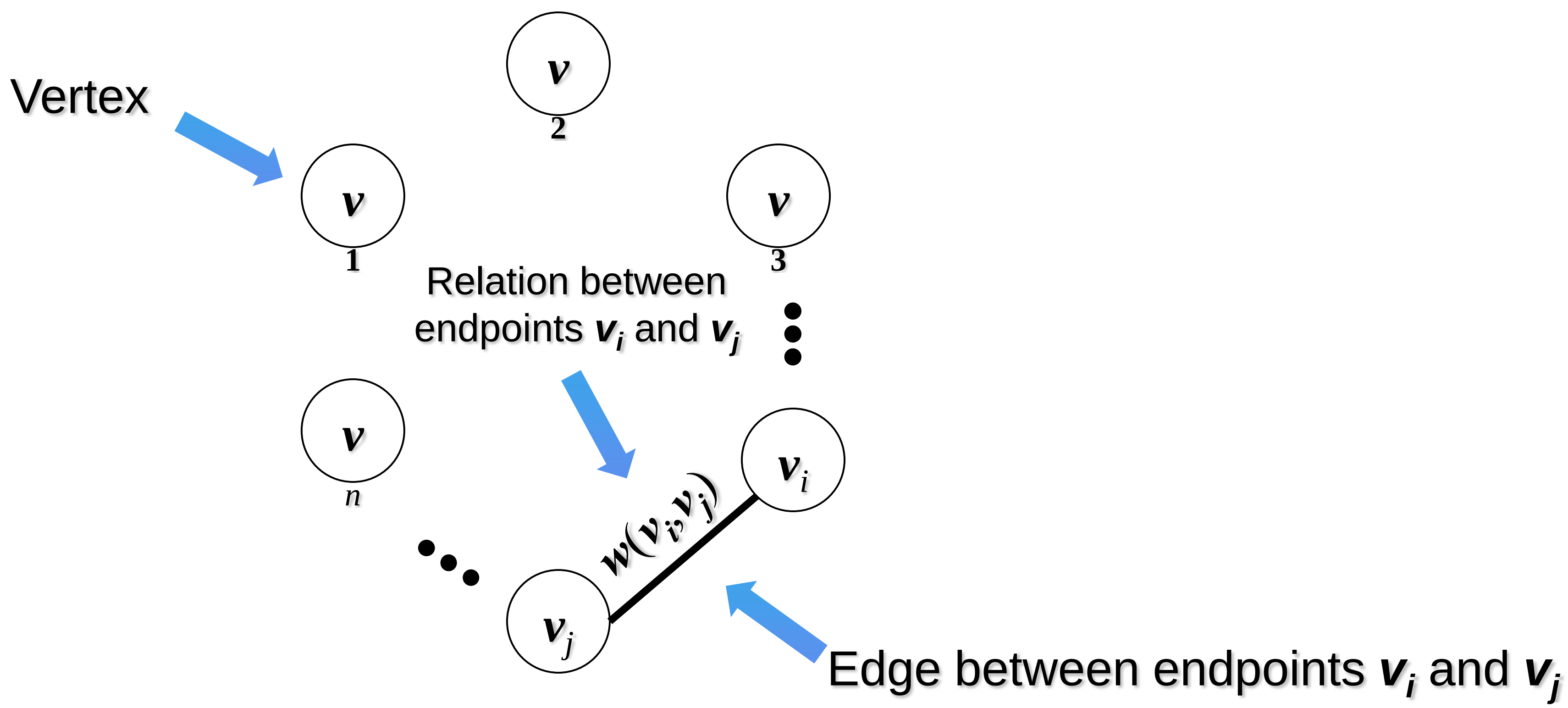
Notations:

A graph ***G*** is a triple consisting of a ***vertex set V***(***G***), an ***edge set E***(***G***), and a ***relation (weight function) W***(***G***) associates with each edge. The two vertices of each edge will be called its ***endpoints*** (not necessarily distinct).



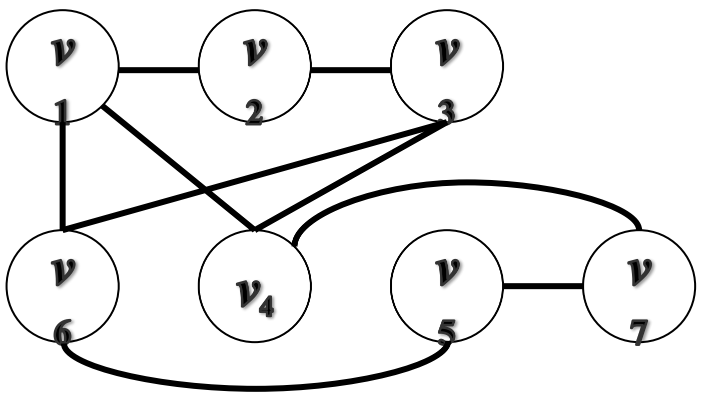
Notations:

A graph $\boldsymbol{G}$ is a triple consisting of a ***vertex set*** $\boldsymbol{V}(\boldsymbol{G})$, an ***edge set*** $\boldsymbol{E}(\boldsymbol{G})$, and a ***relation (weight function)*** $\boldsymbol{W}(\boldsymbol{G})$ associates with each edge. The two vertices of each edge will be called its ***endpoints*** (not necessarily distinct).



Notations:

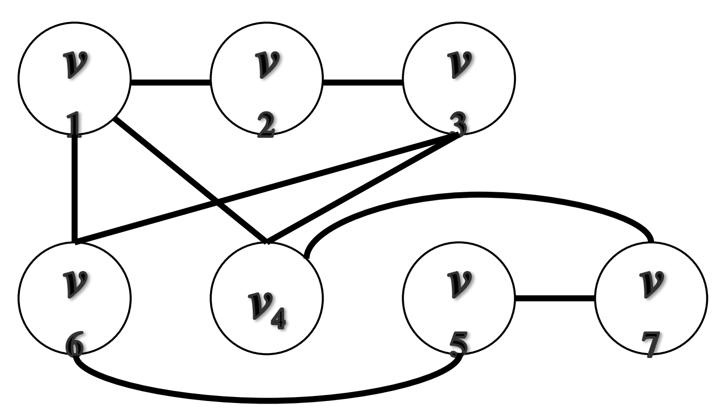
- We will only consider ***finite graphs***, i.e. graphs for which $\boldsymbol{V}(\boldsymbol{G})$ and $\boldsymbol{E}(\boldsymbol{G})$ are finite sets.



Notations:

- We will only consider ***finite graphs***, i.e. graphs for which $\boldsymbol{V}(\boldsymbol{G})$ and $\boldsymbol{E}(\boldsymbol{G})$ are finite sets.

- A ***simple graph*** is a graph having no ***loops*** or ***multiple*** edges, i.e., each edge $\boldsymbol{e=(u,v)}$ in $\boldsymbol{E}(\boldsymbol{G})$ can be specified by its endpoints $\boldsymbol{u}$ and $\boldsymbol{v}$ in $\boldsymbol{V}(\boldsymbol{G})$.

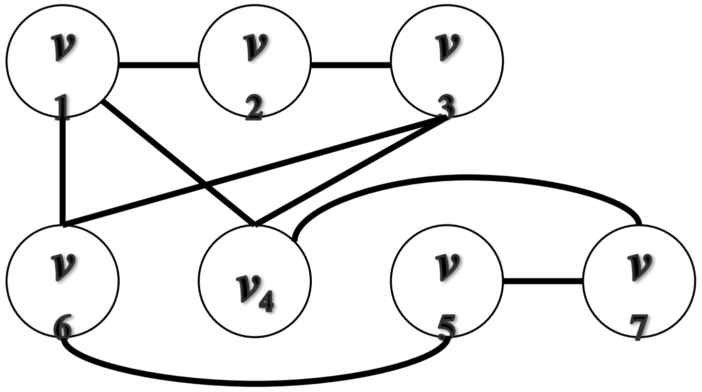


Notations:

- We will only consider ***finite graphs***, i.e. graphs for which ***V***(***G***) and ***E***(***G***) are finite sets.

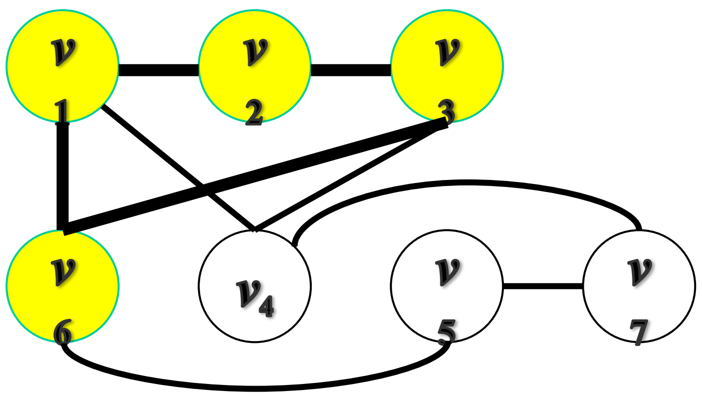
- A ***simple graph*** is a graph having no ***loops*** or ***multiple*** edges, i.e., each edge ***e*=(*u*,*v*)** in ***E***(***G***) can be specified by its endpoints ***u*** and ***v*** in ***V***(***G***).

- **Induced subgraphs**: a subgraph formed by a subset of vertices and edges of a graph.



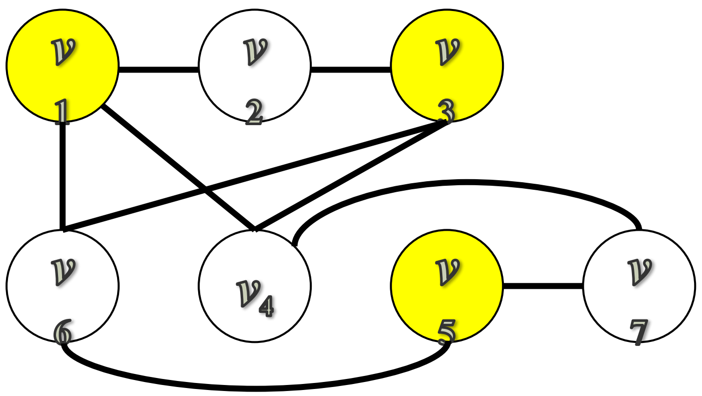
Notations:

- We will only consider ***finite graphs***, i.e. graphs for which ***V***(***G***) and ***E***(***G***) are finite sets.
- A ***simple graph*** is a graph having no ***loops*** or ***multiple*** edges, i.e., each edge ***e*=(*u*,*v*)** in ***E***(***G***) can be specified by its endpoints ***u*** and ***v*** in ***V***(***G***).
- **Induced subgraphs:** a subgraph formed by a subset of vertices and edges of a graph.



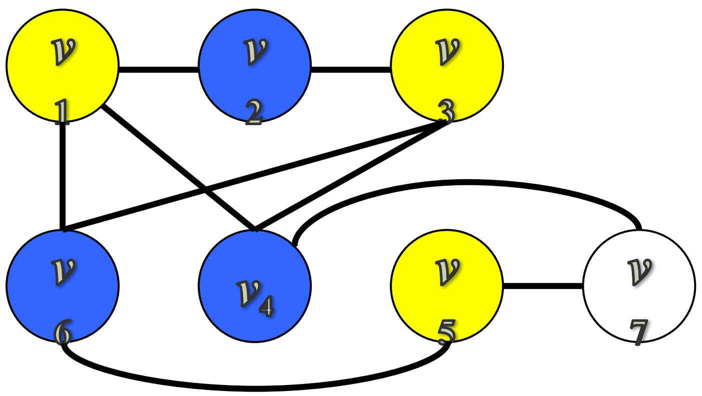
Notations:

- We will only consider ***finite graphs***, i.e. graphs for which $\boldsymbol{V}(\boldsymbol{G})$ and $\boldsymbol{E}(\boldsymbol{G})$ are finite sets.
- A ***simple graph*** is a graph having no ***loops*** or ***multiple*** edges, i.e., each edge $\boldsymbol{e=(u,v)}$ in $\boldsymbol{E}(\boldsymbol{G})$ can be specified by its endpoints $\boldsymbol{u}$ and $\boldsymbol{v}$ in $\boldsymbol{V}(\boldsymbol{G})$.
- Induced subgraphs:
  - Example: an ***independent set*** in a graph is a set of vertices that are pairwise nonadjacent



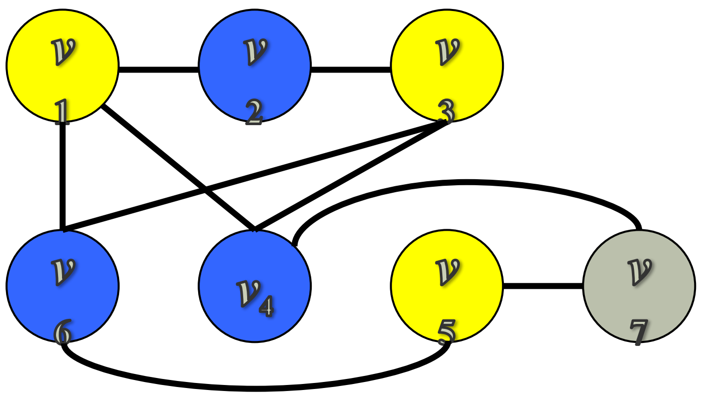
Notations:

- We will only consider ***finite graphs***, i.e. graphs for which $\boldsymbol{V}(\boldsymbol{G})$ and $\boldsymbol{E}(\boldsymbol{G})$ are finite sets.
- A ***simple graph*** is a graph having no ***loops*** or ***multiple*** edges, i.e., each edge $\boldsymbol{e=(u,v)}$ in $\boldsymbol{E}(\boldsymbol{G})$ can be specified by its endpoints $\boldsymbol{u}$ and $\boldsymbol{v}$ in $\boldsymbol{V}(\boldsymbol{G})$.
- Induced subgraphs:
  - Example: an ***independent set*** in a graph is a set of vertices that are pairwise nonadjacent



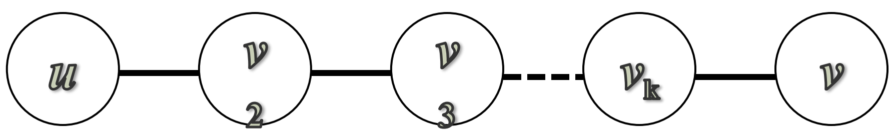
Notations:

- We will only consider ***finite graphs***, i.e. graphs for which ***V***(***G***) and ***E***(***G***) are finite sets.
- A ***simple graph*** is a graph having no ***loops*** or ***multiple*** edges, i.e., each edge ***e*=(*u*,*v*)** in ***E***(***G***) can be specified by its endpoints ***u*** and ***v*** in ***V***(***G***).
- Induced subgraphs:
  - Example: an ***independent set*** in a graph is a set of vertices that are pairwise nonadjacent

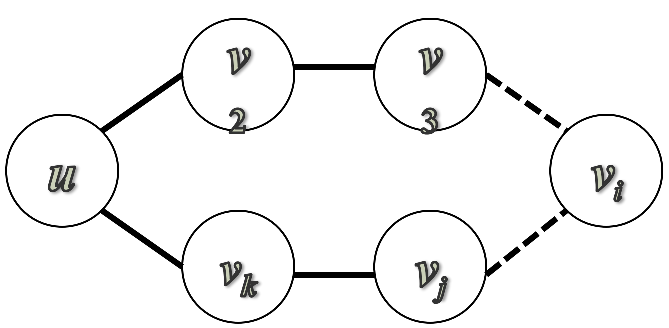


Notations:

- A ***<u,v>-path*** is a simple graph that begins at ***u*** and ends at ***v***, whose vertices can be ordered so that two vertices are adjacent if and only if they are consecutive in the ordering.

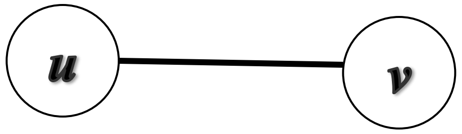


- A ***cycle*** is a simple path whose vertices can be cyclically ordered with overlapping endpoints (***u***=***v***).



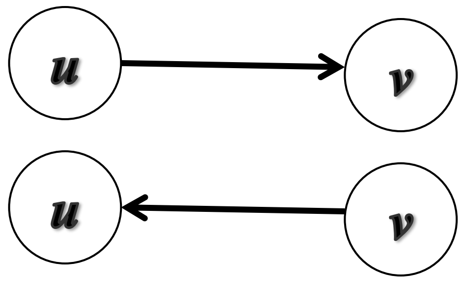
Notations:

- Given a graph $\boldsymbol{G}=(\boldsymbol{V},\boldsymbol{E})$, where
  - $\boldsymbol{V}$ is its vertex set, $|\boldsymbol{V}|=\boldsymbol{n}$,
  - $\boldsymbol{E}$ is its edge set, with $|\boldsymbol{E}|=\boldsymbol{m}=O(\boldsymbol{n}^2)$.
- In an undirected graph an edge $(\boldsymbol{u},\boldsymbol{v})=(\boldsymbol{v},\boldsymbol{u})$.



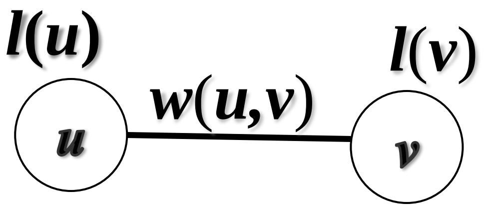
Notations:

- Given a graph $G=(V,E)$, where
  - V is its vertex set, $|V|=n$,
  - E is its edge set, with $|E|=m=O(n^2)$.
- In an undirected graph an edge $(u,v)=(v,u)$.
- In directed graph (u,v) is different from (v,u).



Notations:

- Given a graph $\boldsymbol{G}=(\boldsymbol{V},\boldsymbol{E})$, where
  - $\boldsymbol{V}$ is its vertex set, $|\boldsymbol{V}|=\boldsymbol{n}$,
  - $\boldsymbol{E}$ is its edge set, with $|\boldsymbol{E}|=\boldsymbol{m}=O(\boldsymbol{n}^2)$.
- In an undirected graph an edge $(\boldsymbol{u},\boldsymbol{v})=(\boldsymbol{v},\boldsymbol{u})$.
- In directed graph $(\boldsymbol{u},\boldsymbol{v})$ is different from $(\boldsymbol{v},\boldsymbol{u})$.
- In a weighted graph the labels are the weights associated with edges and/or vertices.

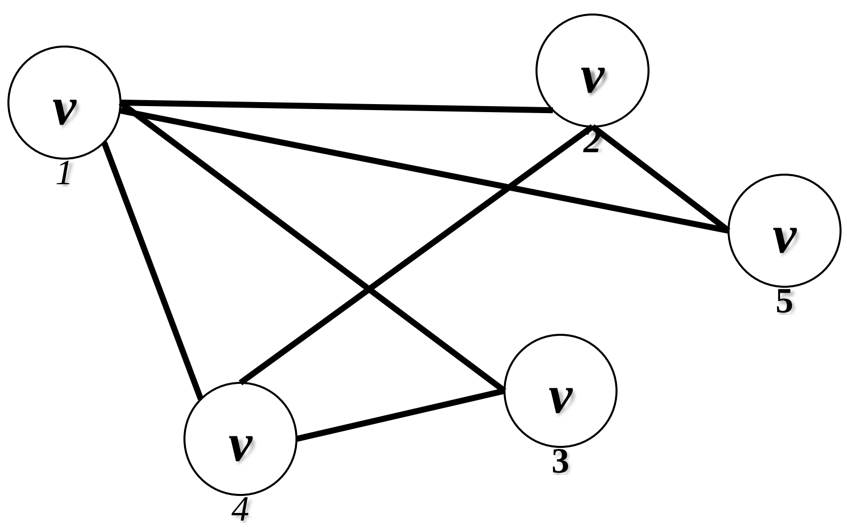


Notations:

- Given a graph $\boldsymbol{G}=(\boldsymbol{V},\boldsymbol{E})$, where
  - $\boldsymbol{V}$ is its vertex set, $|\boldsymbol{V}|=\boldsymbol{n}$,
  - $\boldsymbol{E}$ is its edge set, with $|\boldsymbol{E}|=\boldsymbol{m}=\mathrm{O}(\boldsymbol{n}^2)$.
- In an undirected graph an edge $(\boldsymbol{u},\boldsymbol{v})=(\boldsymbol{v},\boldsymbol{u})$.
- In directed graph $(\boldsymbol{u},\boldsymbol{v})$ is different from $(\boldsymbol{v},\boldsymbol{u})$.
- In a weighted graph the labels are the weights associated with edges and/or vertices.
- Running time of graph algorithms are usually expressed in terms of $\boldsymbol{n}$ or $\boldsymbol{m}$.

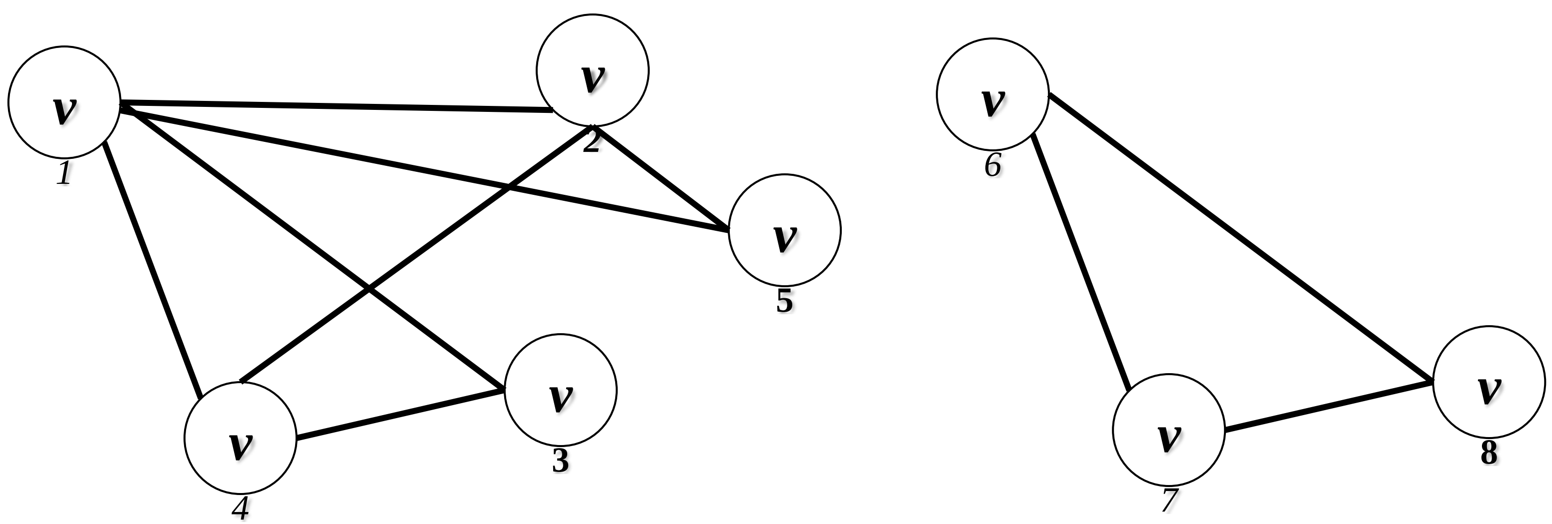
Notations:

- A graph *G* is ***connected*** if for every ***u***,***v*** in ***V***(***G***) there exists a simple <***u,v***>-path in ***G*** (otherwise ***G*** is ***disconnected***).



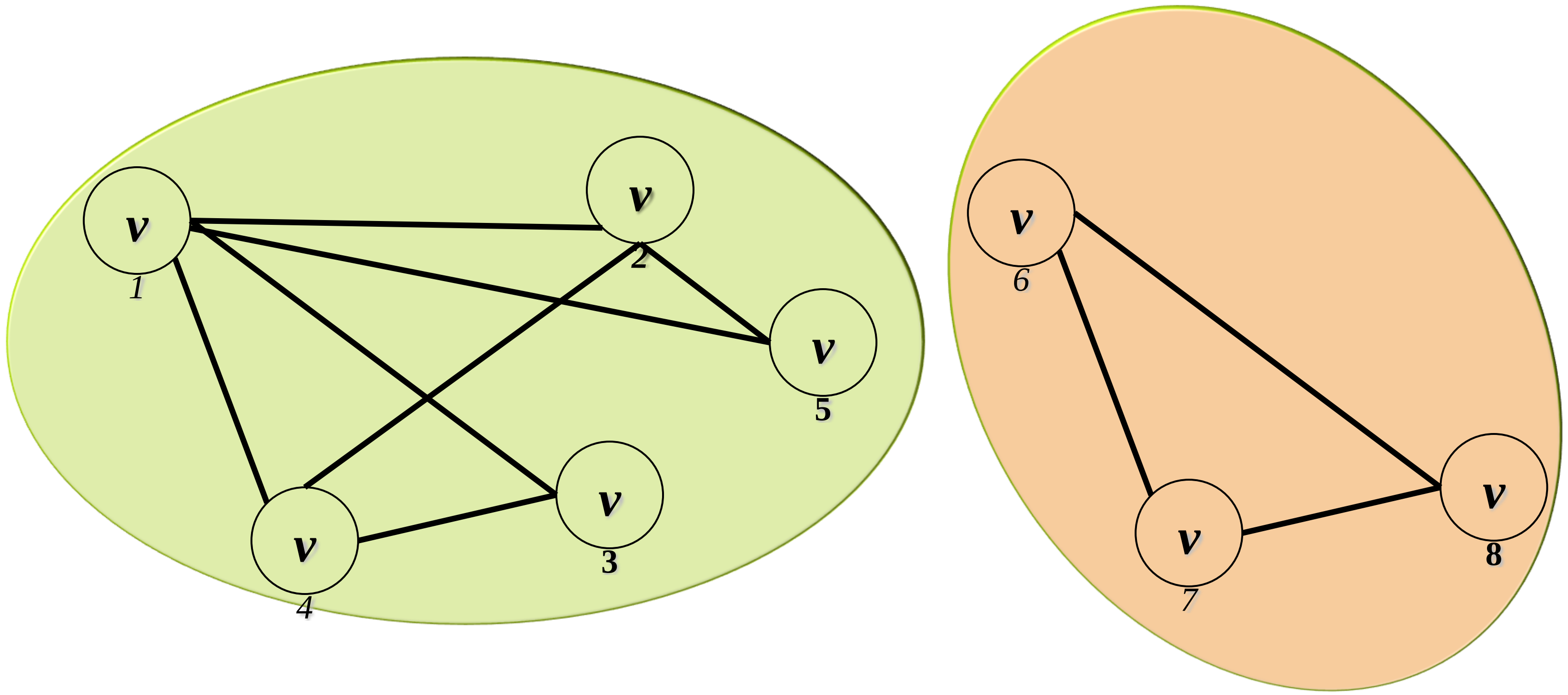
Notations:

- A graph *G* is ***connected*** if for every ***u***,***v*** in ***V***(***G***) there exists a simple <***u,v***>-path in ***G*** (otherwise ***G*** is ***disconnected***).



Notations:

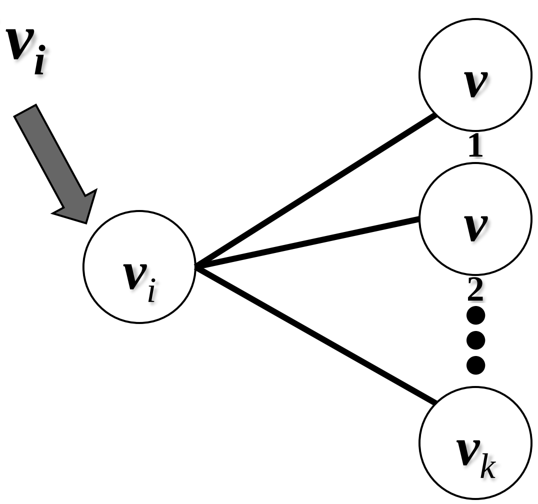
- A graph *G* is ***connected*** if for every ***u*,*v*** in ***V*(*G*)** there exists a simple <***u*,*v***>-path in ***G*** (otherwise ***G*** is ***disconnected***).
- The ***maximal*** connected subgraphs of *G* are called its ***connected components***.



Notations:

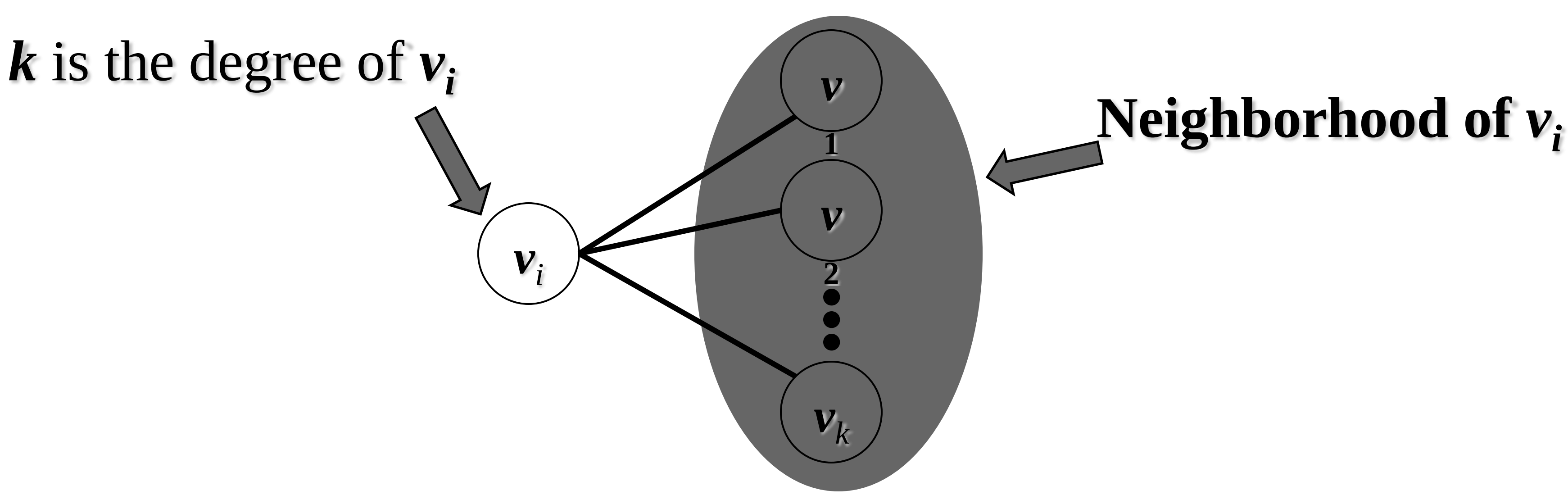
- A graph *G* is ***connected*** if for every ***u***,***v*** in ***V***(***G***) there exists a simple <***u,v***>-path in ***G*** (otherwise ***G*** is ***disconnected***).
- The ***maximal*** connected subgraphs of *G* are called its ***connected components***.
- The ***degree*** of a vertex ***v*** in a graph ***G***, denoted **deg**(***v***), is the number of edges in ***G*** which have ***v*** as an endpoint.

k is the degree of $\boldsymbol{v_i}$



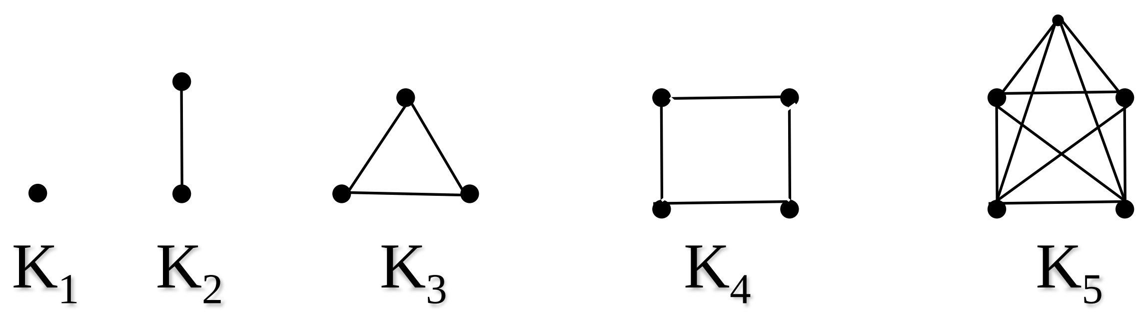
Notations:

- A graph *G* is ***connected*** if for every ***u*,*v*** in ***V*(*G*)** there exists a <*u*,*v*>-path in *G* (otherwise *G* is ***disconnected***).
- The ***maximal*** connected subgraphs of *G* are called its ***connected components***.
- The ***degree*** of a vertex ***v*** in a graph ***G***, denoted **deg(*v*)**, is the number of edges in ***G*** which have ***v*** as an endpoint.

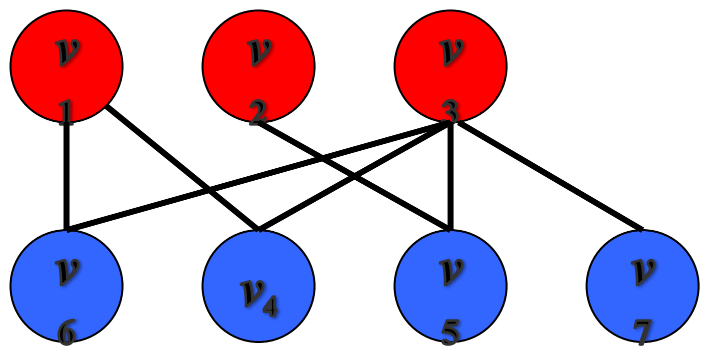


Important Graphs:

- A ***complete graph*** is a simple graph whose vertices are pairwise adjacent. The complete graph with ***n*** vertices is denoted $\mathbf{K}_n$.

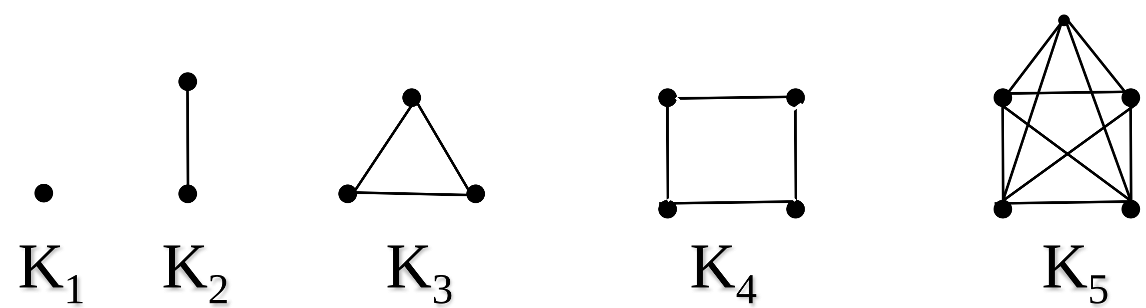


- A graph ***G*** is ***bipartite*** if ***V***(***G***) is the union of two disjoint (possibly empty) independent sets, called partite sets of ***G***.



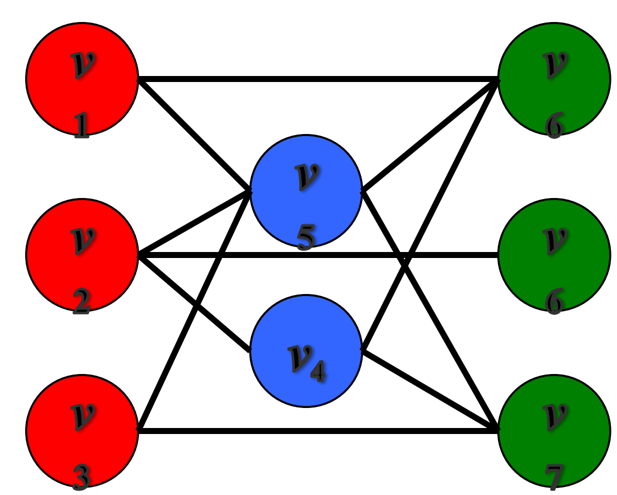
Important Graphs:

- A ***complete graph*** is a simple graph whose vertices are pairwise adjacent. The complete graph with $\boldsymbol{n}$ vertices is denoted $\mathbf{K}_{\boldsymbol{n}}$.



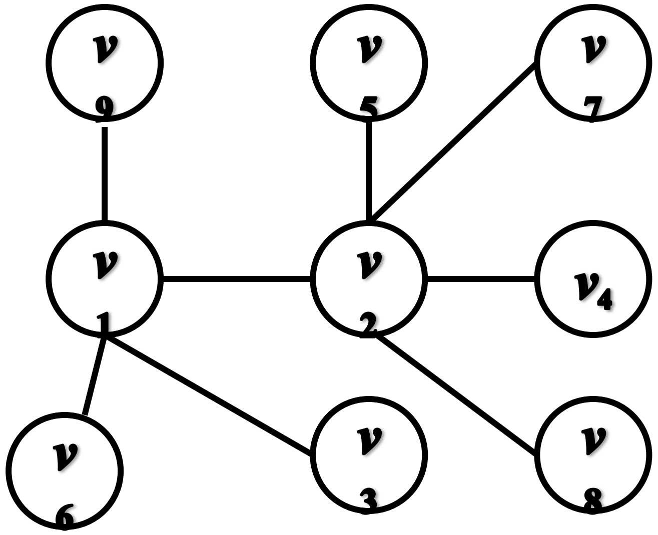
- A graph $\boldsymbol{G}$ is bipartite if $\boldsymbol{V}(\boldsymbol{G})$ is the union of two disjoint (possibly empty) independent sets, called partite sets of $\boldsymbol{G}$.

- A graph is $\boldsymbol{k}$-partite if $\boldsymbol{V}(\boldsymbol{G})$ is the union of $\boldsymbol{k}$ disjoint independent sets.



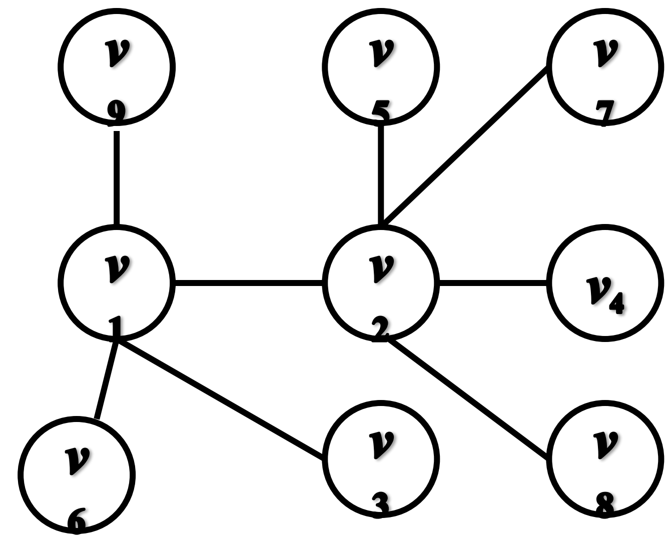
Important Graphs:

- A Tree is a connected acyclic graph.

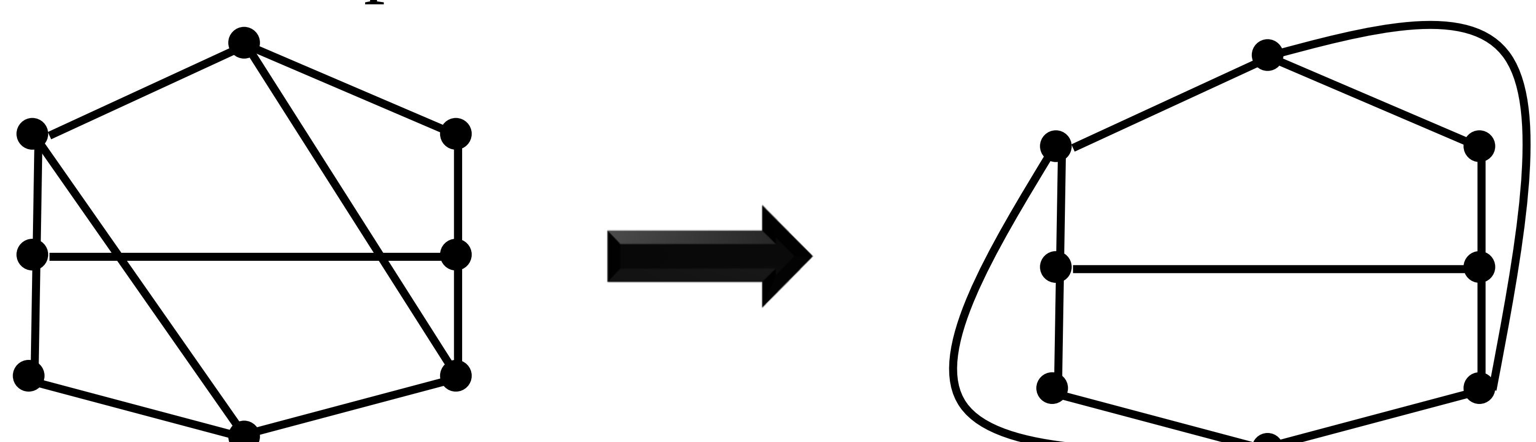


Important Graphs:

- A Tree is a connected acyclic graph.



- A graph is ***planar*** if it can be drawn in the plane without crossings.
- A graph that is so drawn in the plane is also said to be embedded in the plane.

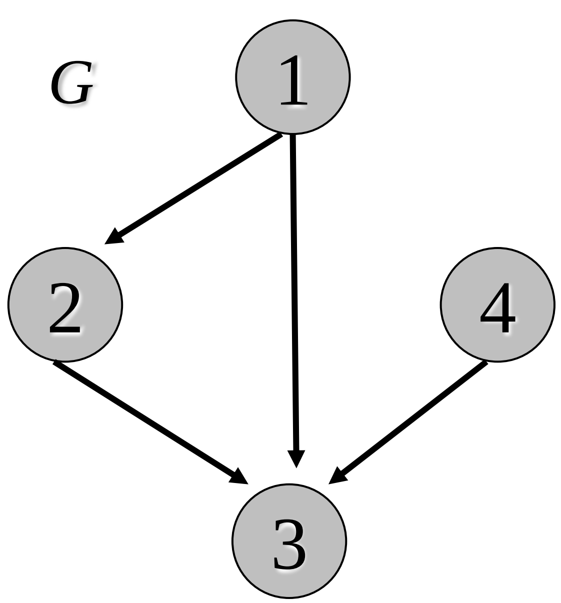


Graph Representation:

- The adjacency matrix of a graph G, denoted by A_G is an $n \times n$ defined as follows:

$$A_G[i,j] = \begin{cases} 1 & \text{if } (i,j) \in E \\ 0 & \text{if } (i,j) \notin E \end{cases}$$

- If G is directed then A_G is asymmetric.



$$A_G = \begin{bmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

Notes:

- Number of 1's in A_G is m if G is directed; if its undirected, then number of 1's is $2m$.
- Degree of a vertex is the sum of entries in corresponding row of A_G
- If G is undirected then sum of all degree is $2m$.
- In a directed graph sum of the out degrees is equal to m.

Overview of this talk

Introduction:

Notations and Definitions

Graphs and Modeling

Algorithmic Graph Theory and Combinatorial Optimization

Graphs and Representation:

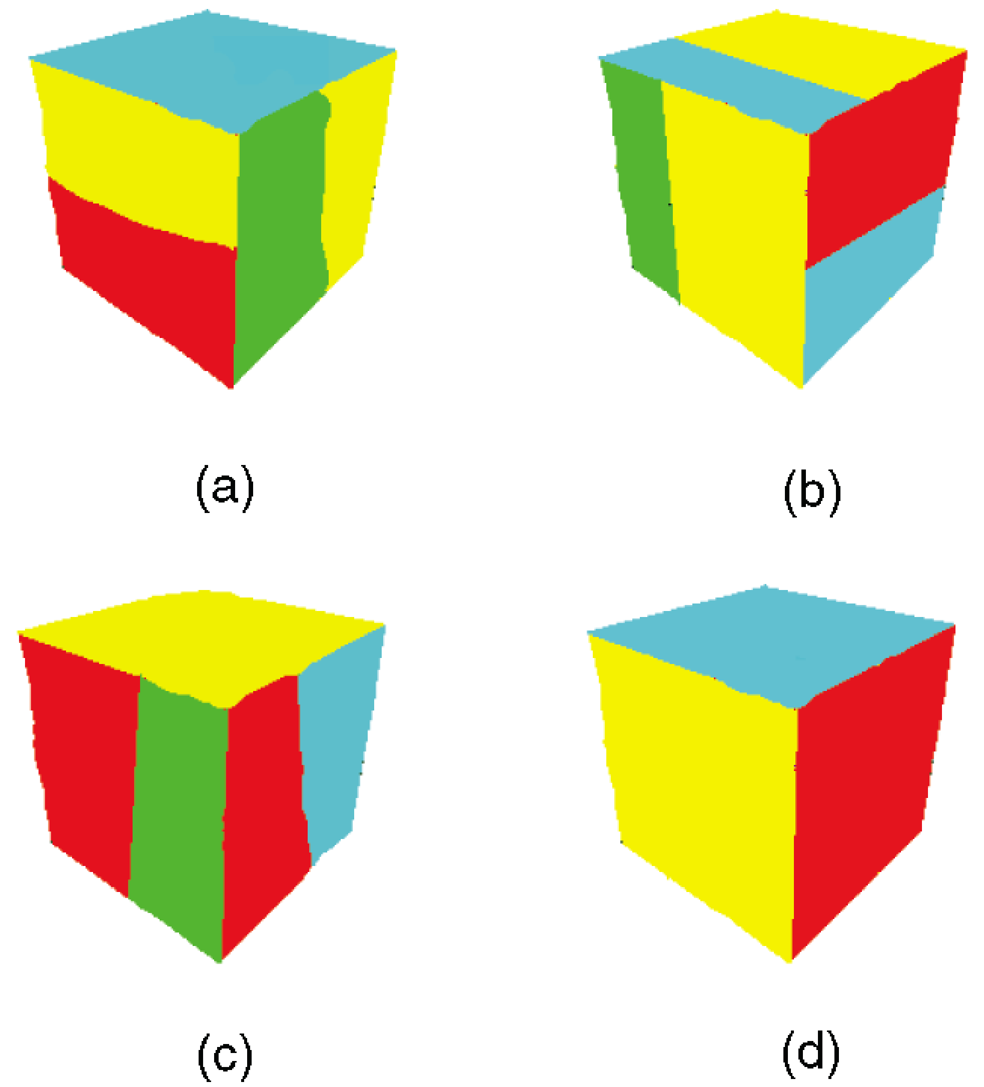
- **Objective**: To capture the ***essential*** structure an ***entity/object*** using a graph-based representation.

Graphs and Representation:

- **Objective**: To capture the ***essential*** structure an ***entity/object*** using a graph-based representation.
- Mechanics:
  - The vertex set ***V***(***G***) represents feature primitives extracted from object, encoded as a label/attribute function ***l***(***v***) for each ***v*** in ***V***(***G***).
  - The edge set ***E***(***G***) capture the ***affinity*** or ***relative*** distribution of features within object. The edge weight ***w***(***e***) for each ***e*** in ***E***(***G***) captures the attributes of the edge.

Graphs and Representation:

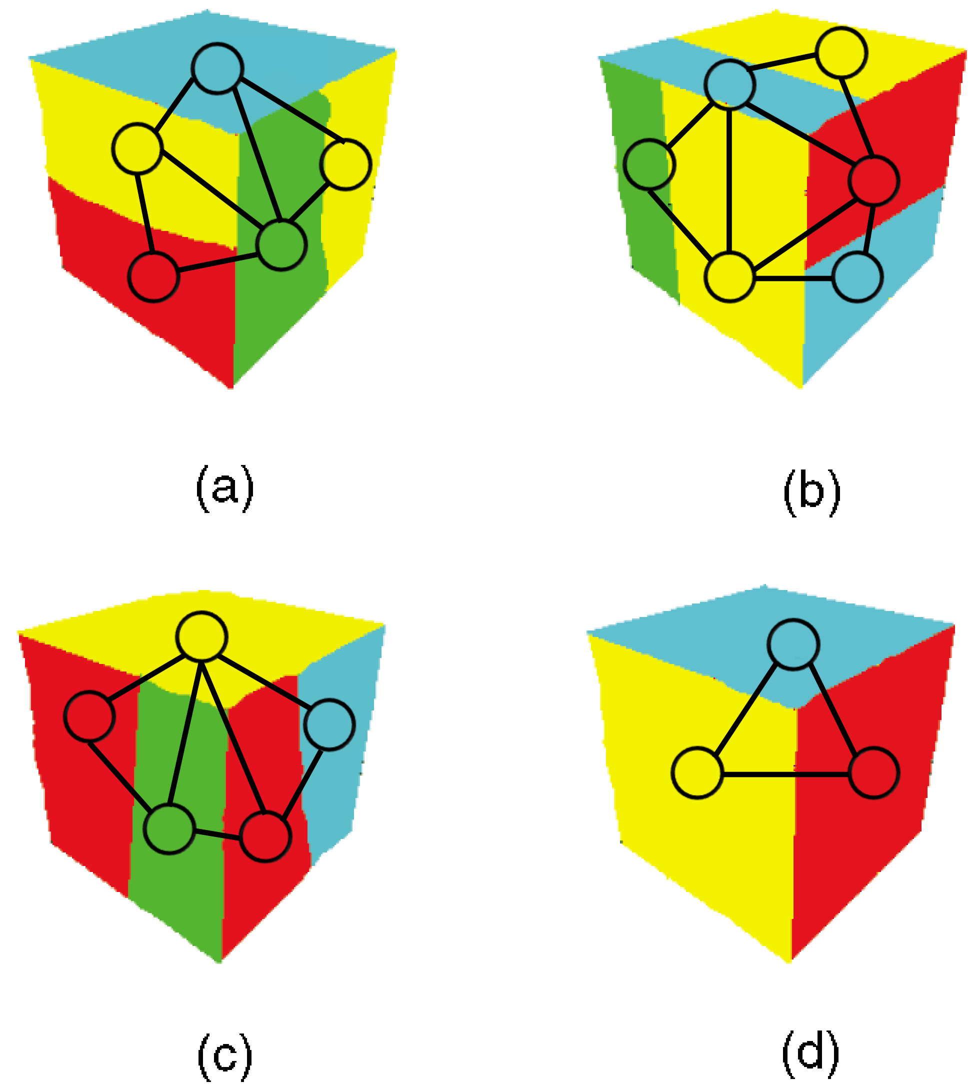
- **Objective**: To capture the ***essential*** structure an ***entity/object*** using a graph-based representation.
- Mechanics:
  - The vertex set $\boldsymbol{V(G)}$ represents feature primitives extracted from object, encoded as a label/attribute function $\boldsymbol{l(v)}$ for each $\boldsymbol{v}$ in $\boldsymbol{V(G)}$.
  - The edge set $\boldsymbol{E(G)}$ capture the ***affinity*** or ***relative*** distribution of features within object. The edge weight $\boldsymbol{w(e)}$ for each $\boldsymbol{e}$ in $\boldsymbol{E(G)}$ captures the attributes of the edge.



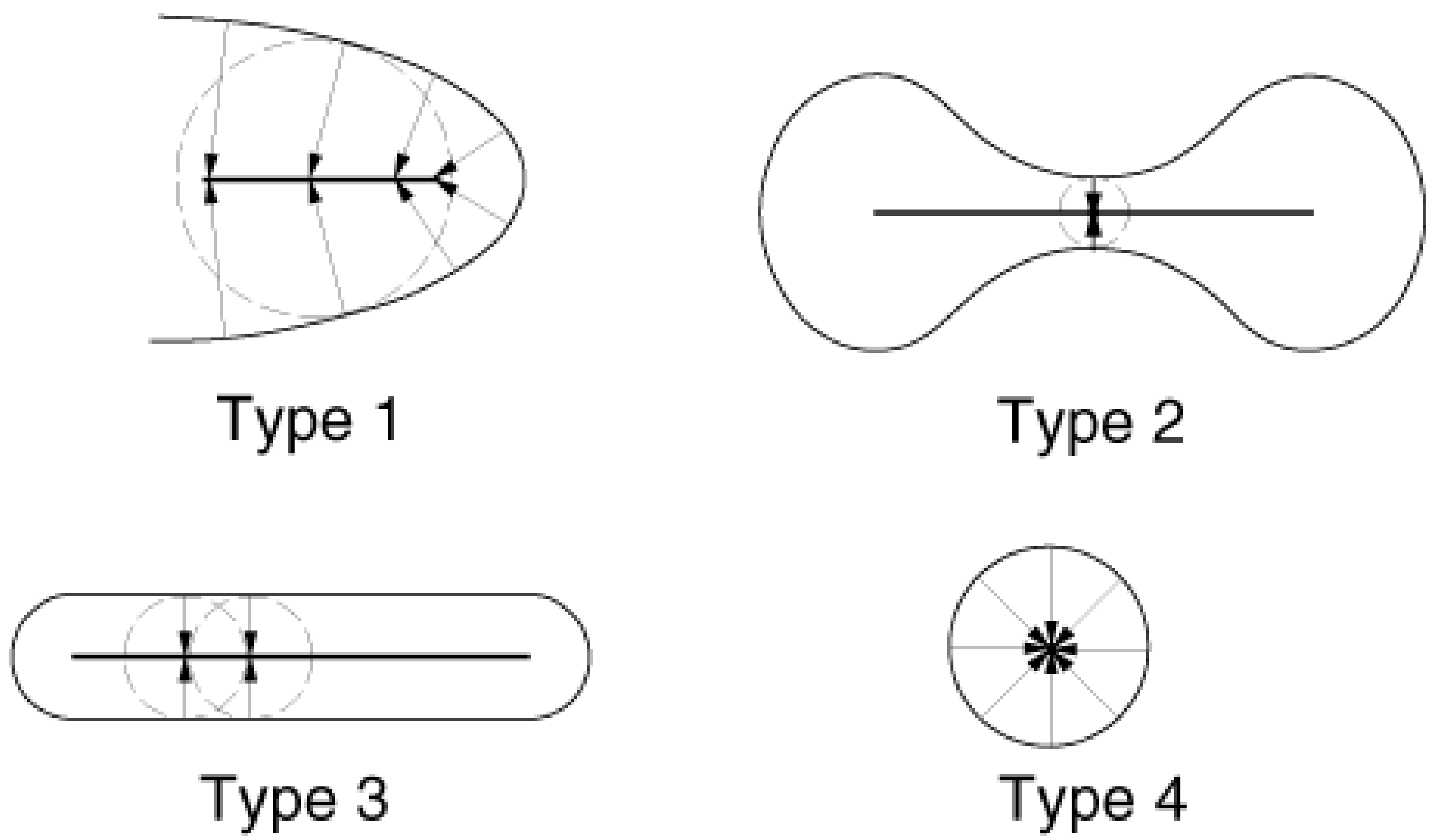
Graphs and Representation:

- **Objective**: To capture the ***essential*** structure an ***entity/object*** using a graph-based representation.

- Mechanics:
  - The vertex set ***V***(***G***) represents feature primitives extracted from object, encoded as a label/attribute function ***l***(***v***) for each ***v*** in ***V***(***G***).
  - The edge set ***E***(***G***) capture the ***affinity*** or ***relative*** distribution of features within object. The edge weight ***w***(***e***) for each ***e*** in ***E***(***G***) captures the attributes of the edge.

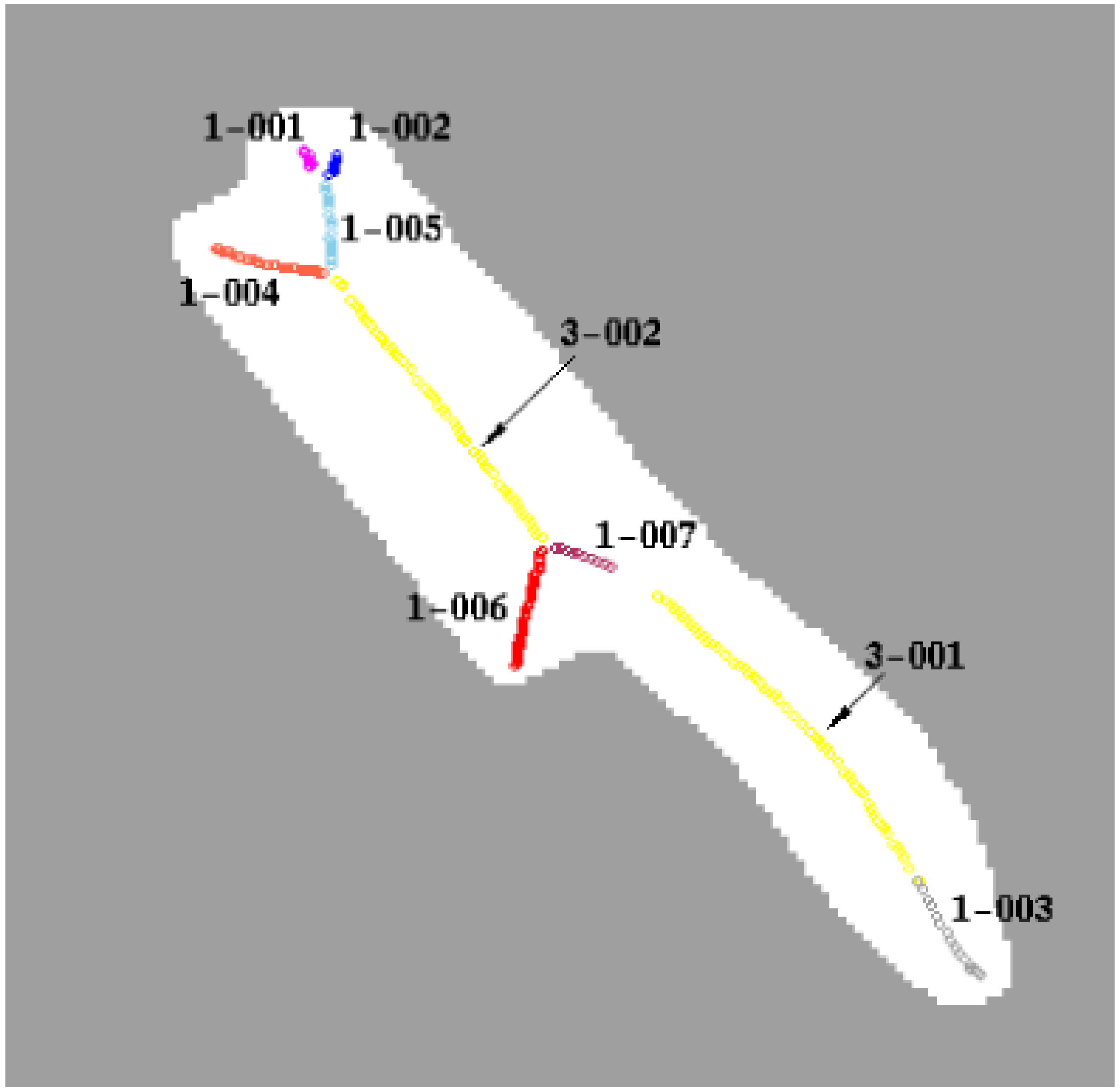


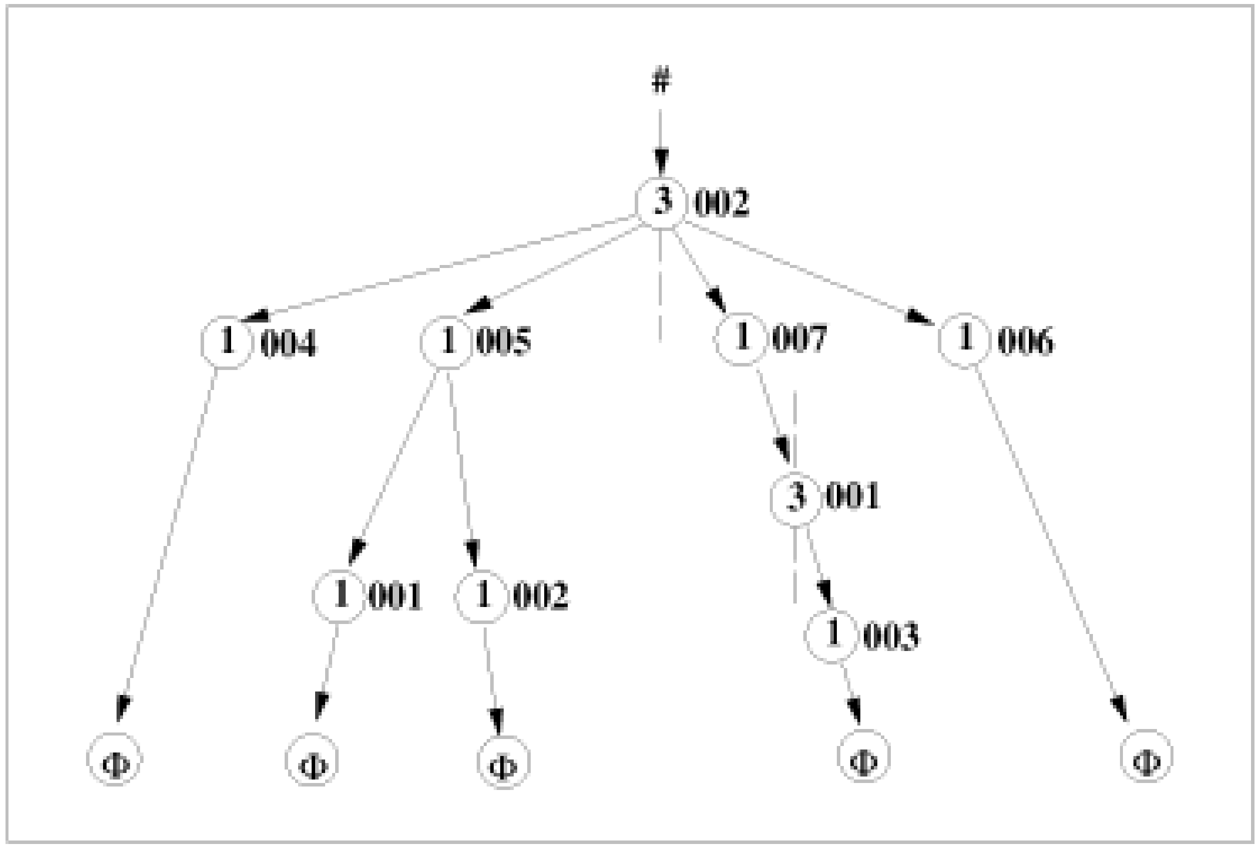
Shock:



Shock Graph (Siddiqi et. al. 1999)

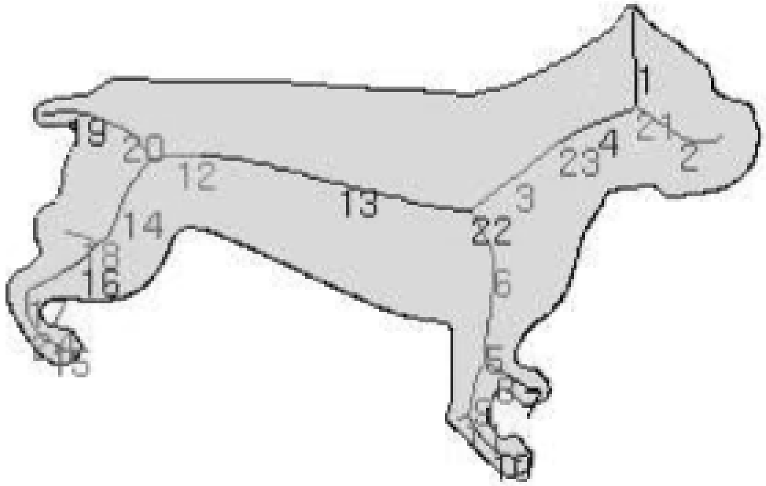
Shock Graphs

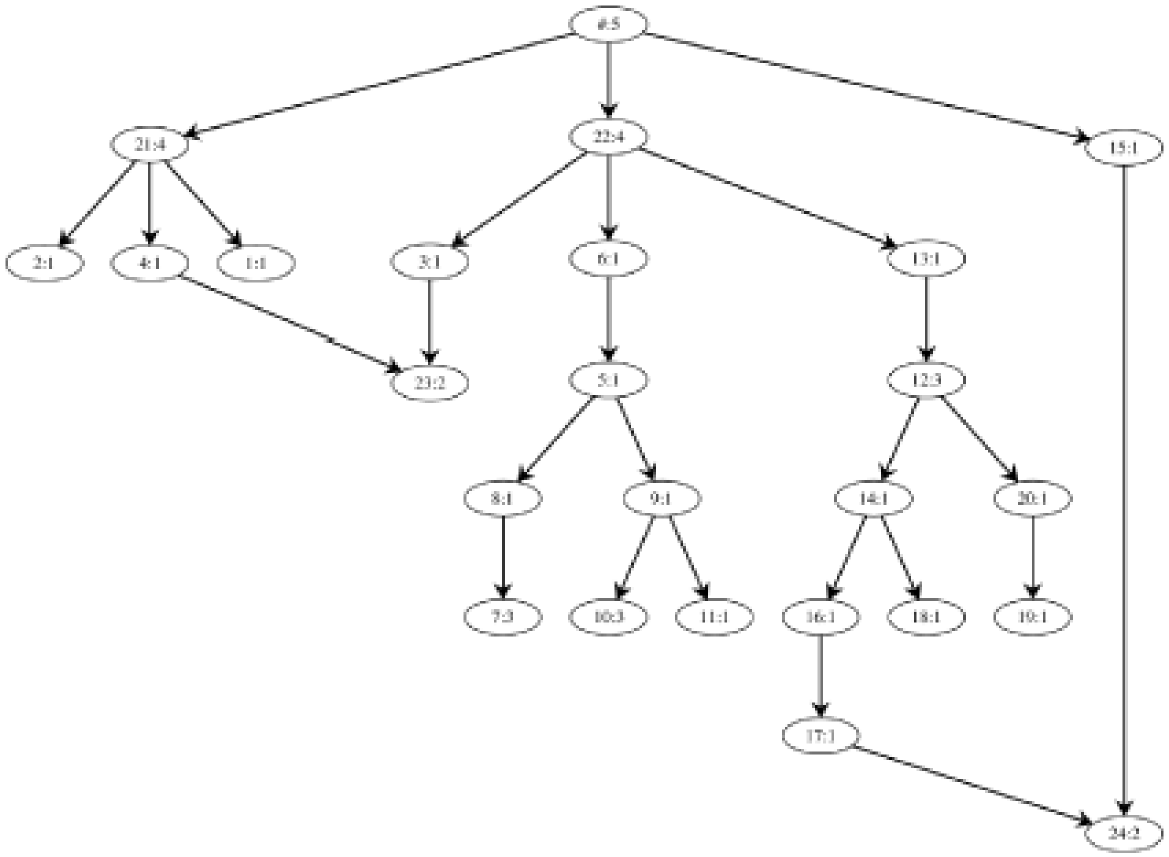




Shock Graphs

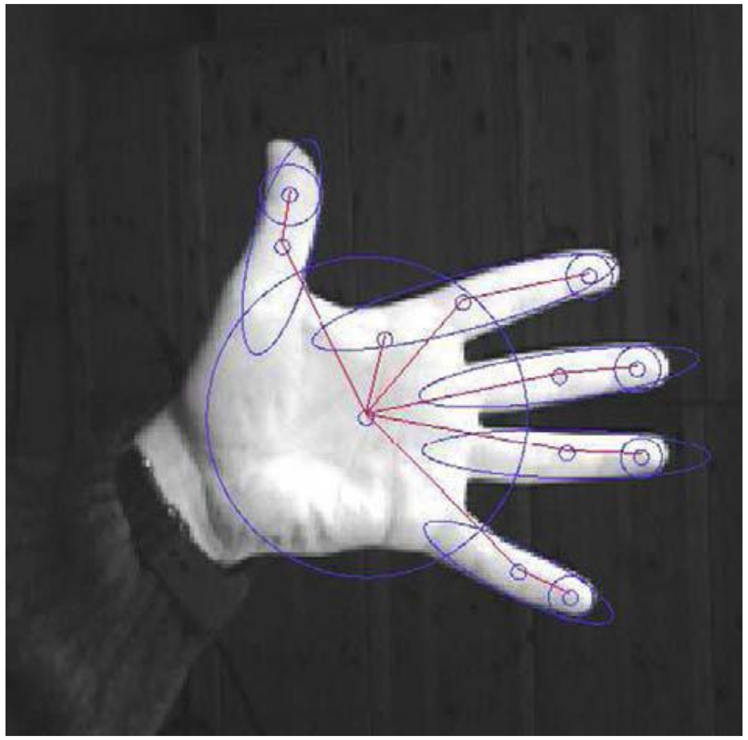
- Representing shape properties.

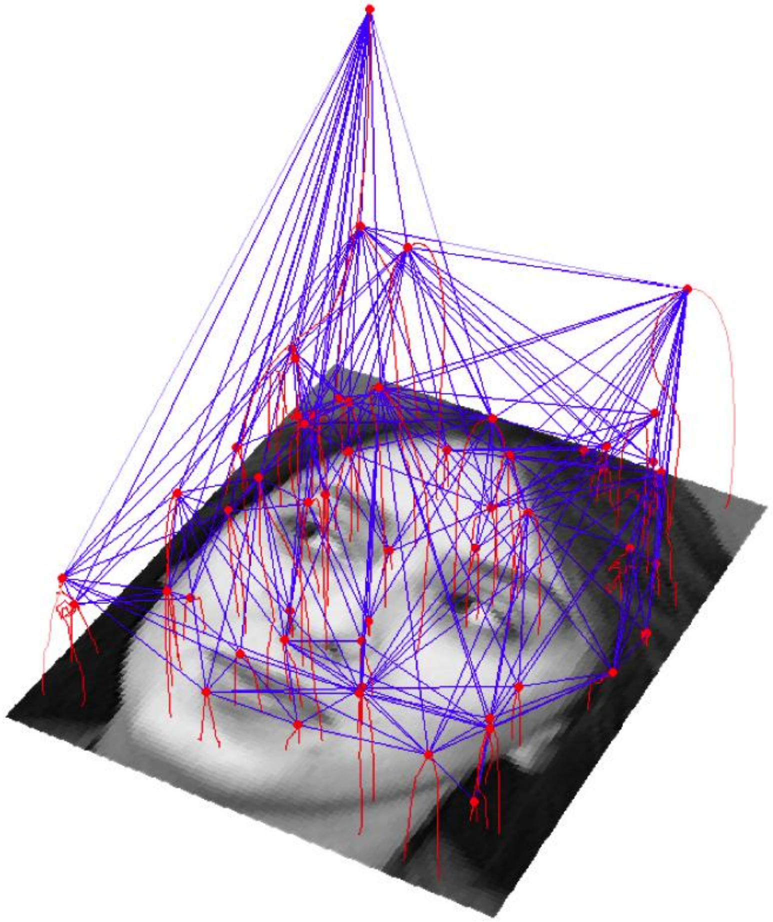




Variations in Reresentation:

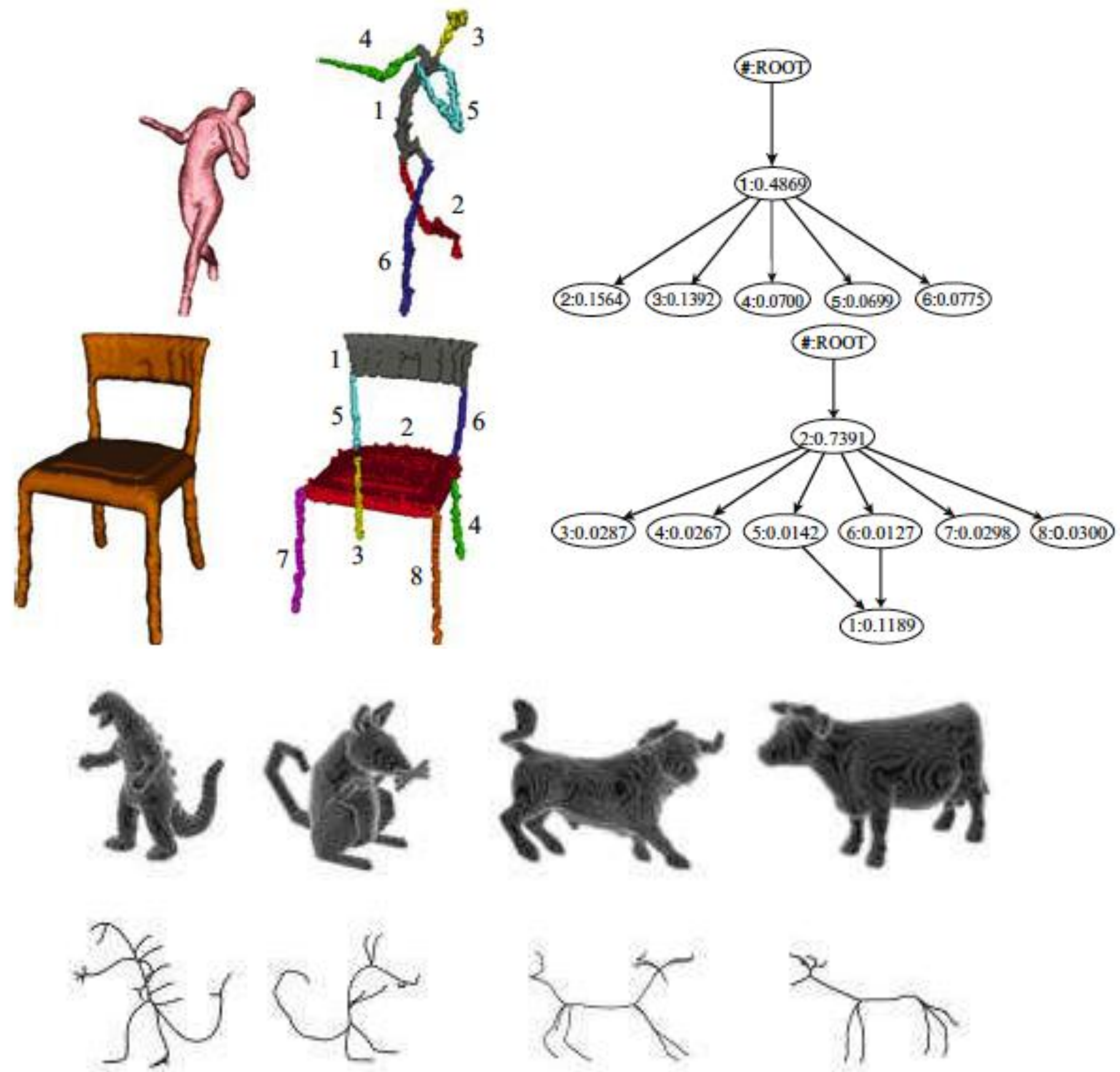
- Representing shape properties.
- Representing appearance features.
- ...





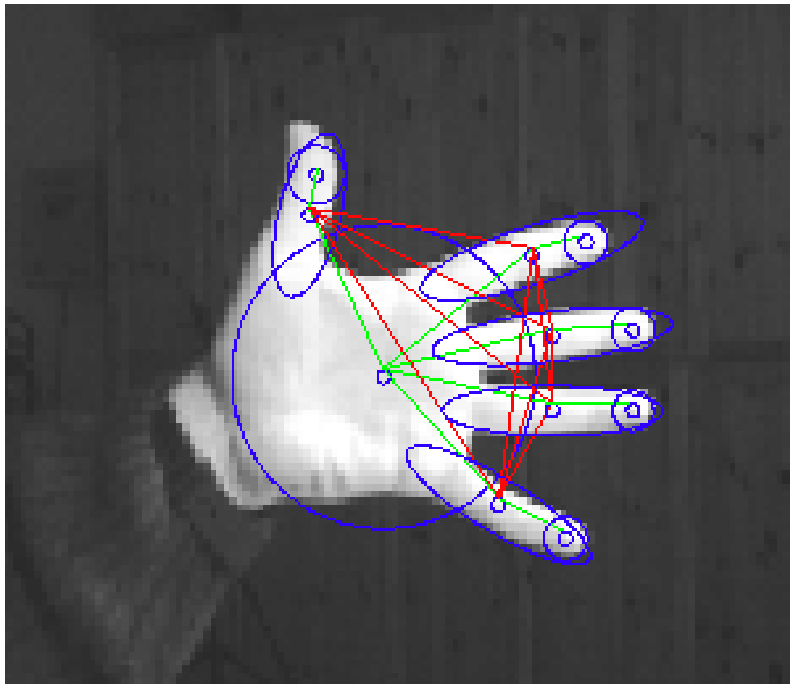
Why Graphs?

- Many reasons:
  - Intuitive (representation)



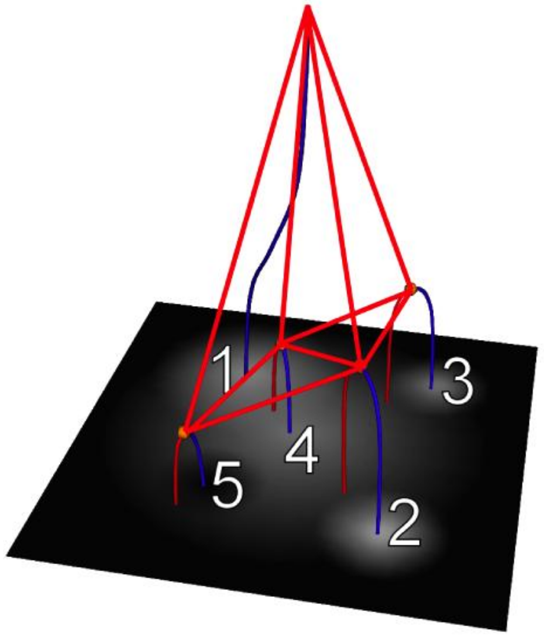
Why Graphs?

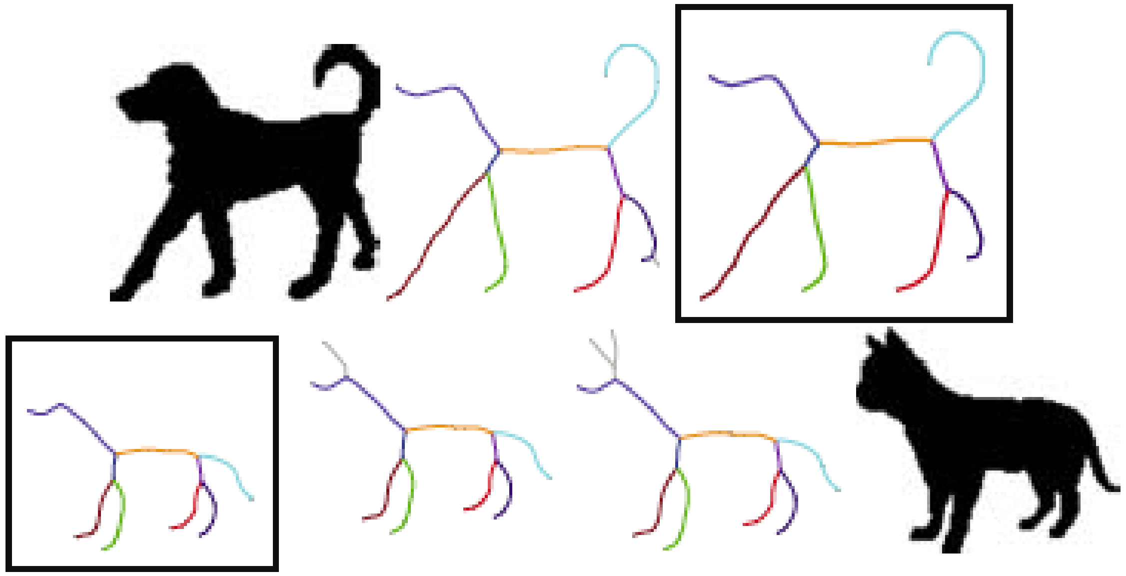
- Many reasons:
  - Intuitive (representation)
  - Compactness (representation)



Why Graphs?

- Many reasons:
  - Intuitive (representation)
  - Compactness (representation)
  - Generative (morphologically)

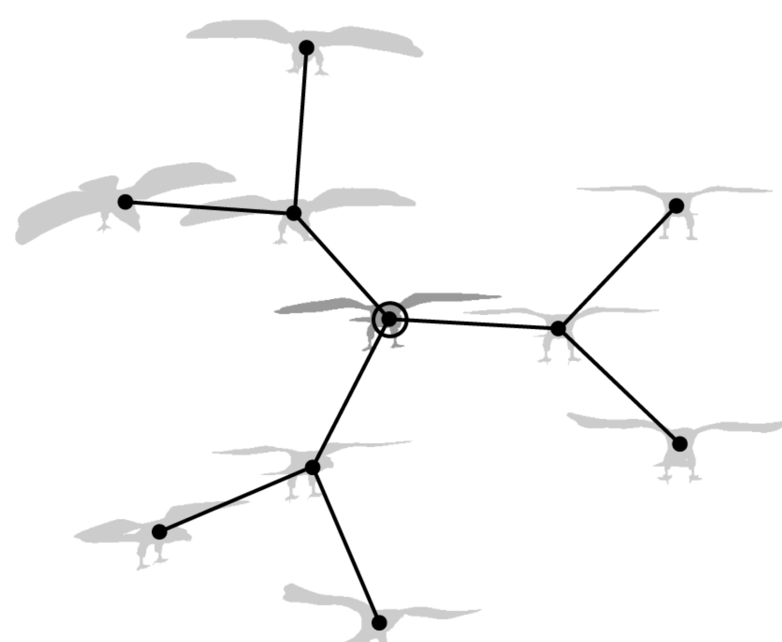


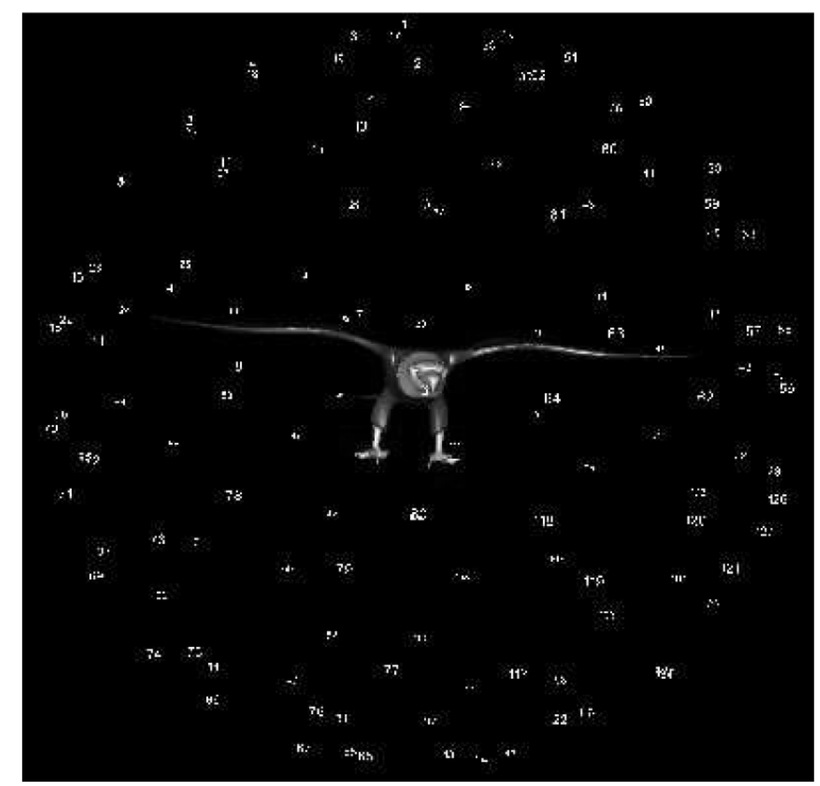


Sebastian et al., 2004

Why Graphs?

- Many reasons:
  - Intuitive (representation)
  - Compactness (representation)
  - Generative (morphologically)
  - Capturing distributions.



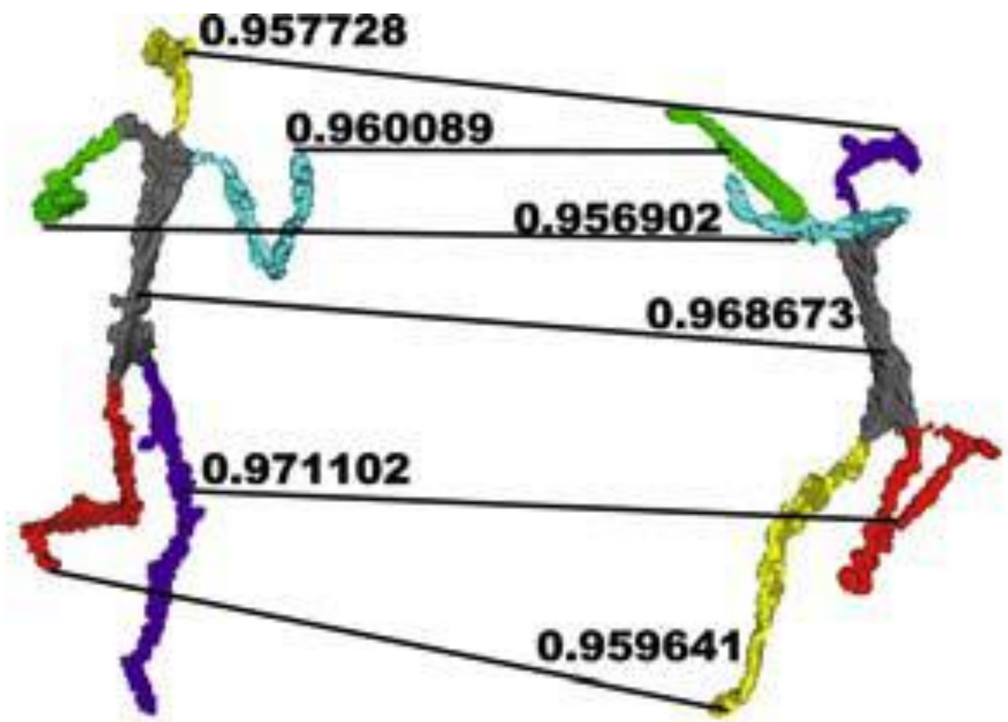


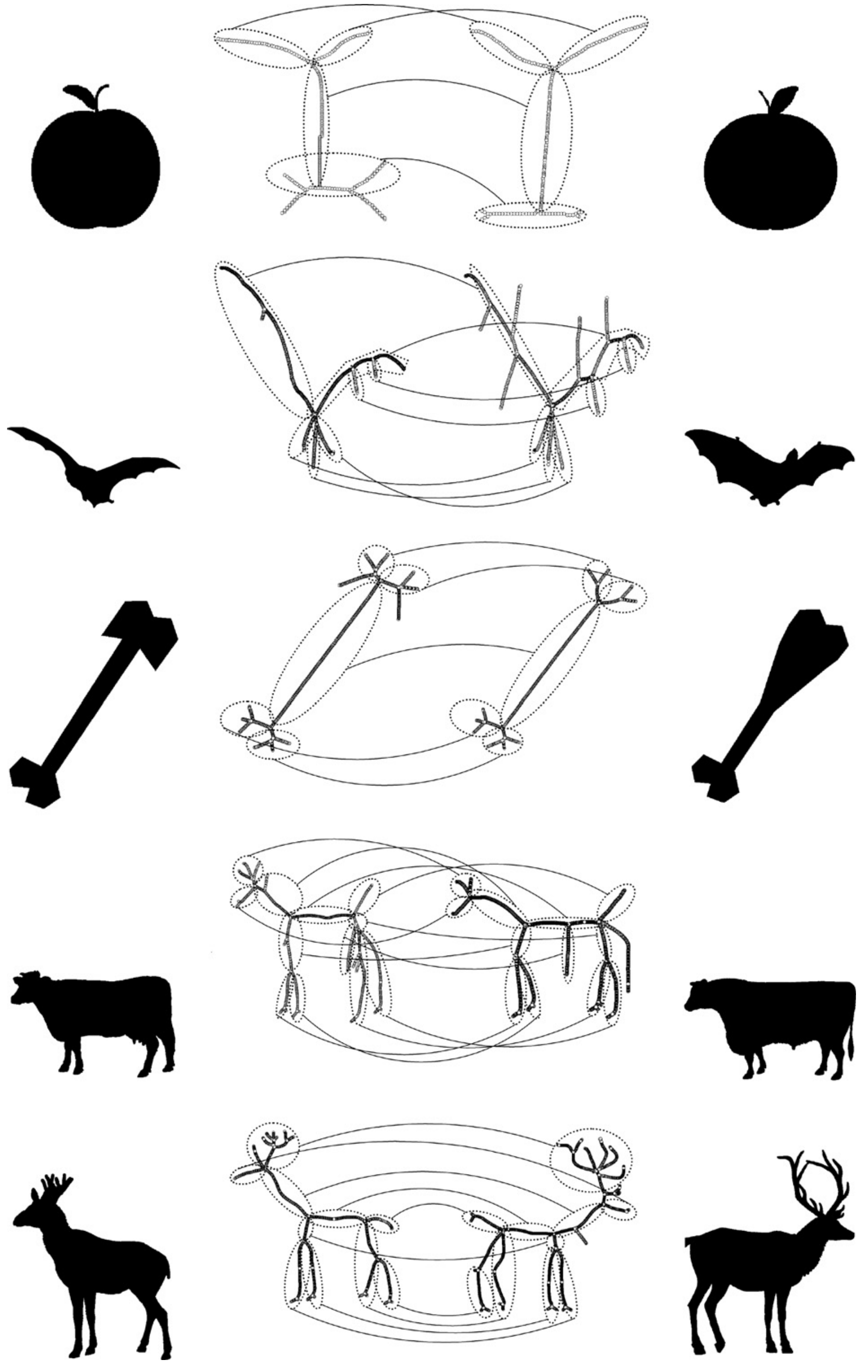
Why Graphs?

- Many reasons:
  - Intuitive (representation)
  - Compactness (representation)
  - Generative (morphologically)
  - Capturing distributions.
    -
    -
    -
  - **Makes computational tasks easier!**

Sample Computational Tasks:

- Shape matching reduced to graph matching





Sample Computational Tasks:

- Shape matching reduced to graph matching.
- Object recognition (affinity matrix).

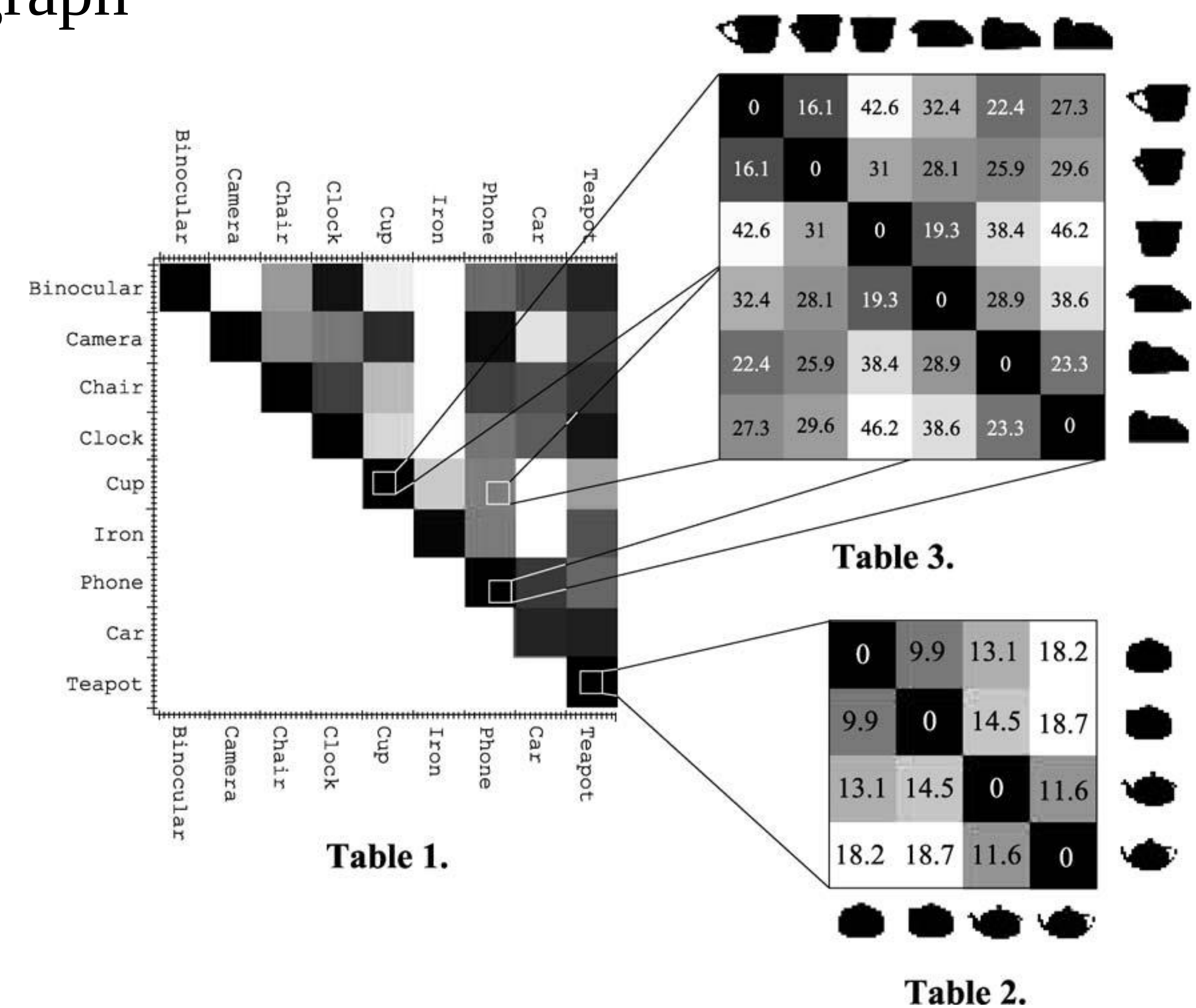


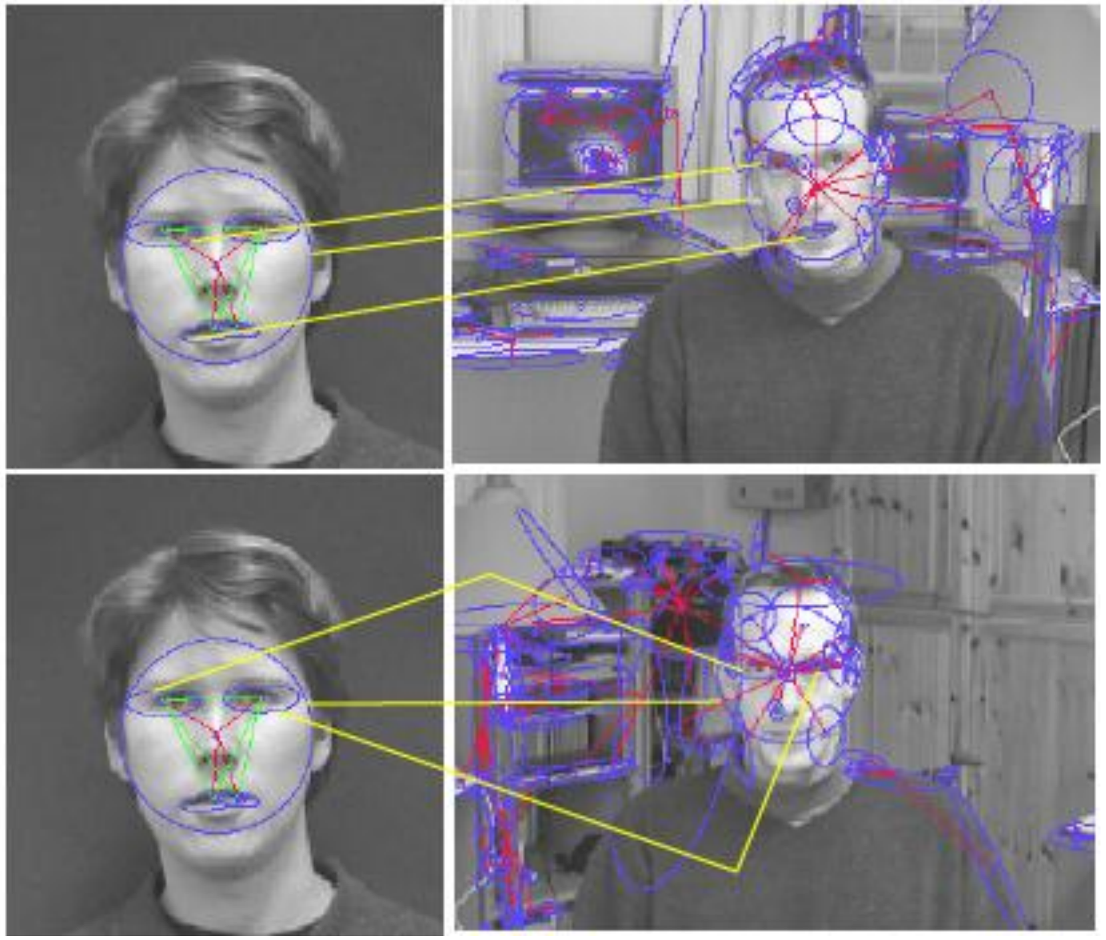
Table 1.

Table 3.

Table 2.

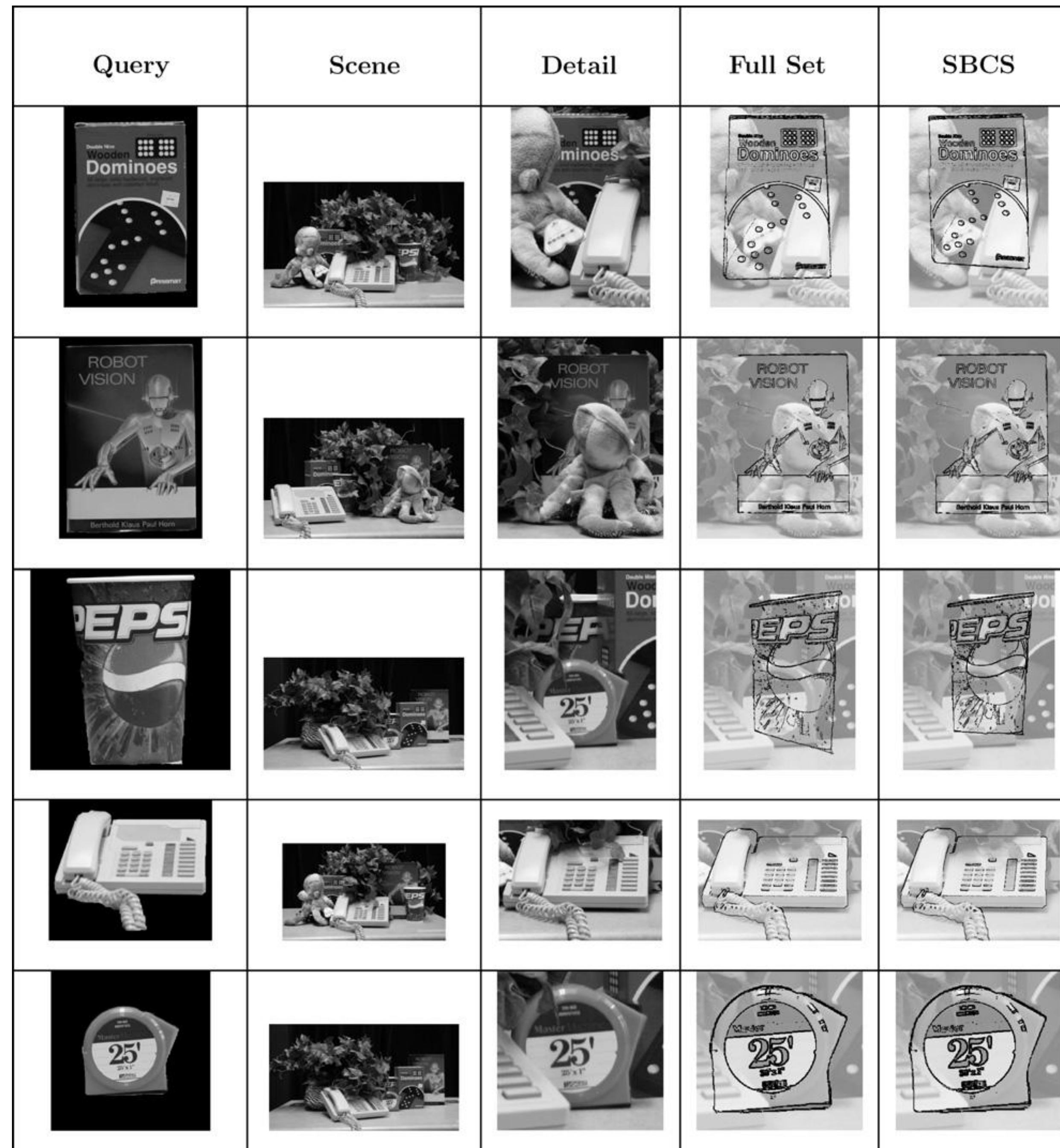
Sample Computational Tasks:

- Shape matching reduced to graph matching.
- Object recognition (cluttered scene).



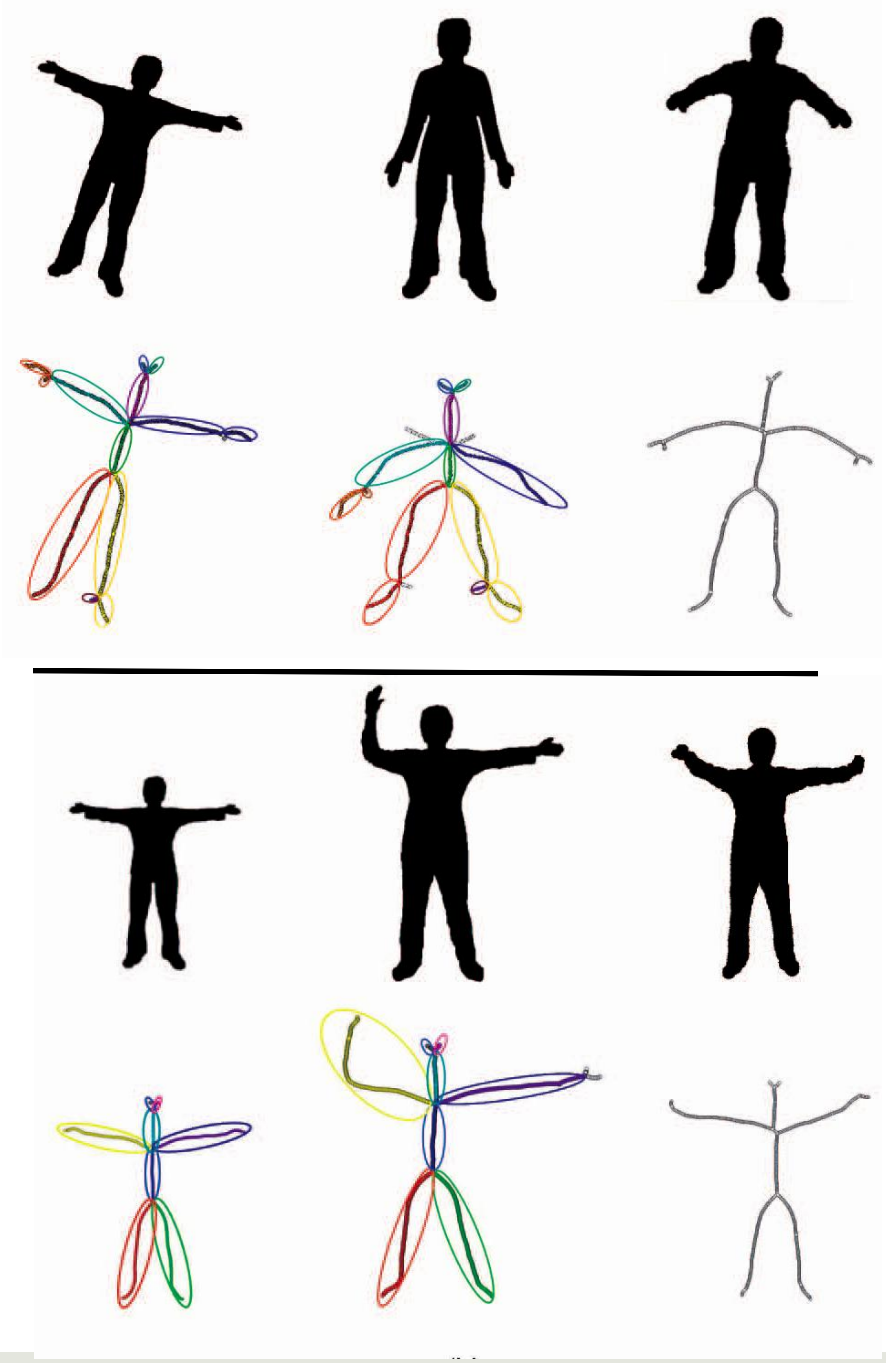
Sample Computational Tasks:

- Shape matching reduced to graph matching.
- Object recognition
- Localization.



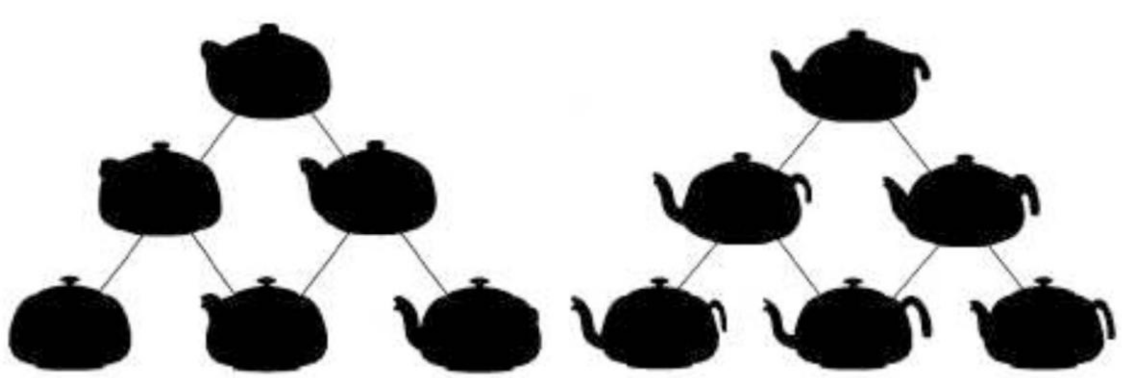
Sample Computational Tasks:

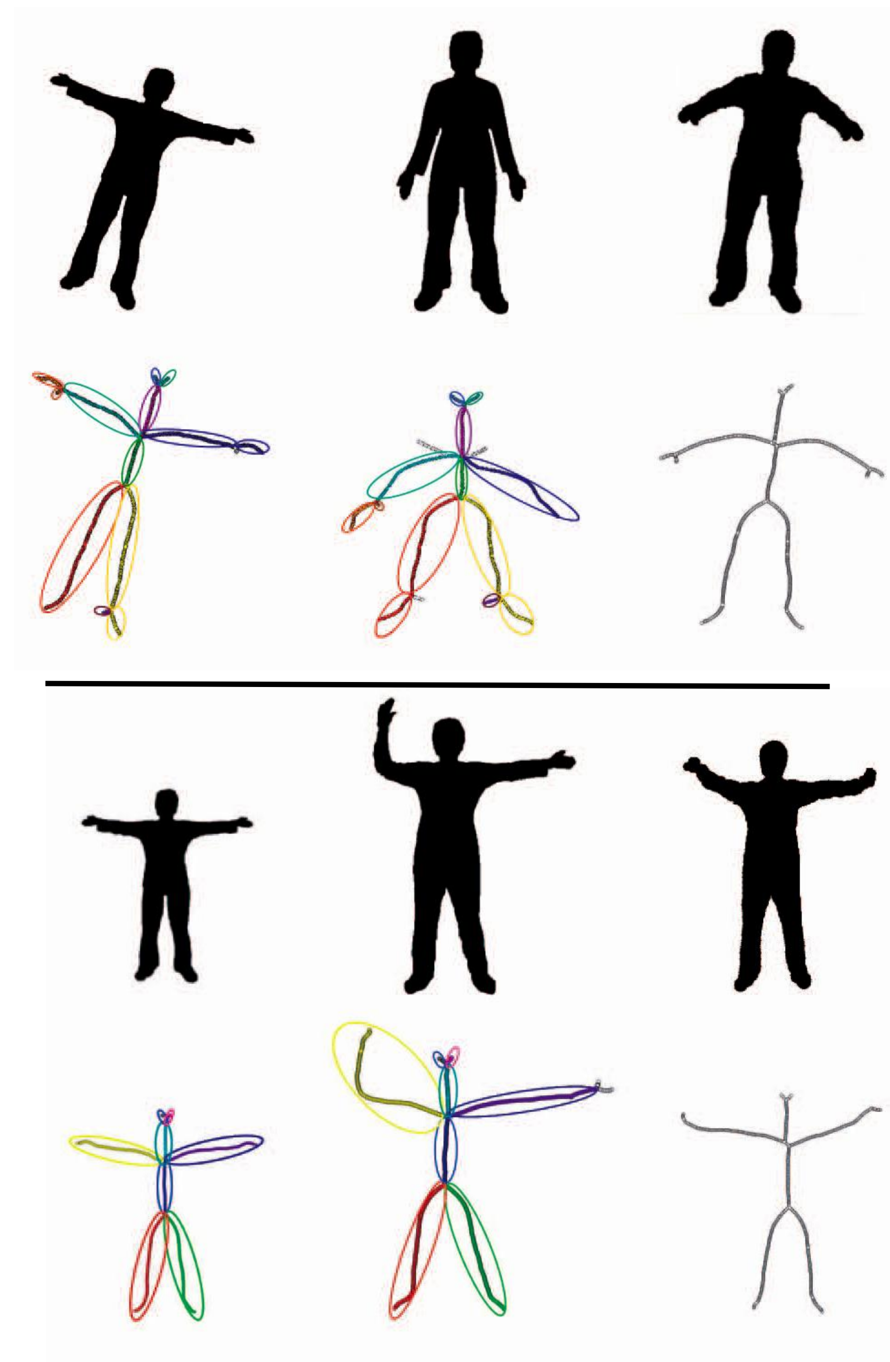
- Shape matching reduced to graph matching.
- Object recognition
- Localization.
- Shape abstraction



Sample Computational Tasks:

- Shape matching reduced to graph matching.
- Object recognition
- Localization.
- Shape abstraction





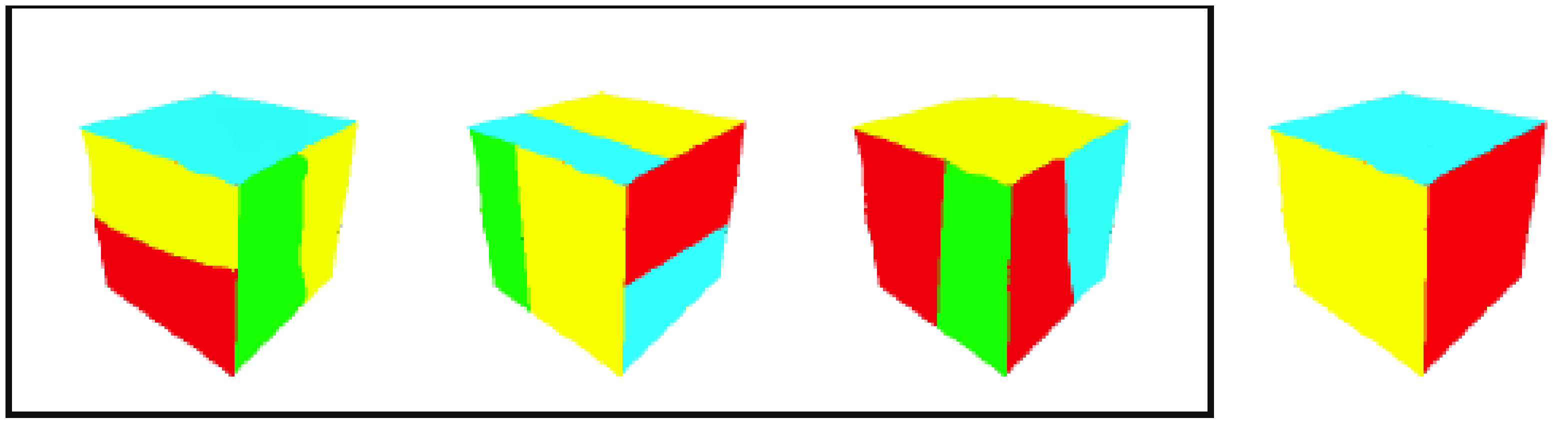
Sample Computational Tasks:

- Shape matching reduced to graph matching.
- Object recognition
- Localization.
- Shape abstraction
- Segmentation

Felzenszwalb and Huttenlocher 2004

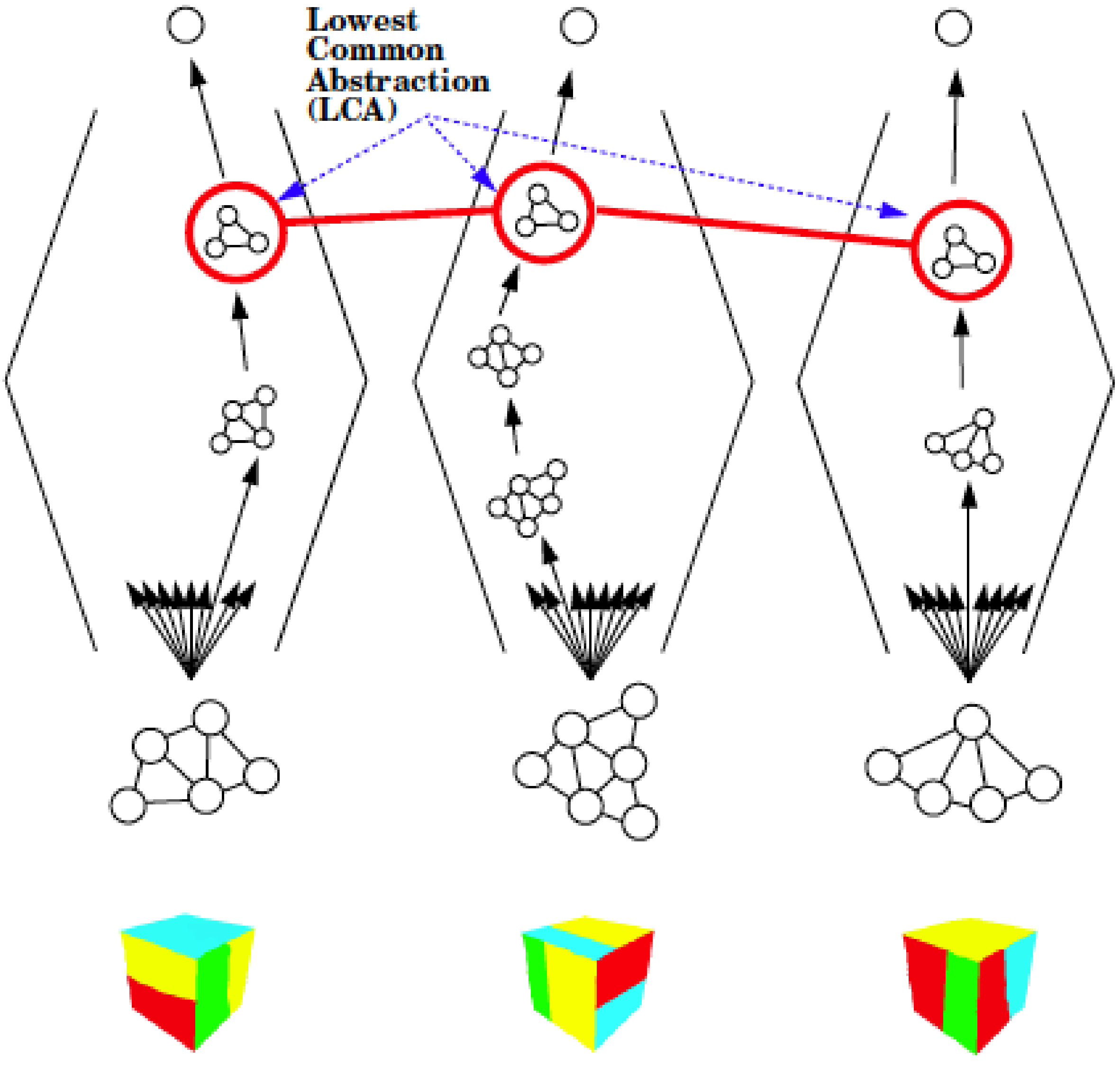
Why is representation hard?

- What is the right level of abstraction?



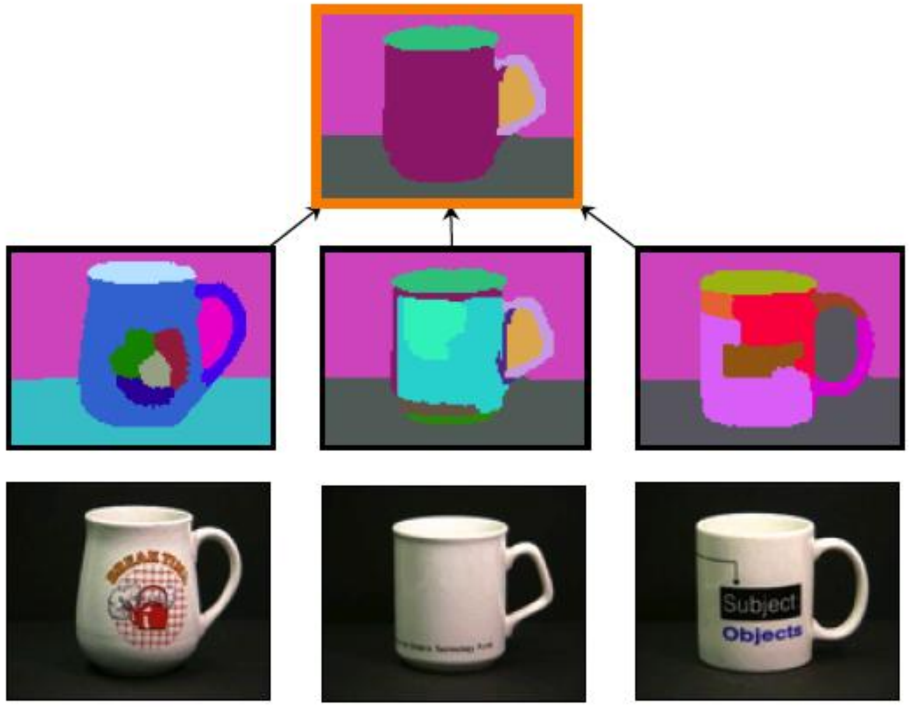
Hardness of Representation

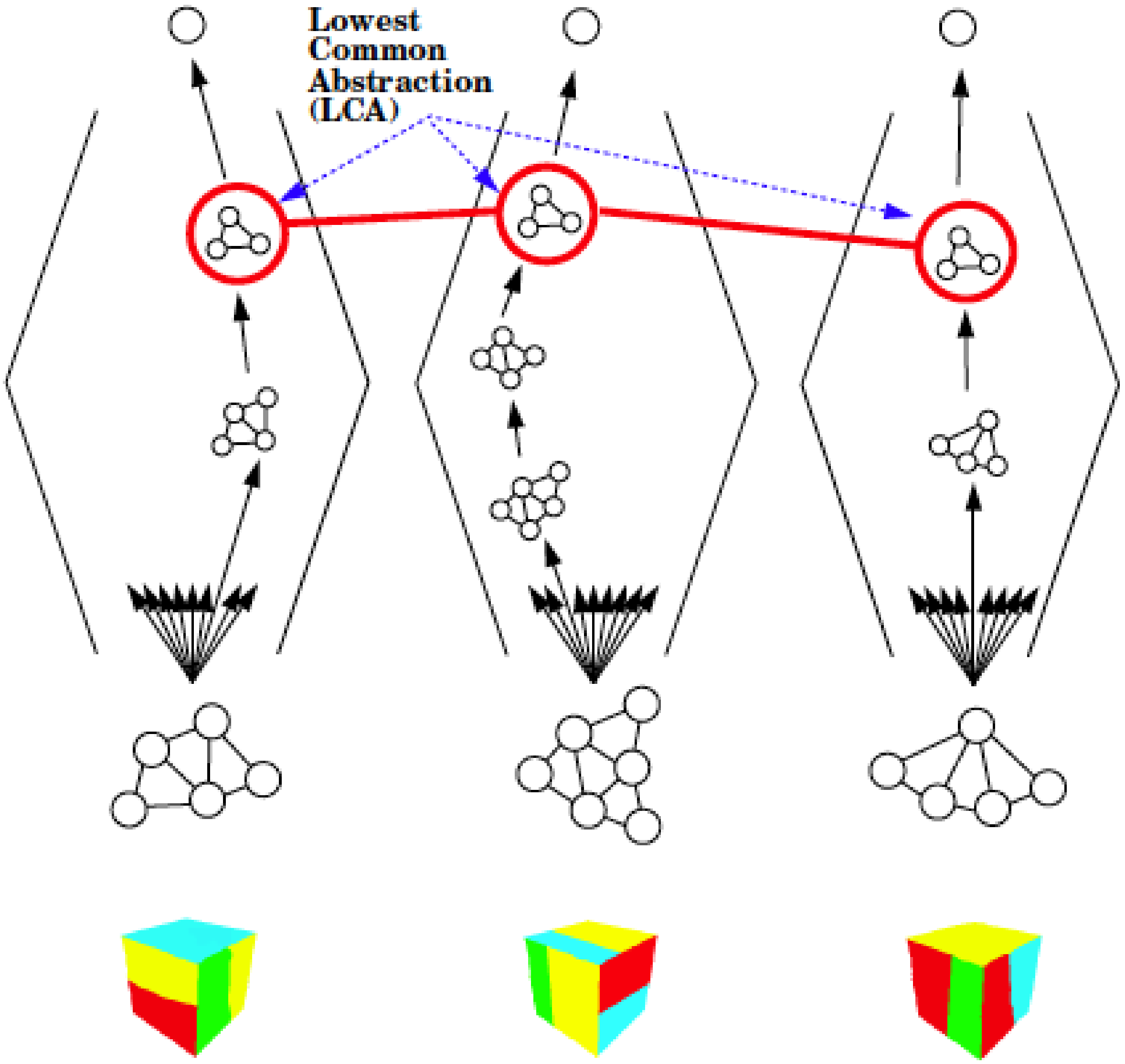
- What is the right level of abstraction?
- Generic model construction requires complex grouping algorithms.



Hardness of Representation

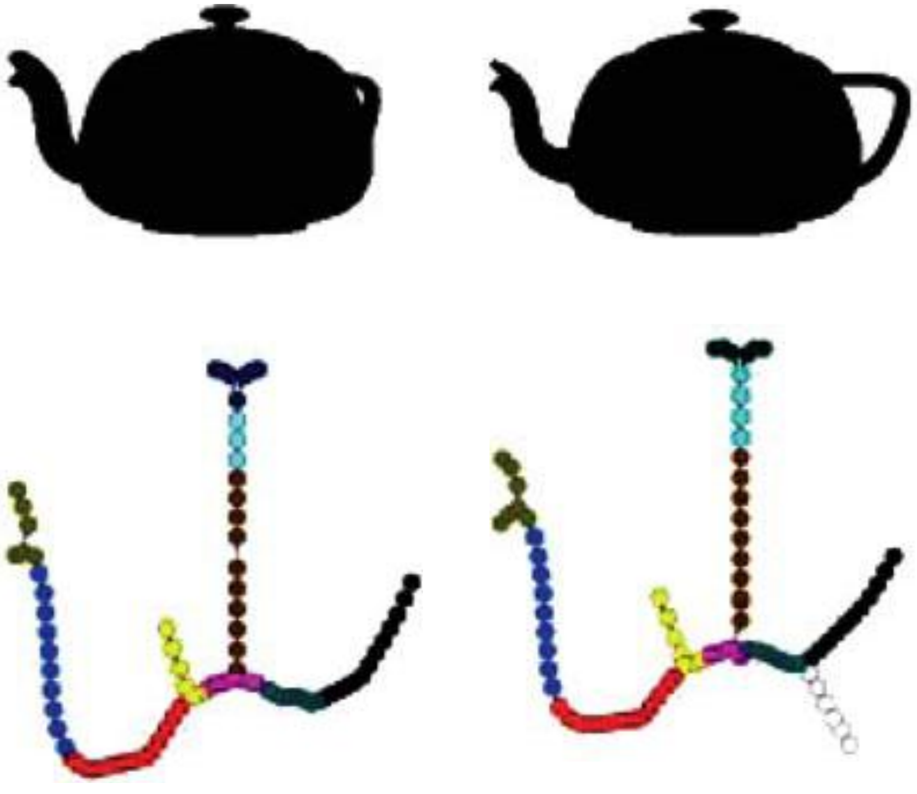
- What is the right level of abstraction?
- Generic model construction requires complex grouping algorithms.





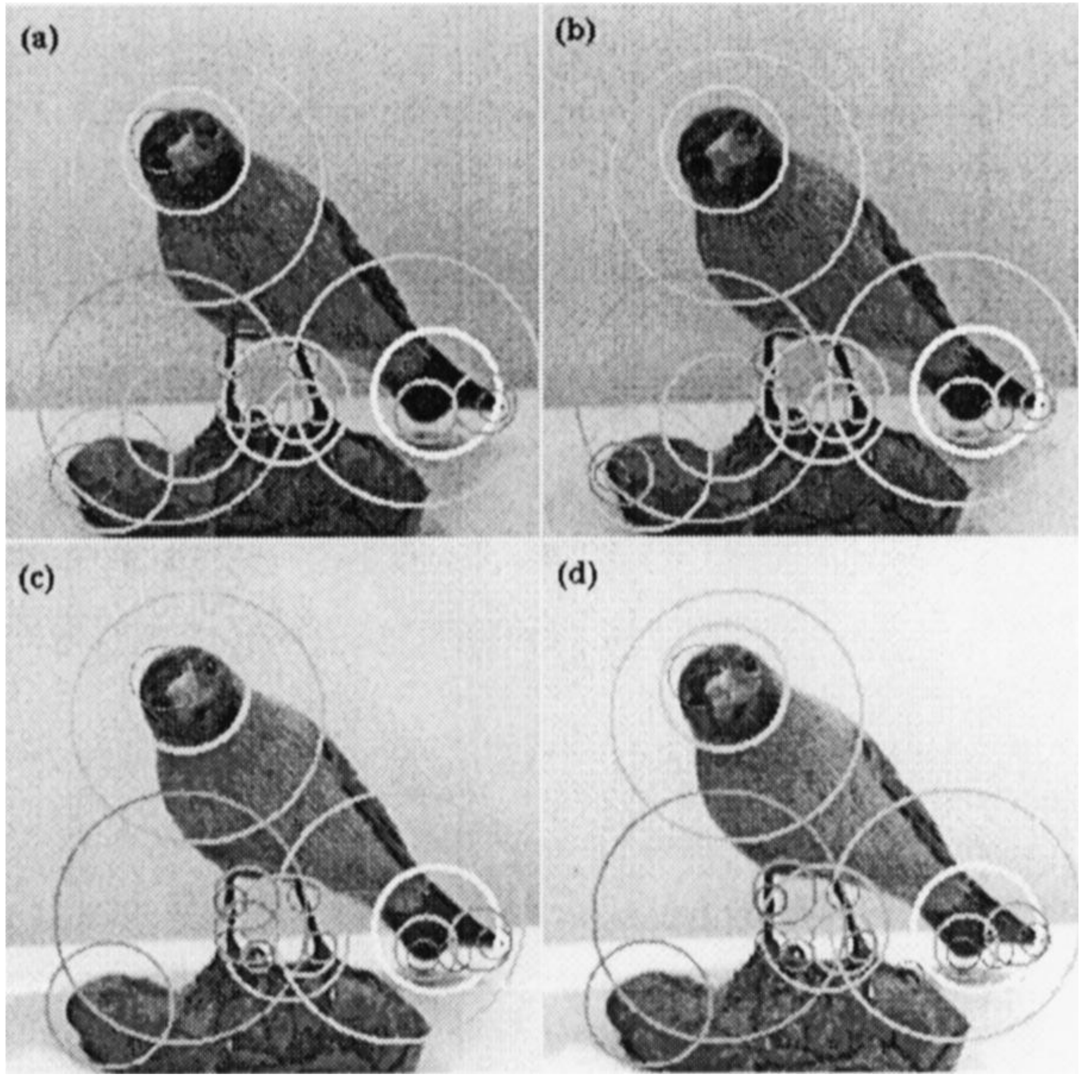
Hardness of Representation

- What is the right level of abstraction?
- Model construction requires complex grouping algorithms.
- Invariance (view point).



Hardness of Representation

- What is the right level of abstraction?
- Model construction requires complex grouping algorithms.
- Invariance (noise).



Overview of this talk

Introduction:

Notations and Definitions

Graphs and Modeling

Algorithmic Graph Theory and Combinatorial Optimization

Problems:

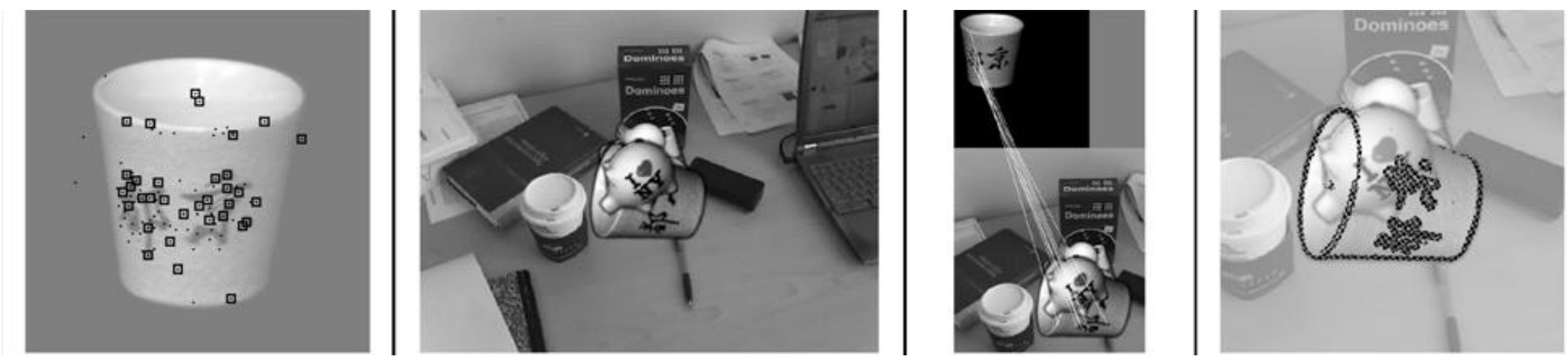
Decision (yes or no) **problems**:

- Does graph ***G*** contain an induced copy of graph ***H***?

Problems:

Decision (yes or no) **problems**:

- Does graph G contain an induced copy of graph H?
- **Localization problem**



Graph Problems:

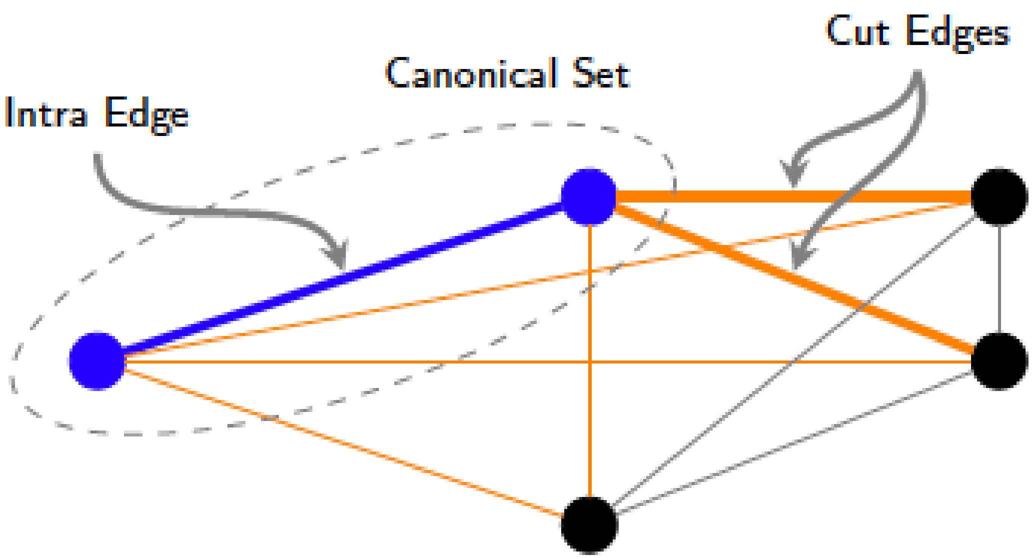
Optimization problems:

- Find the optimal induced substructure in a graph:
  - Maximum cardinality minimum weight matching, minimum spanning tree, maximum clique, maximum hitting set, etc.

Graph Problems:

Optimization problems:

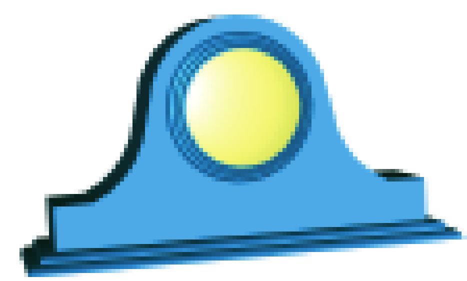
- Find the optimal induced substructure in a graph:
  - Maximum cardinality minimum weight matching, minimum spanning tree, maximum clique, maximum hitting set, etc.
- **Max-cut dominating sets (Canonical sets)**

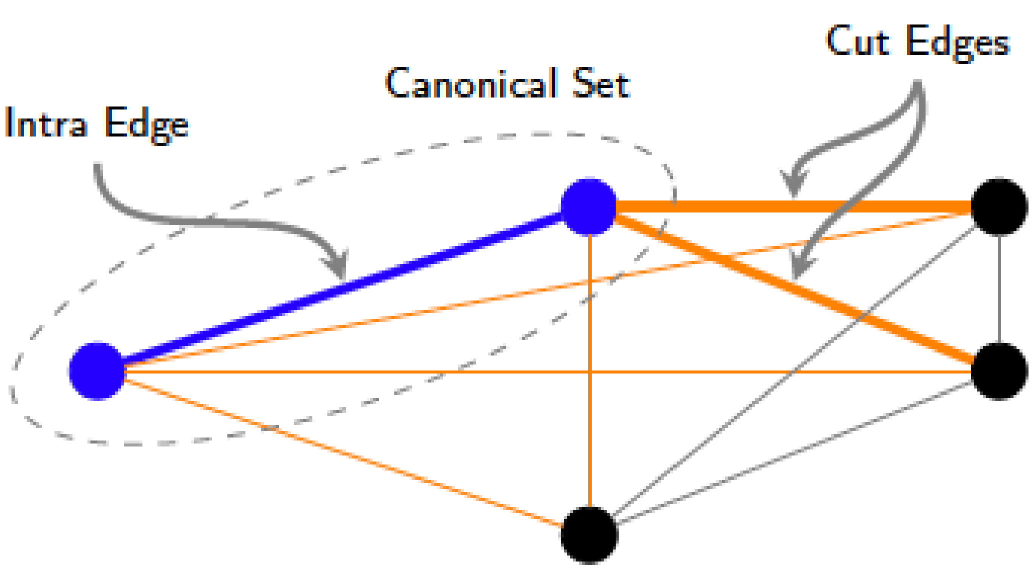


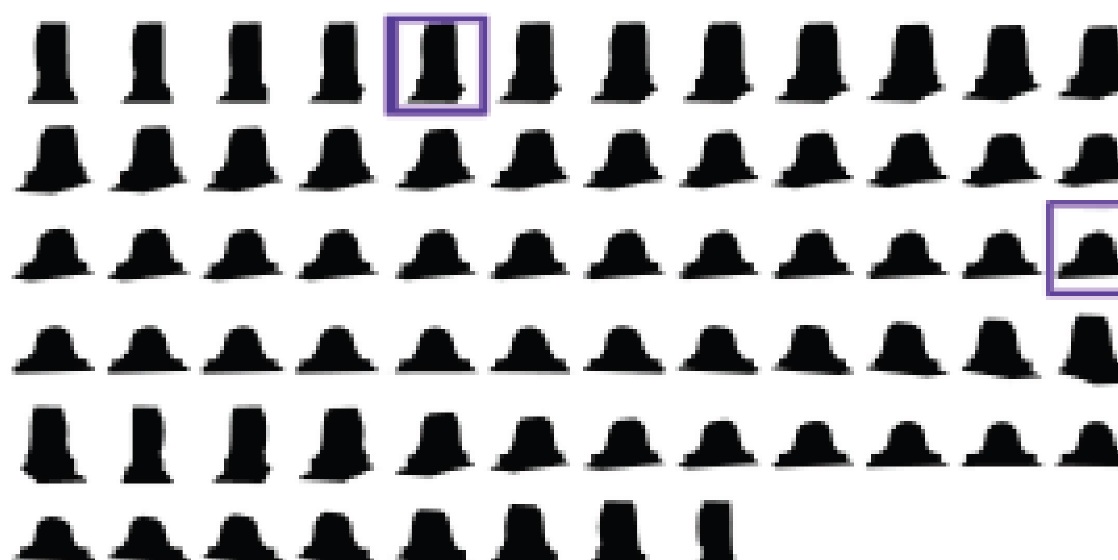
Graph Problems:

Optimization problems:

- Find the optimal induced substructure in a graph:
  - Maximum cardinality minimum weight matching, minimum spanning tree, maximum clique, maximum hitting set, etc.
- **Max-cut dominating sets (Canonical sets)**

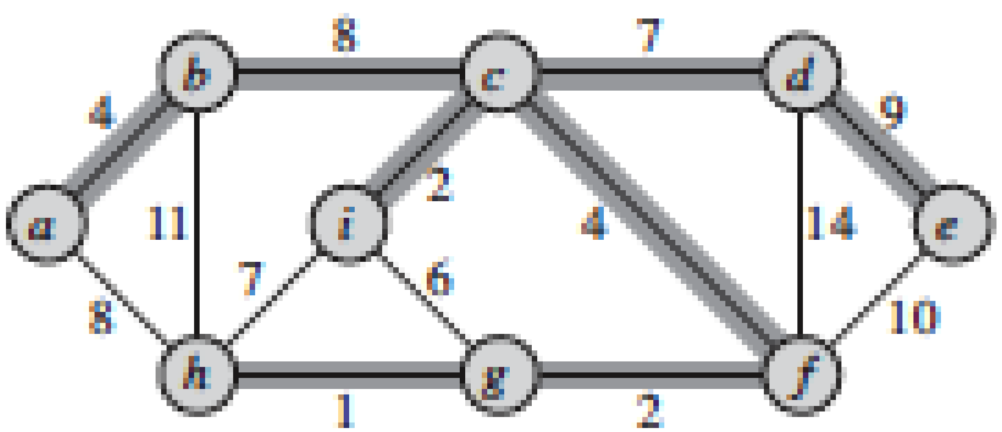






Algorithmic Graph Theory:

- **Objective:** Designing efficient combinatorial methods for solving decision or optimization problems.
  - Runs in polynomial number of steps in terms of size of the graph; $\boldsymbol{n}=|\boldsymbol{V}(\mathbf{G})|$ and $\boldsymbol{m}=|\boldsymbol{E}(\boldsymbol{G})|$.
  - **Example**: Minimum Spanning Tree (MST): $\boldsymbol{T}(\boldsymbol{m},\boldsymbol{n})=\boldsymbol{O}(\boldsymbol{m}+\boldsymbol{n}\ \mathbf{log}\ \boldsymbol{n})$.

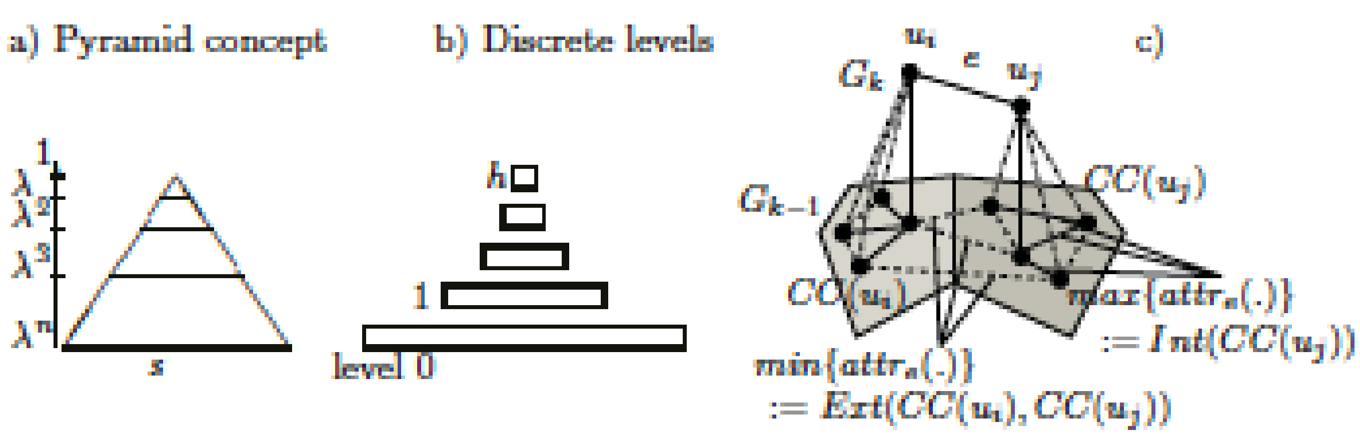


Intro. to Algorithms. Corman et al.

```
GENERIC-MST(G, w)
A = ∅
while A does not form a spanning tree
    find an edge (u, v) that is safe for A
    A = A ∪ {(u, v)}
return A
```

Algorithmic Graph Theory:

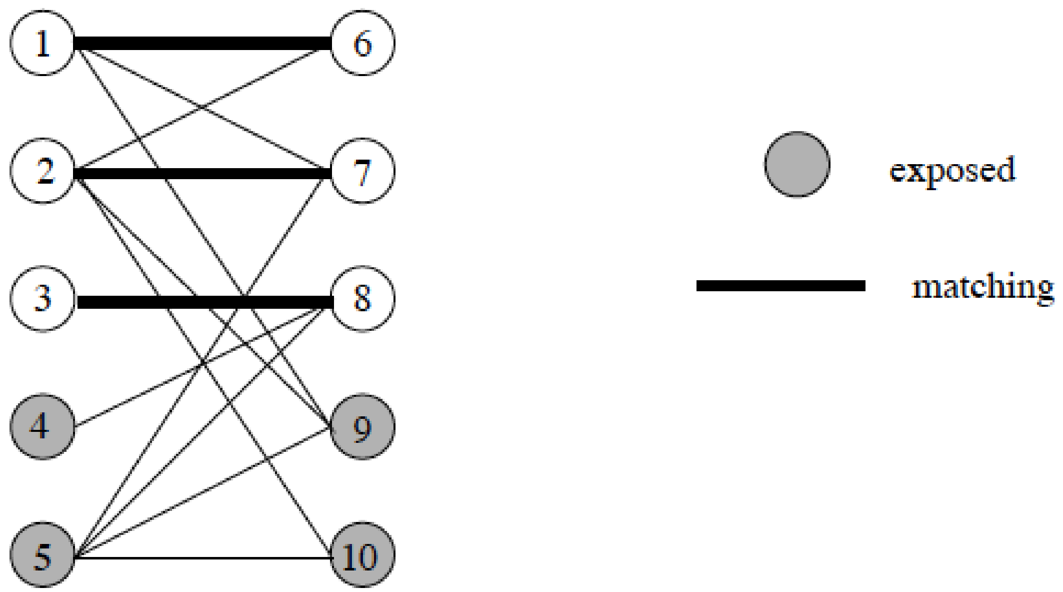
- **Objective:** Designing efficient combinatorial methods for solving decision or optimization problems.
  - Runs in polynomial number of steps in terms of size of the graph; $\boldsymbol{n}=|\boldsymbol{V}(\mathbf{G})|$ and $\boldsymbol{m}=|\boldsymbol{E}(\boldsymbol{G})|$.
  - Example: Minimum Spanning Tree (MST): $\boldsymbol{T}(\boldsymbol{m},\boldsymbol{n})=\boldsymbol{O}(\boldsymbol{m}+\boldsymbol{n}\ \mathbf{log}\ \boldsymbol{n})$.



Haxhimusa and Kropatsch, 2004

Algorithmic Graph Theory:

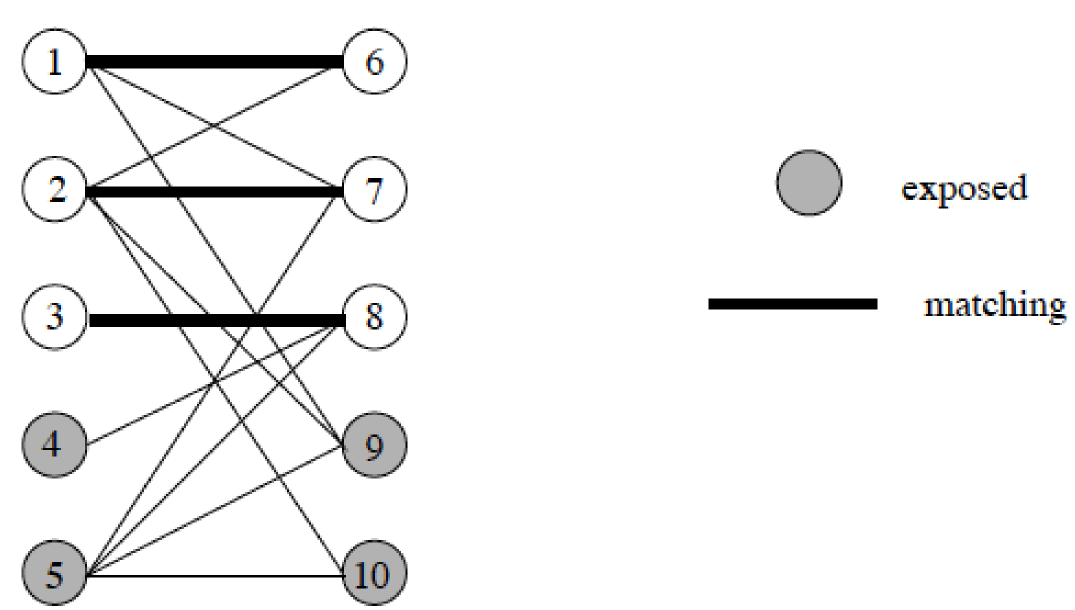
- **Objective:** Designing efficient combinatorial methods for solving decision or optimization problems.
  - Runs in polynomial number of steps in terms of size of the graph; $\boldsymbol{n}=|\boldsymbol{V}(\mathbf{G})|$ and $\boldsymbol{m}=|\boldsymbol{E}(\boldsymbol{G})|$.
  - Optimality of solution:
    - Example: Maximum matching problem



Algorithmic Graph Theory:

- **Objective:** Designing efficient combinatorial methods for solving decision or optimization problems.
  - Runs in polynomial number of steps in terms of size of the graph; ***n***=|***V***(**G**)| and ***m***=|***E***(***G***)|.
  - Optimality of solution:
    - Example: Maximum matching problem

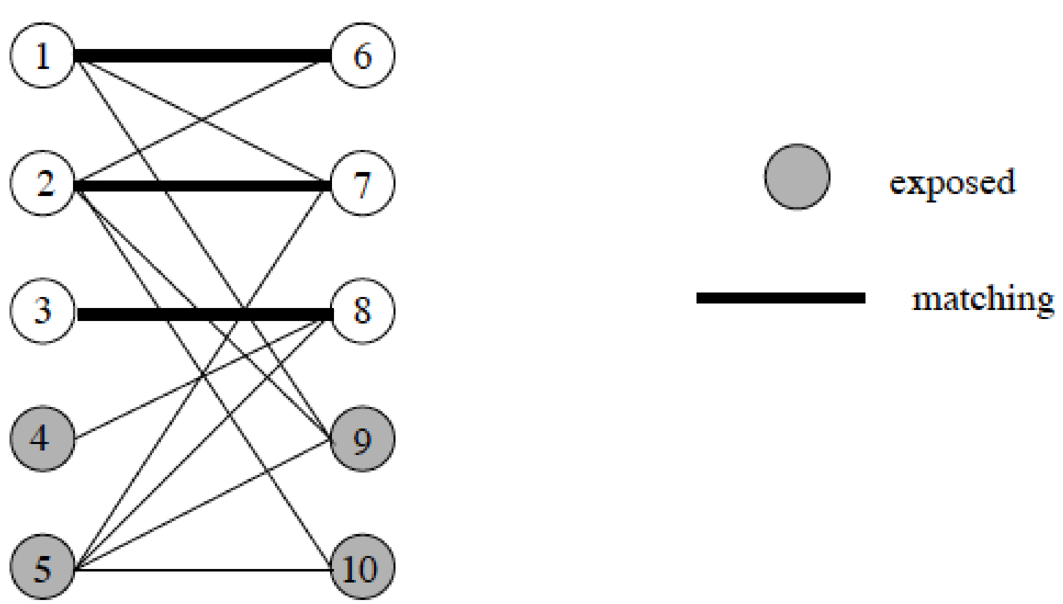
$$
\begin{aligned}
d_{GM}(\mathbf{x},\mathbf{y}) = \max_{E} \quad & \sum_{uv} s_{uv} e_{uv} \\
\text{s.t. } \quad & e_{ij} \in \{0,1\} \\
& \sum_{u} e_{uv} \leq 1 \\
& \sum_{v} e_{uv} \leq 1
\end{aligned}
$$



Algorithmic Graph Theory:

- **Objective:** Designing efficient combinatorial methods for solving decision or optimization problems.
  - Runs in polynomial number of steps in terms of size of the graph; ***n***=|***V***(**G**)| and ***m***=|***E***(***G***)|.
  - Optimality of solution:
    - Example: Maximum matching problem

$$
\begin{aligned}
d_{GM}(\mathbf{x},\mathbf{y}) = \max_{E} \quad & \sum_{uv} s_{uv} e_{uv} \\
\text{s.t.} \quad & \boxed{e_{ij} \in \{0,1\}} \\
& \sum_{u} e_{uv} \leq 1 \\
& \sum_{v} e_{uv} \leq 1
\end{aligned}
$$



Algorithmic Graph Theory:

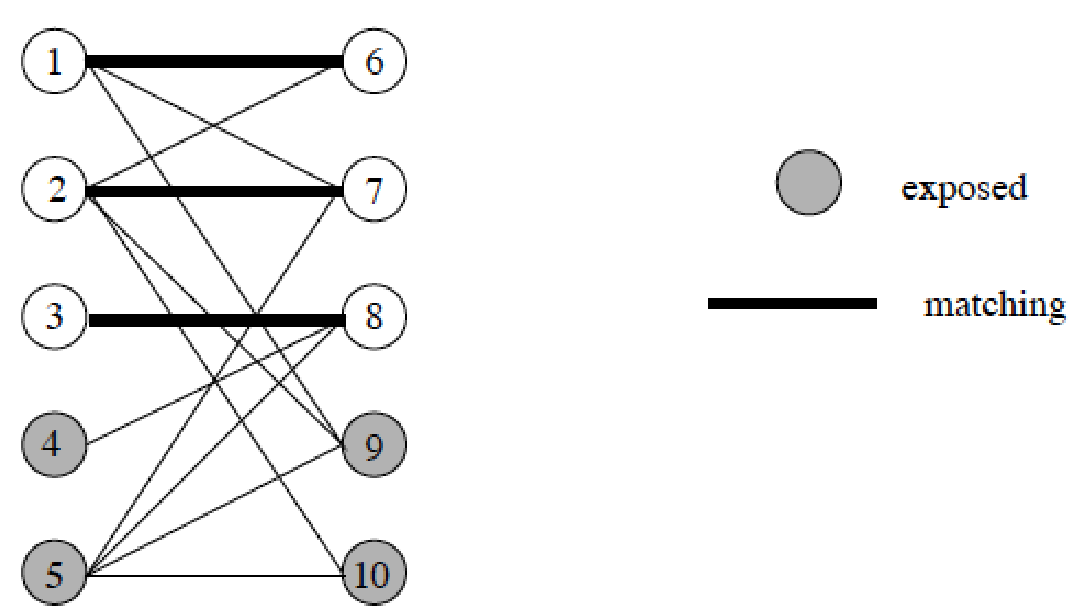
- **Objective:** Designing efficient combinatorial methods for solving decision or optimization problems.
  - Runs in polynomial number of steps in terms of size of the graph; ***n***=|***V***(**G**)| and ***m***=|***E***(***G***)|.
  - Optimality of solution:
    - Example: Maximum matching problem

$$d_{GM}(\mathbf{x},\mathbf{y}) = \boxed{\max_{E} \sum_{uv} s_{uv} e_{uv}}$$

$$\text{s.t. } e_{ij} \in \{0,1\}$$

$$\sum_{u} e_{uv} \leq 1$$

$$\sum_{v} e_{uv} \leq 1$$



Algorithmic Graph Theory:

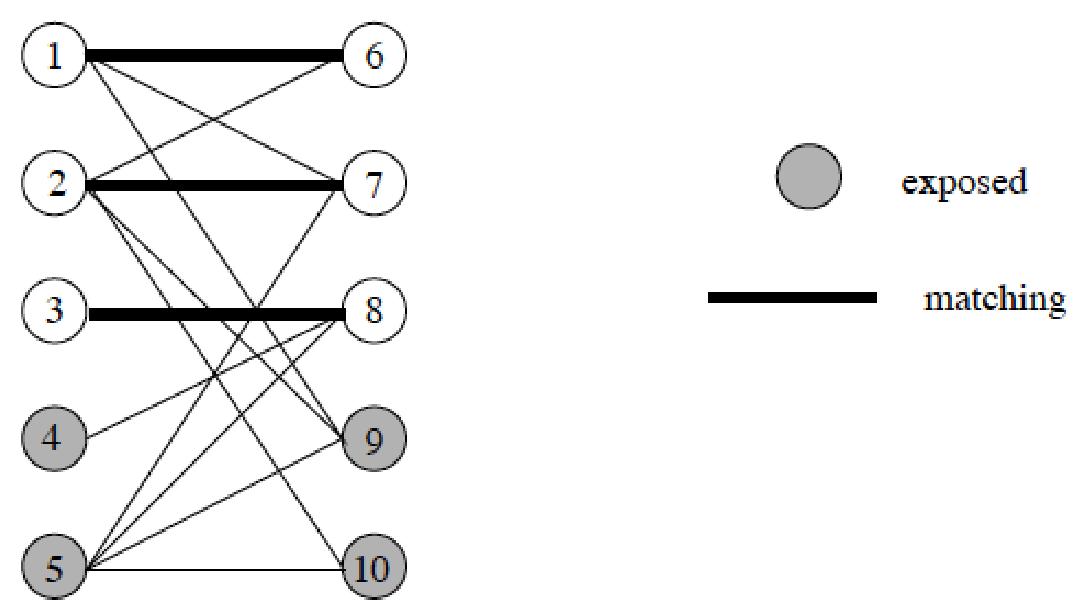
- **Objective:** Designing efficient combinatorial methods for solving decision or optimization problems.
  - Runs in polynomial number of steps in terms of size of the graph; $\boldsymbol{n}=|\boldsymbol{V}(\mathbf{G})|$ and $\boldsymbol{m}=|\boldsymbol{E}(\boldsymbol{G})|$.
  - Optimality of solution:
    - Example: Maximum matching problem

$$d_{GM}(\mathbf{x},\mathbf{y}) = \max_{E} \sum_{uv} s_{uv} e_{uv}$$

$$\text{s.t. } e_{ij} \in \{0,1\}$$

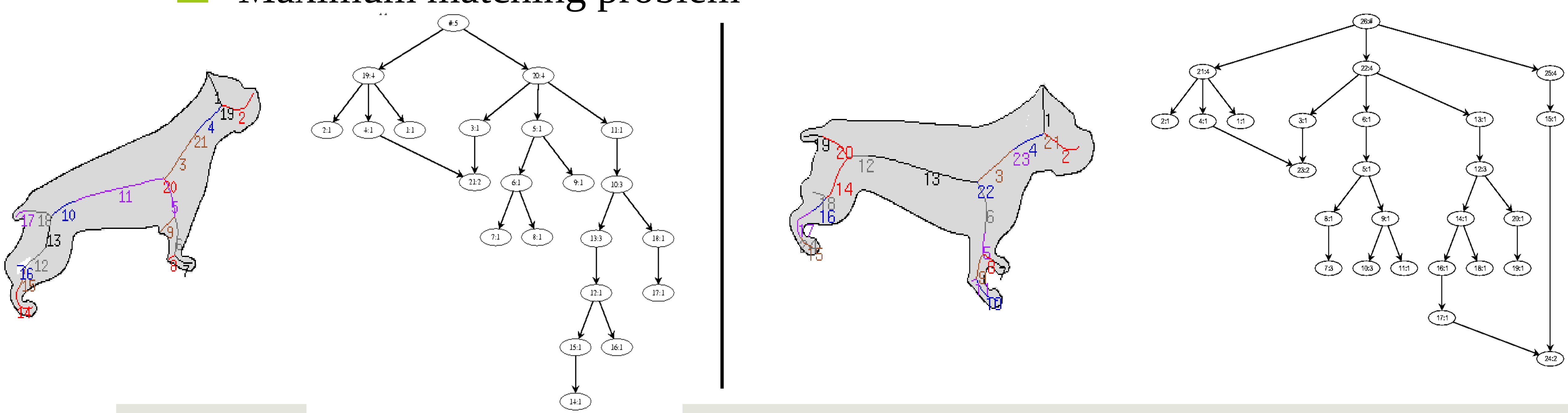
$$\sum_{u} e_{uv} \leq 1$$

$$\sum_{v} e_{uv} \leq 1$$



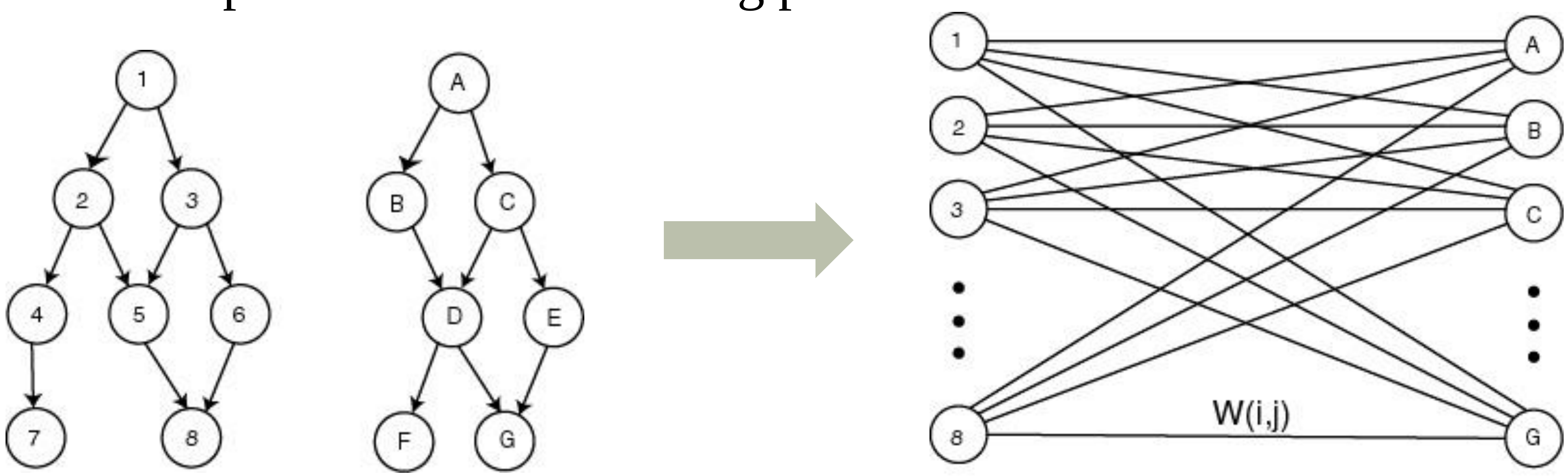
Algorithmic Graph Theory:

- **Objective:** Designing efficient combinatorial methods for solving decision or optimization problems.
  - Runs in polynomial number of steps in terms of size of the graph; $\boldsymbol{n}=|\boldsymbol{V}(\mathbf{G})|$ and $\boldsymbol{m}=|\boldsymbol{E}(\boldsymbol{G})|$.
  - Optimality of solution:
    - Maximum matching problem



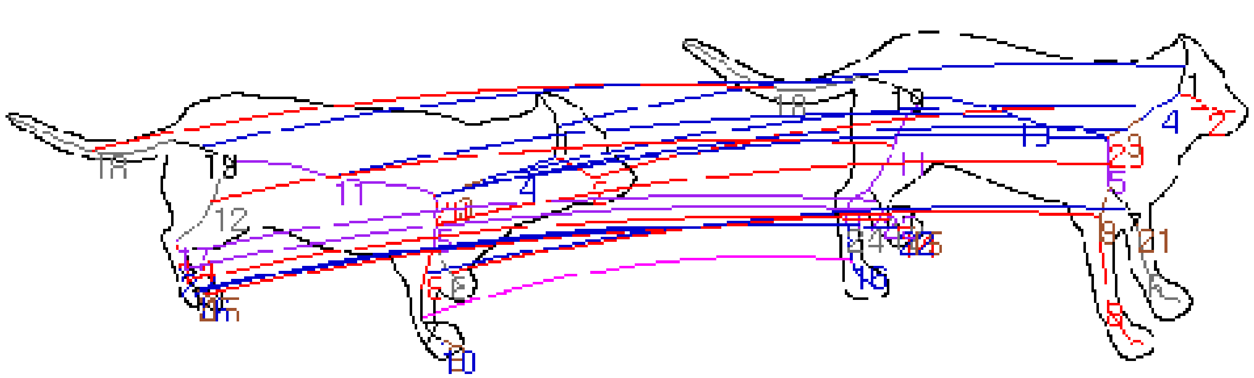
Algorithmic Graph Theory:

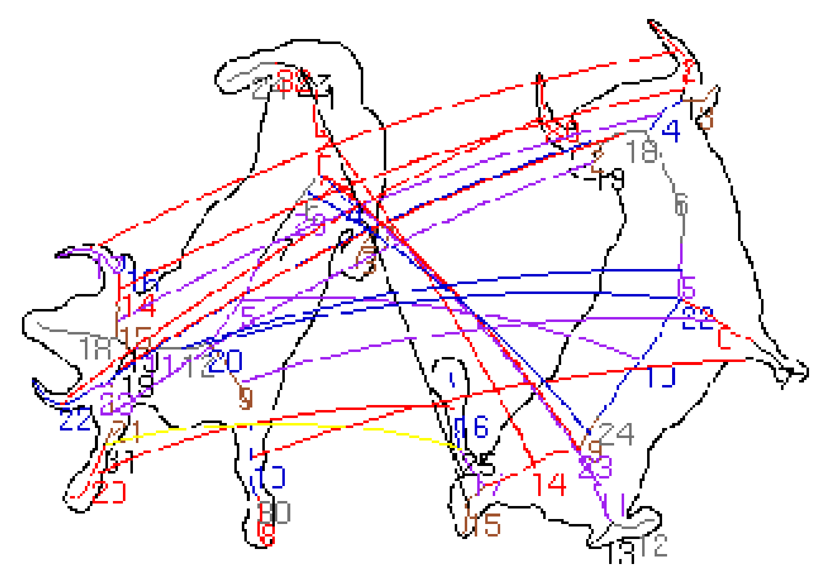
- **Objective:** Designing efficient combinatorial methods for solving decision or optimization problems.
  - Runs in polynomial number of steps in terms of size of the graph; $\boldsymbol{n}=|\boldsymbol{V}(\mathbf{G})|$ and $\boldsymbol{m}=|\boldsymbol{E}(\boldsymbol{G})|$.
  - Optimality of solution:
    - Example: Maximum matching problem



Algorithmic Graph Theory:

- **Objective:** Designing efficient combinatorial methods for solving decision or optimization problems.
  - Runs in polynomial number of steps in terms of size of the graph; $\boldsymbol{n}=|\boldsymbol{V}(\mathbf{G})|$ and $\boldsymbol{m}=|\boldsymbol{E}(\boldsymbol{G})|$.
  - Optimality of solution:
    - Example: Maximum matching problem





Algorithmic Graph Theory:

- **Objective:** Designing efficient combinatorial methods for solving decision or optimization problems.
  - Runs in polynomial number of steps in terms of size of the graph; $\boldsymbol{n}=|\boldsymbol{V}(\mathbf{G})|$ and $\boldsymbol{m}=|\boldsymbol{E}(\boldsymbol{G})|$.
  - Optimality of solution.

- **Bad news:** most of the combinatorial optimization problems involving graphs are computationally intractable:
  - traveling salesman problem, maximum cut problem, independent set problem, maximum clique problem, minimum vertex cover problem, maximum independent set problem, multidimensional matching problem,…

Algorithmic Graph Theory:

- **Dealing with the intractability:**
  - Bounded approximation algorithms
  - Suboptimal heuristics.

Algorithmic Graph Theory:

Bounded approximation algorithms

- Example: Vertex cover problem:
  - **A *vertex cover* of an undirected graph $G=(V,E)$ is a subset V' of V such that if (u,v) is an edge in E, then u or v (or both) belong to V'.**

Algorithmic Graph Theory:

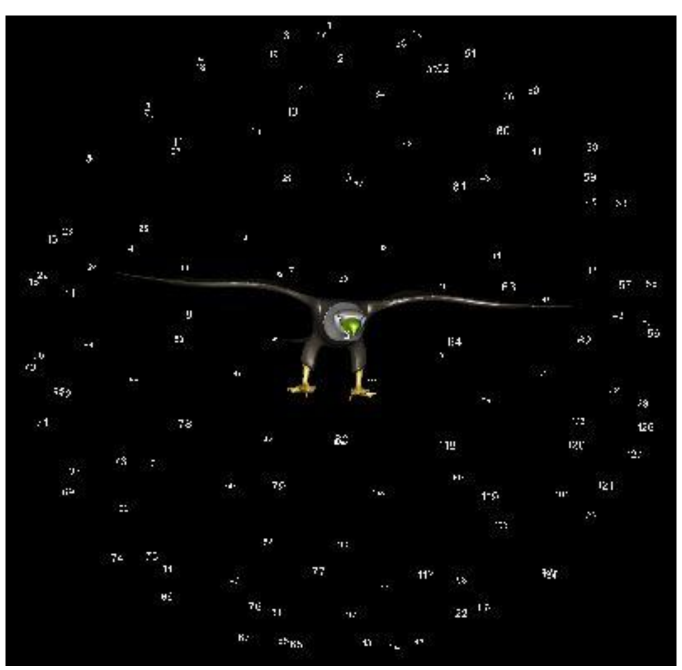
Bounded approximation algorithms

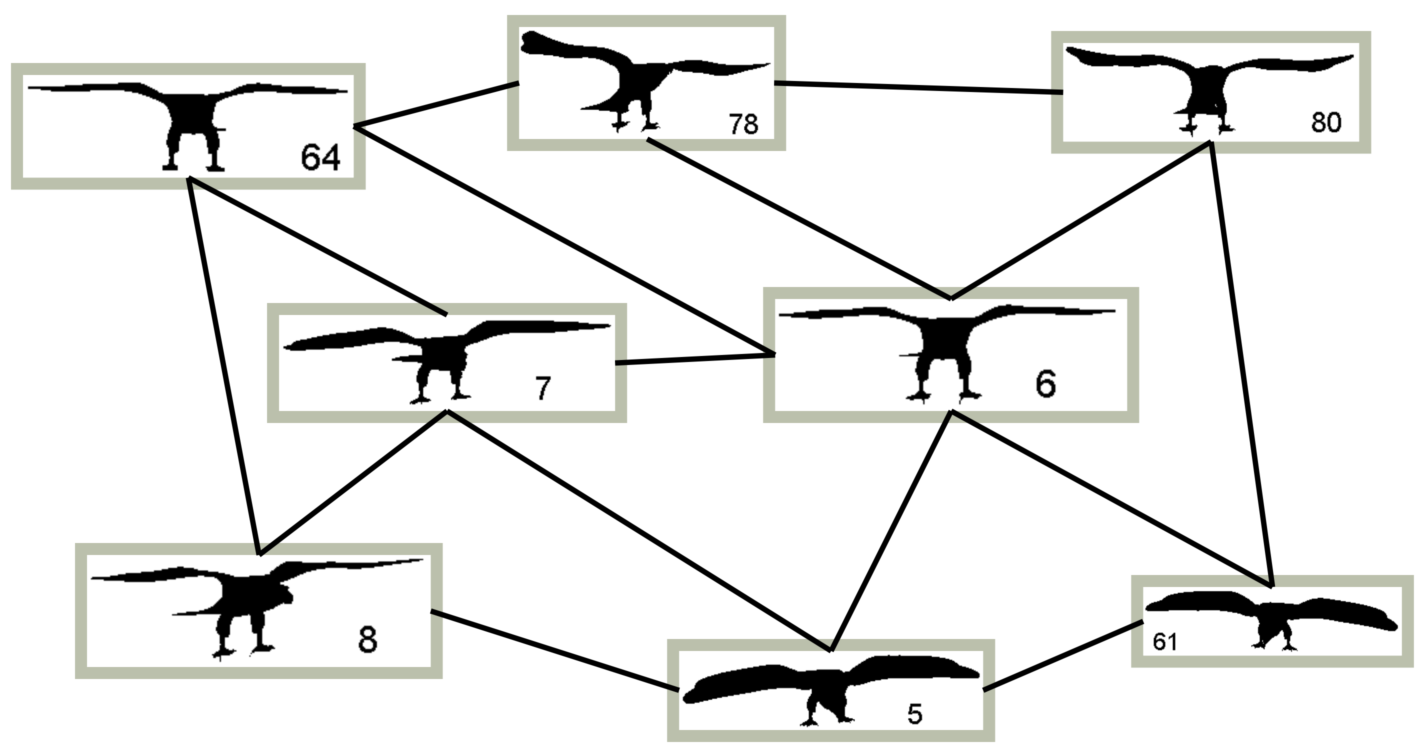
- Example: Vertex cover problem:
  - A ***vertex cover*** of an undirected graph $\boldsymbol{G}=(\boldsymbol{V},\boldsymbol{E})$ is a subset $\boldsymbol{V'}$ of $\boldsymbol{V}$ such that if $(\boldsymbol{u},\boldsymbol{v})$ is an edge in $\boldsymbol{E}$, then $\boldsymbol{u}$ or $\boldsymbol{v}$ (or both) belong to $\boldsymbol{V'}$.
  - **The *vertex cover problem* is to find a vertex cover of minimum size in a given undirected graph.**

Algorithmic Graph Theory:

Bounded approximation algorithms

- Example: Vertex cover problem:
  - A ***vertex cover*** of an undirected graph ***G***=(***V,E***) is a subset ***V'*** of ***V*** such that if (***u,v***) is an edge in ***E***, then ***u*** or ***v*** (or both) belong to ***V'***.
  - The ***vertex cover problem*** is to find a vertex cover of **minimum** size in a given undirected graph.

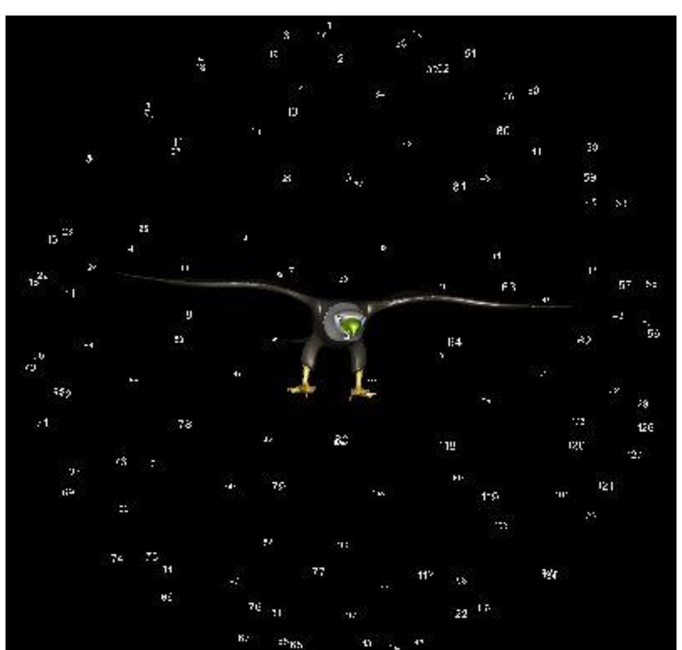


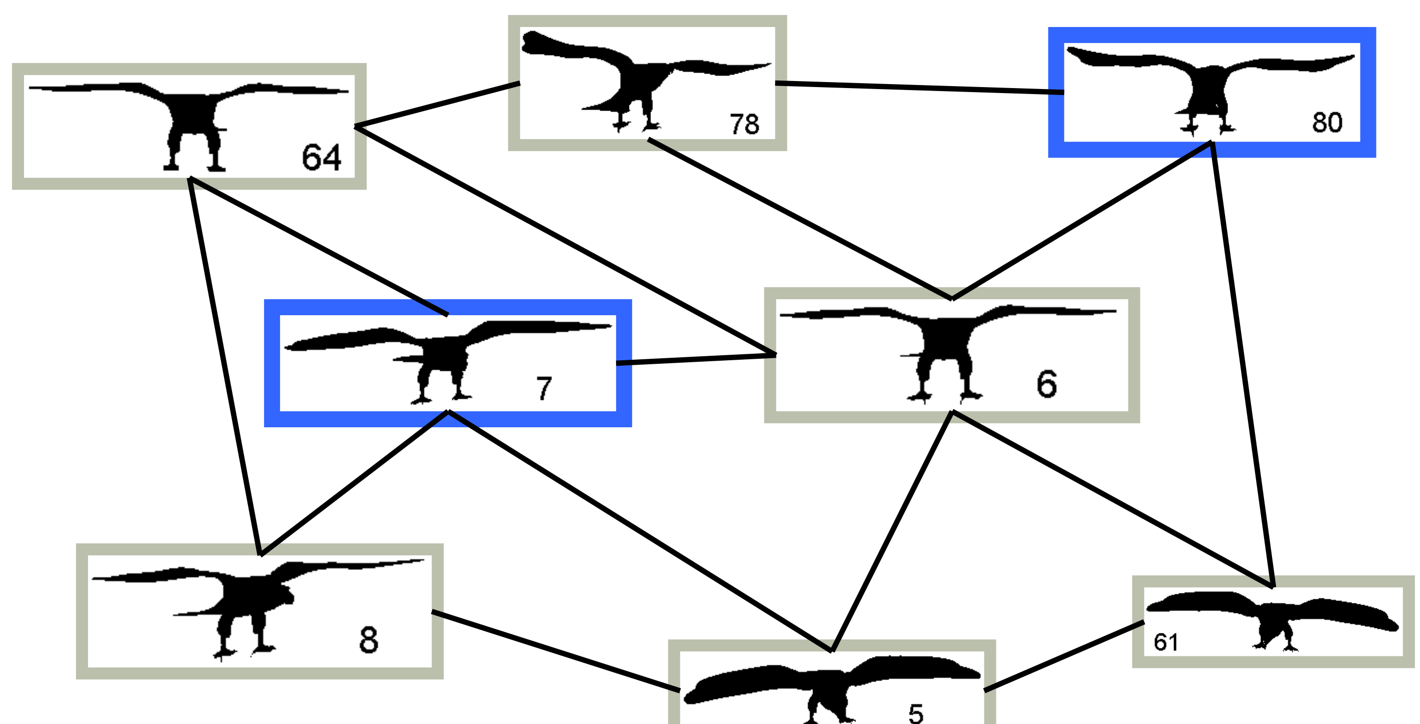


Algorithmic Graph Theory:

Bounded approximation algorithms

- Example: Vertex cover problem:
  - A ***vertex cover*** of an undirected graph $\boldsymbol{G}=(\boldsymbol{V},\boldsymbol{E})$ is a subset $\boldsymbol{V'}$ of $\boldsymbol{V}$ such that if $(\boldsymbol{u},\boldsymbol{v})$ is an edge in $\boldsymbol{E}$, then $\boldsymbol{u}$ or $\boldsymbol{v}$ (or both) belong to $\boldsymbol{V'}$.
  - The ***vertex cover problem*** is to find a vertex cover of **minimum** size in a given undirected graph.





Algorithmic Graph Theory:

Bounded approximation algorithms

- Example: Vertex cover problem:
  - A ***vertex cover*** of an undirected graph ***G***=(***V,E***) is a subset ***V'*** of ***V*** such that if (***u,v***) is an edge in ***E***, then ***u*** or ***v*** (or both) belong to ***V'***.
  - The size of a vertex cover is the number of vertices in it.
  - The ***vertex cover problem*** is to find a vertex cover of **minimum** size in a given undirected graph.
  - **We call such a vertex cover an *optimal vertex cover*.**
  - **The vertex cover problem was shown to be NP-complete.**

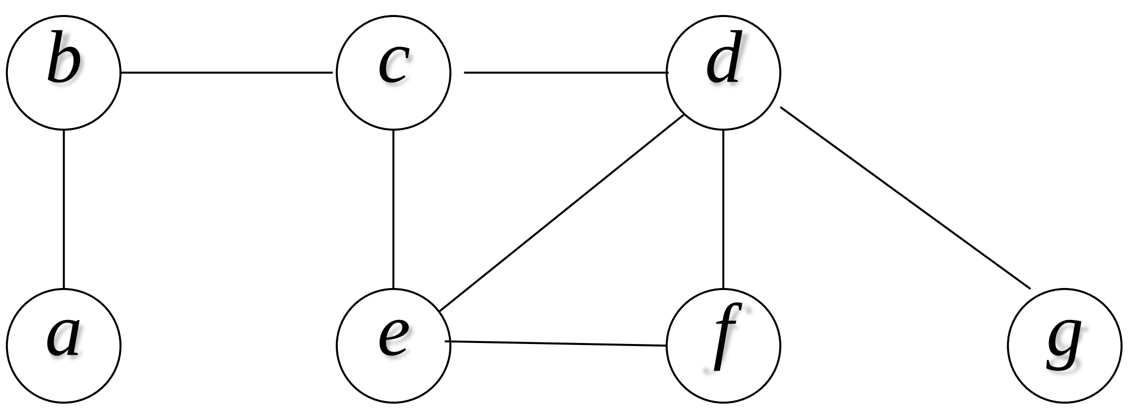
Algorithmic Graph Theory:

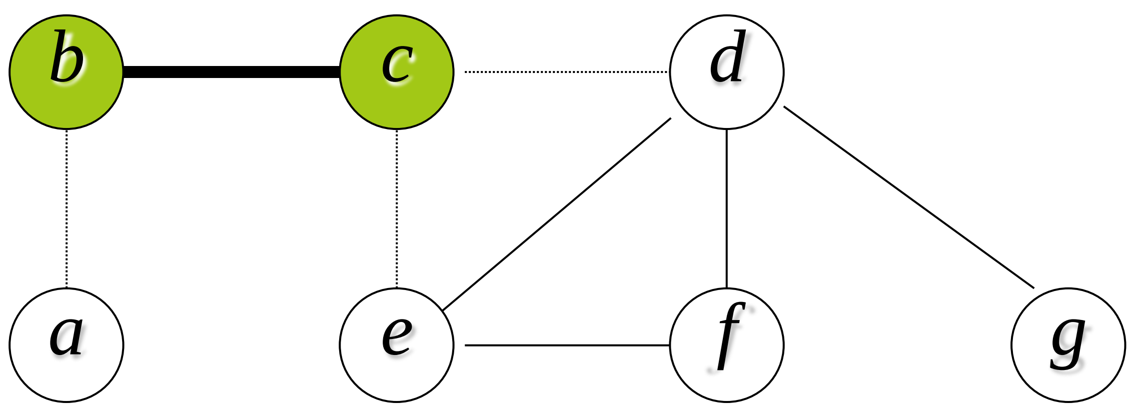
Vertex cover problem:

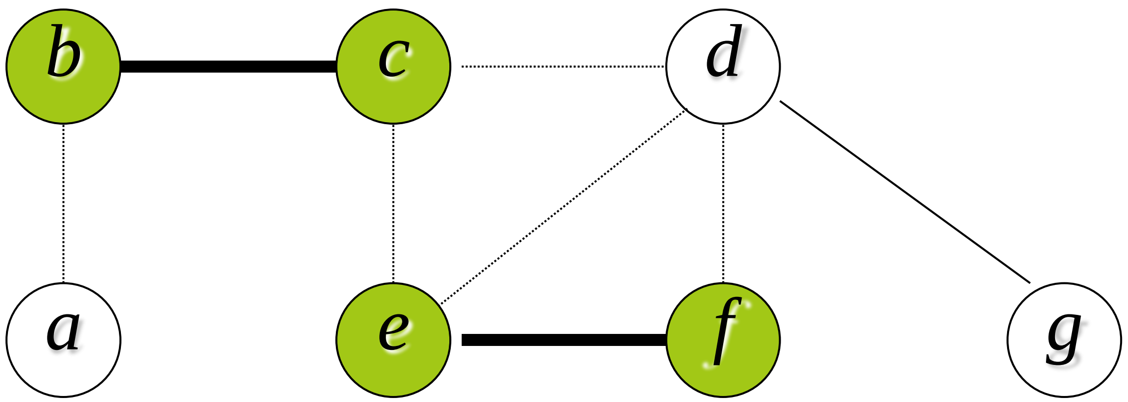
- The following approximation algorithm takes as input an undirected graph G and returns a vertex cover whose size is guaranteed no more than twice the size of optimal vertex cover:

1. $C \leftarrow \emptyset$
2. $E' \leftarrow E[G]$
3. While $E' \neq \emptyset$ do
4. Let (u, v) be an arbitrary edge in E'
5. $C \leftarrow C \cup \{u, v\}$
6. Remove from E' every edge incident on either u or v
7. Return C

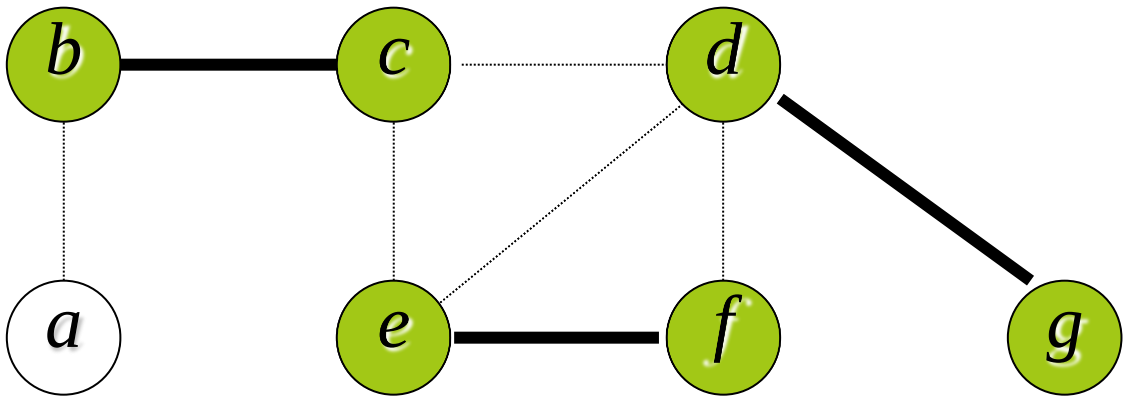
Algorithmic Graph Theory:

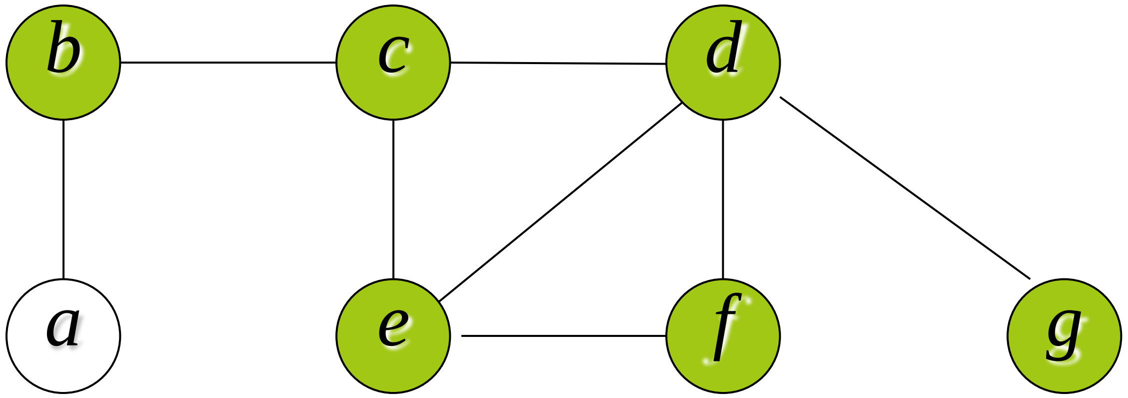




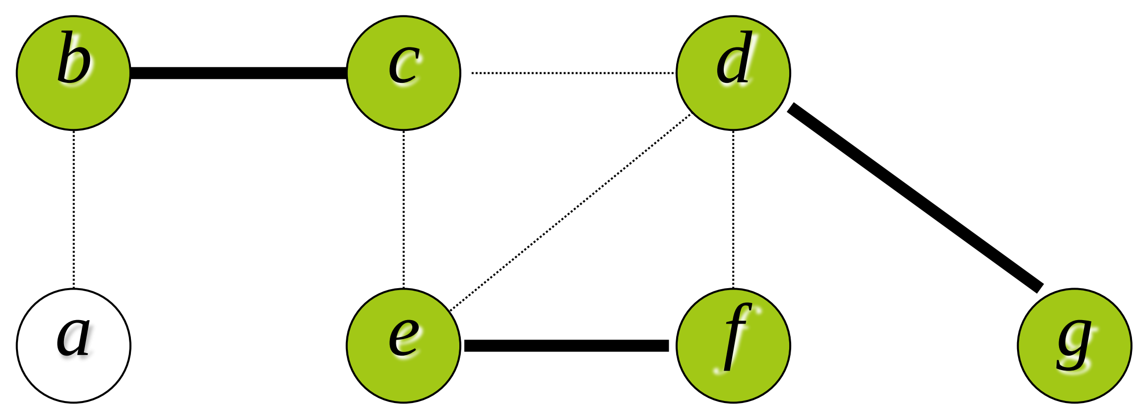


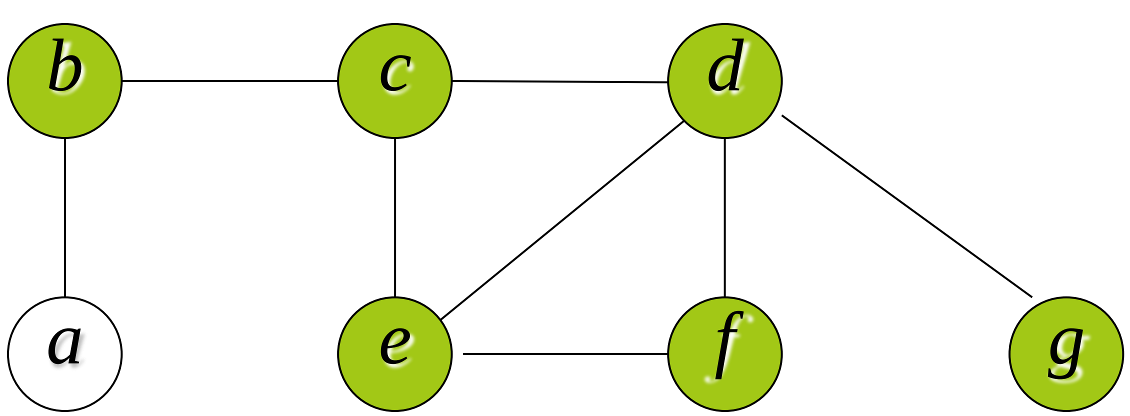
The Vertex Cover Problem

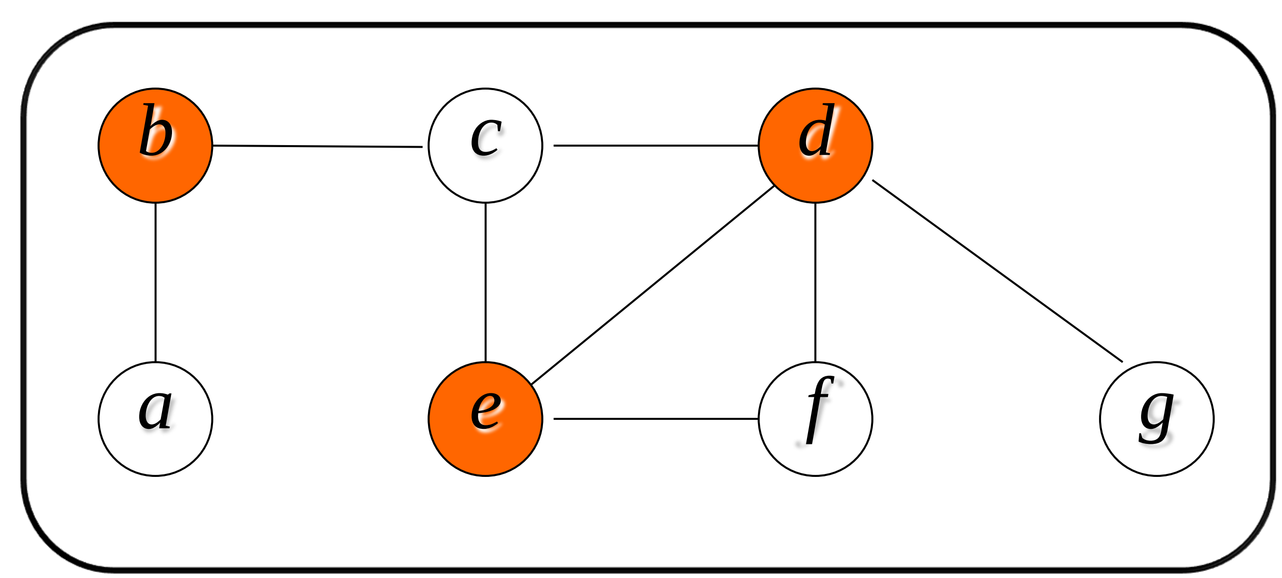




The Vertex Cover Problem







Algorithmic Graph Theory:

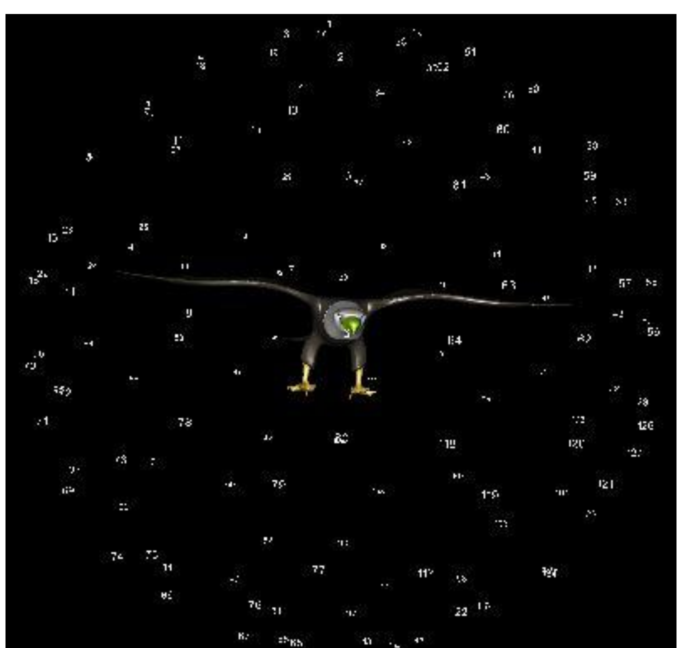
Theorem: Approximate vertex cover has a ratio bound of 2.

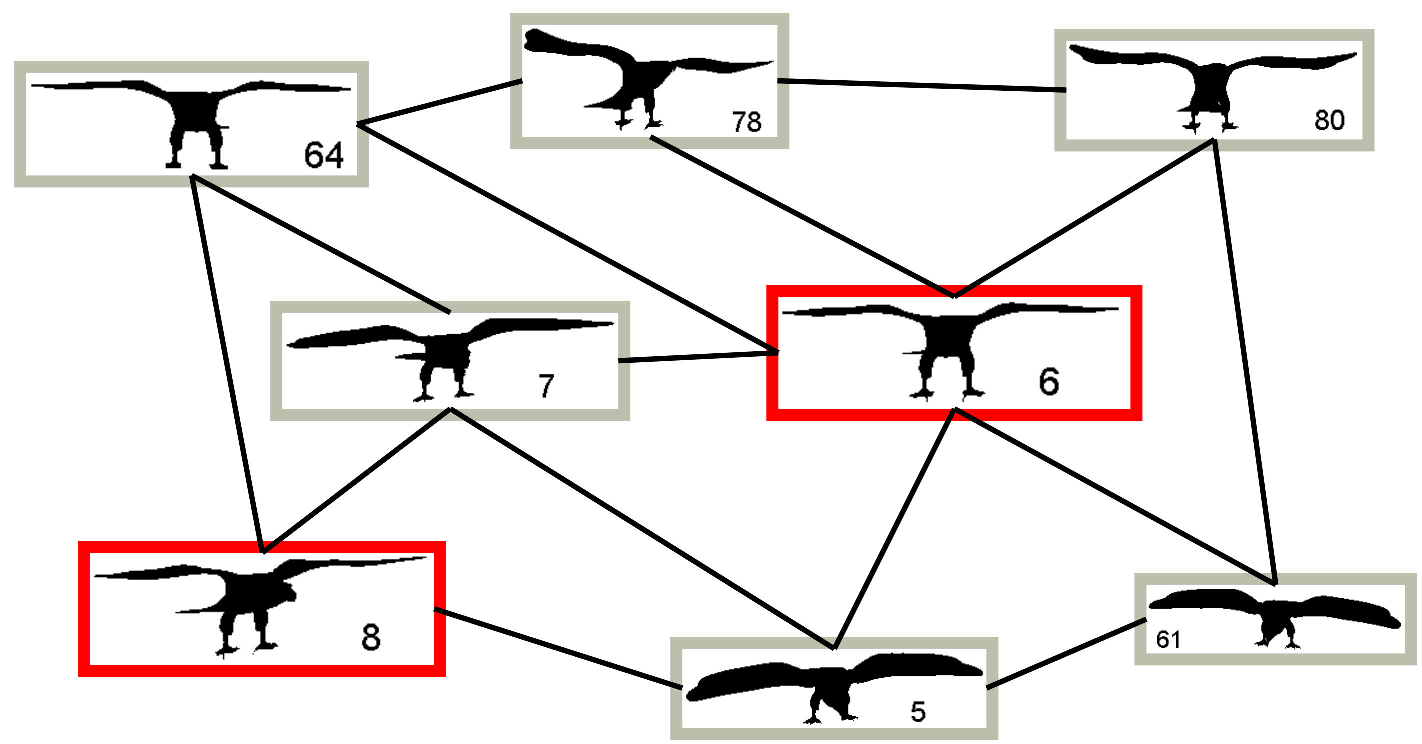
- ***Proof***:
  - It is easy to see that ***C*** is a vertex cover.
  - To show that the size of ***C*** is twice the size of optimal vertex cover.
  - Let ***A*** be the set of edges picked in line 4 of algorithm.
  - No two edges in ***A*** share an endpoint, therefore each new edge adds two new vertices to ***C***, so |***C***|=***2***|***A***|.
  - Any vertex cover should cover the edges in ***A***, which means at least one of the end points of each edge in *A* belongs to ***C*.***
  - So, |***A***|<=|***C****|, which will imply the desired bound.

Algorithmic Graph Theory:

Bounded approximation algorithms

- Example: Vertex cover problem:
  - A ***vertex cover*** of an undirected graph $\boldsymbol{G}=(\boldsymbol{V},\boldsymbol{E})$ is a subset $\boldsymbol{V'}$ of $\boldsymbol{V}$ such that if $(\boldsymbol{u},\boldsymbol{v})$ is an edge in $\boldsymbol{E}$, then $\boldsymbol{u}$ or $\boldsymbol{v}$ (or both) belong to $\boldsymbol{V'}$.
  - The ***vertex cover problem*** is to find a vertex cover of **minimum** size in a given undirected graph.





What Next?

Geometry of Graphs and Graphs Encoding the Geometry

Spectral Graph Theory