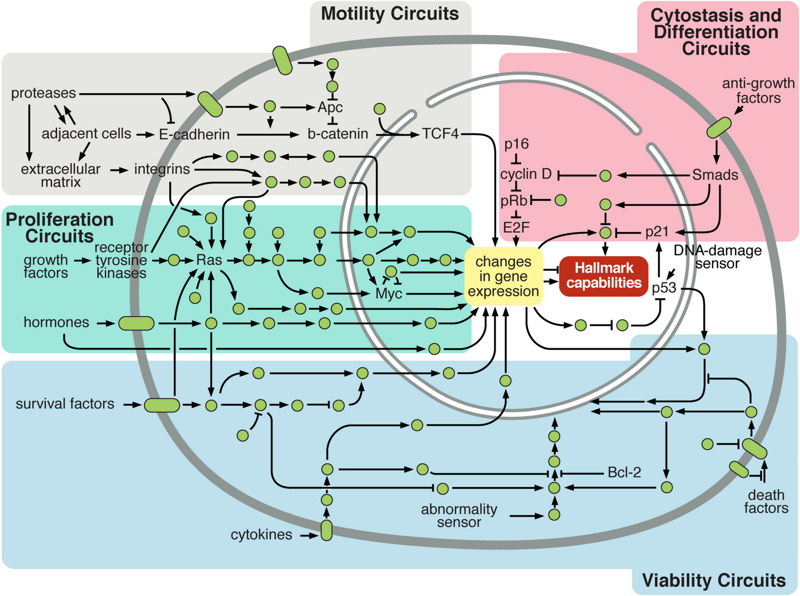
CS 3824: Network Modules

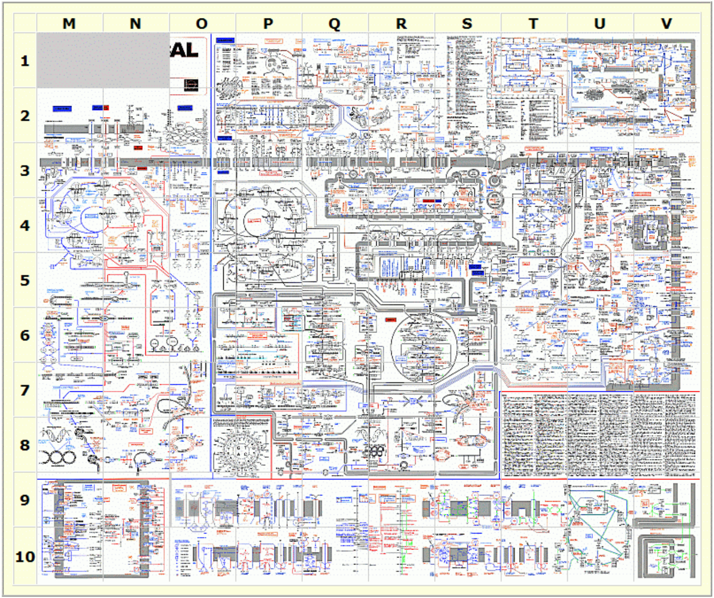
T. M. Murali

September 15, 20, and 27, 2022

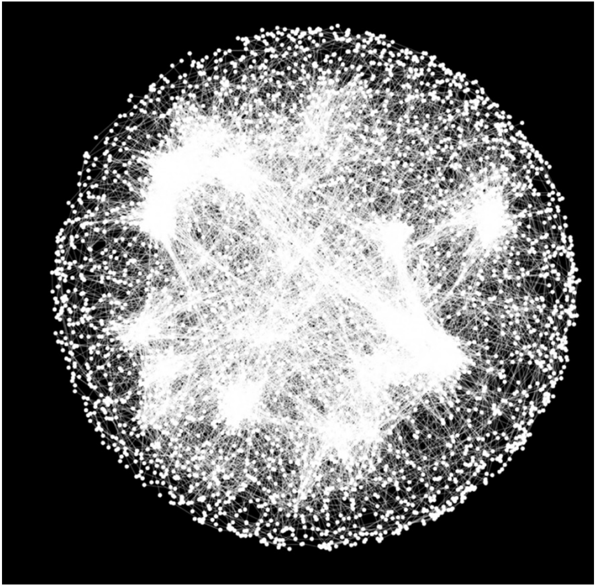
Network is Complex



Network is Complex

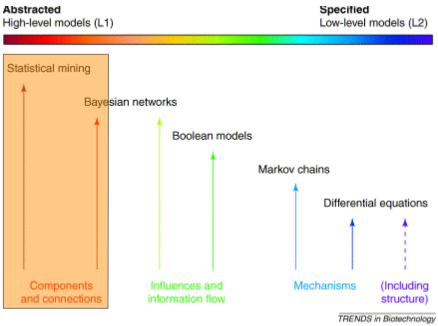


Network is Complex but Very Poorly Understood



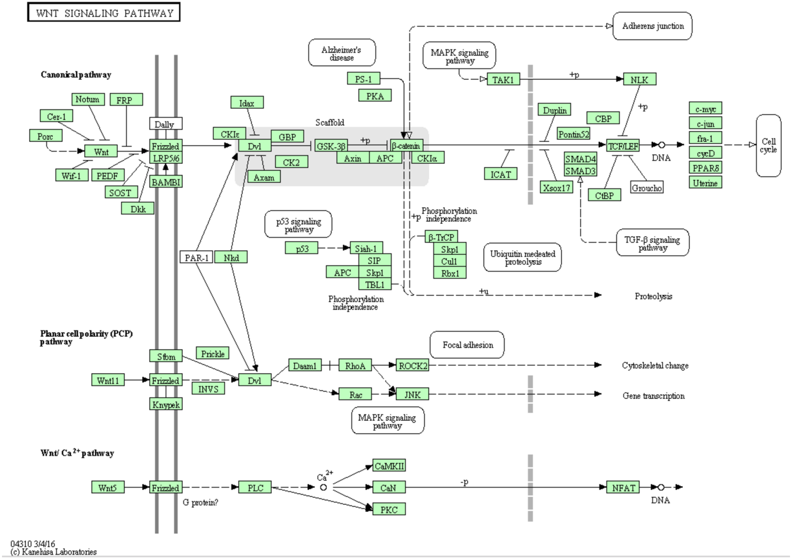
Costanzo et al., Cell, 2019.

Goals of the Course



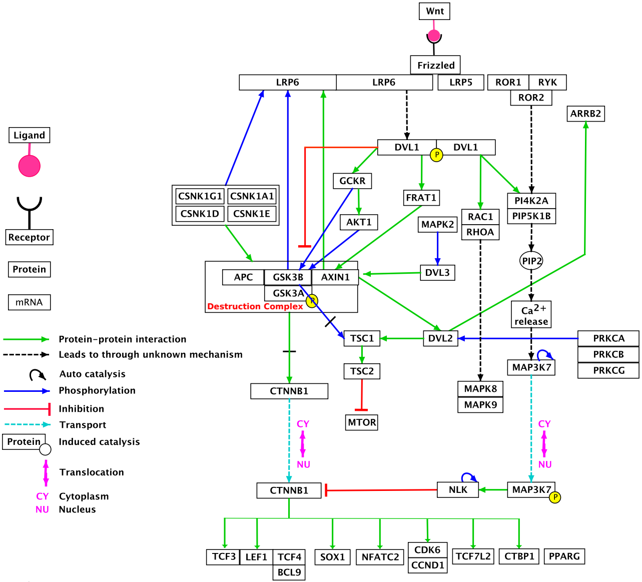
- Emphasise a data-driven approach to biology.
- Take a network-level view of cellular processes.
- Abstract biological questions into computer science problems.
- Describe graph algorithms to solve these problems.

Wnt Signaling in a Pathway Database



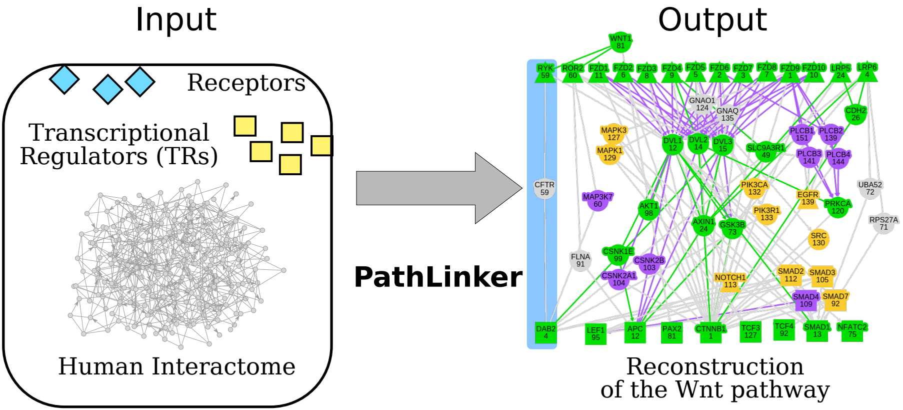
KEGG database

Wnt Signaling in a Pathway Database



www.netpath.org/netslim

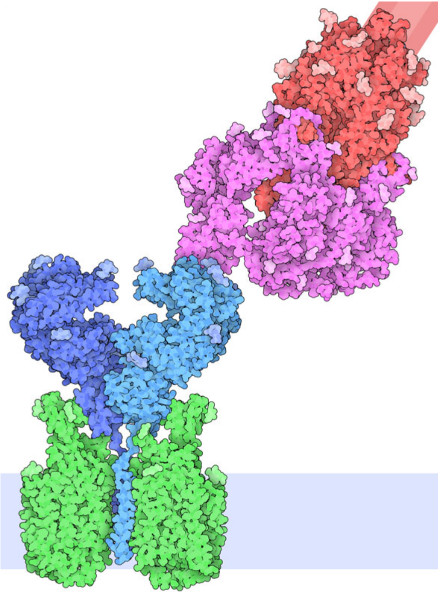
Automated Reconstruction of Signaling Pathways



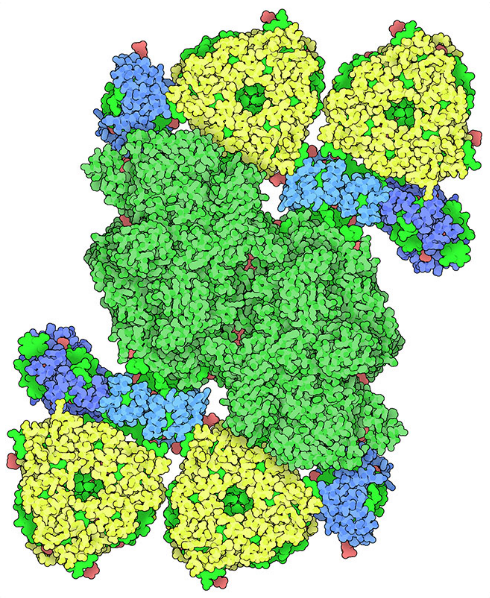
- PATHLINKER and other algorithms can automatically reconstruct signaling pathways.

"Pathways on Demand: Automatic Reconstruction of Human Signaling Pathways," Ritz et al., Systems Biology and Applications, a Nature partner journal, 2, 16002, 2016.

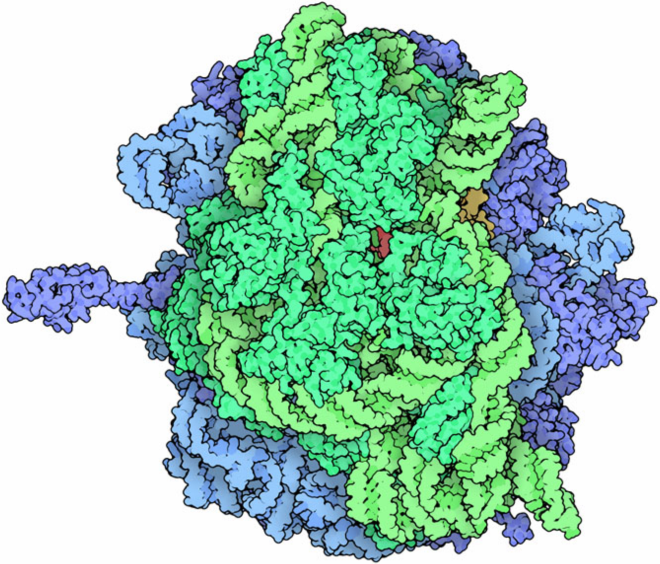
Protein Complexes



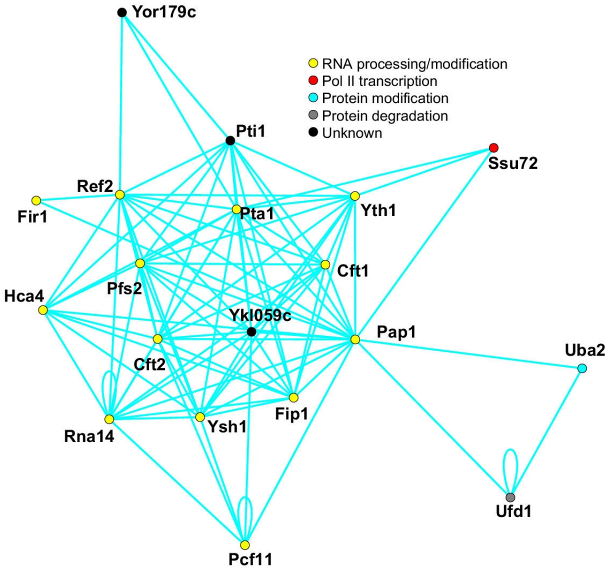
Protein Complexes



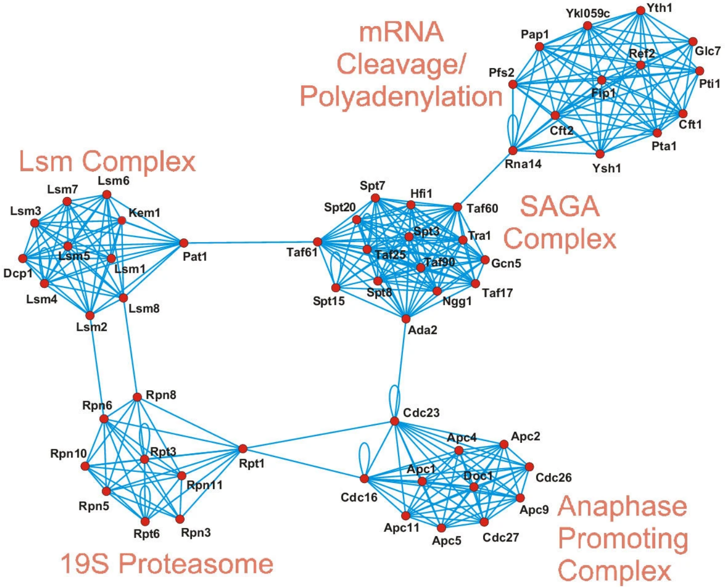
Protein Complexes



Protein Complexes

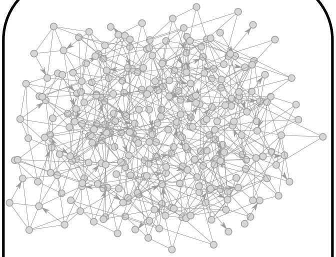


Protein Complexes



Problem Formulation

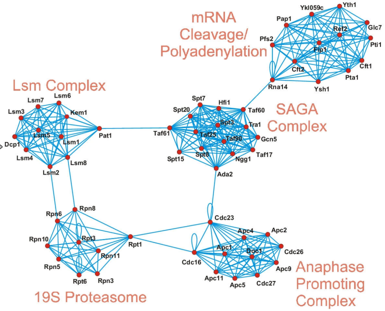
Input



Human Interactome

Module finding algorithm

Output

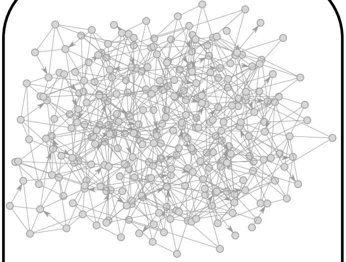


Set of modules

- Given a protein interaction network $G = (V, E, W)$, compute the modules (clusters) in it.

Problem Formulation

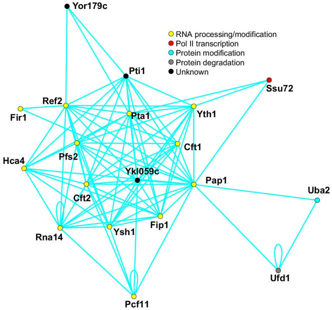
Input



Human Interactome

Module finding algorithm

Output

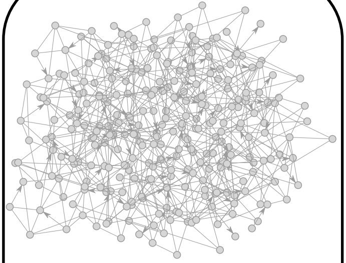


One module

- Given a protein interaction network $G = (V, E, W)$, compute the modules (clusters) in it.
- Given a protein interaction network $G = (V, E, W)$, compute one module in it.

Problem Formulation

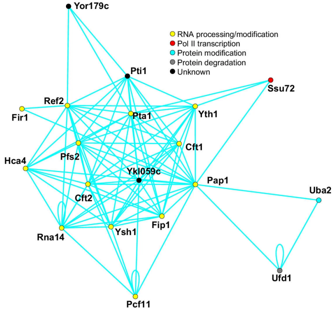
Input



Human Interactome

Module finding algorithm

Output



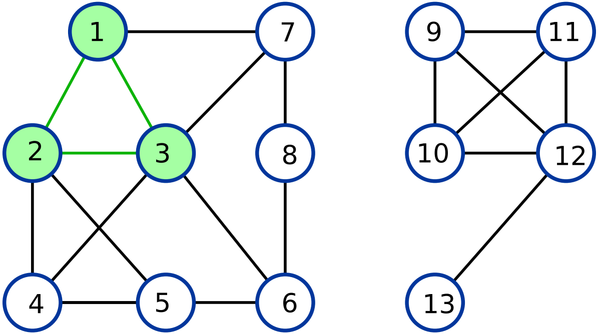
One module

- Given a protein interaction network $G = (V, E, W)$, compute the modules (clusters) in it.
- Given a protein interaction network $G = (V, E, W)$, compute one module in it.
- First, define what we mean by a module. Then, develop algorithm to compute one or more modules.

Modules and Clustering

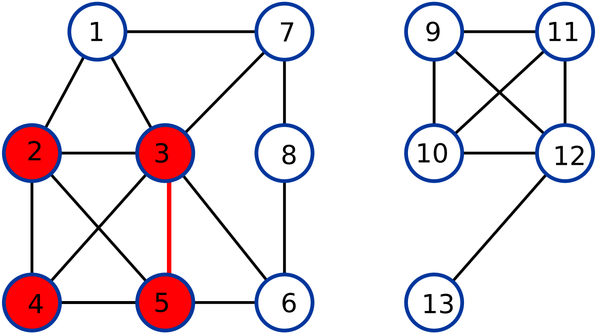
- Finding modules or clusters formed by a set of objects is a widely studied problem.
- Long history in mathematics, statistics, and computer science.
- Module $\equiv$ Cluster $\equiv$ Community.

Defining Modules



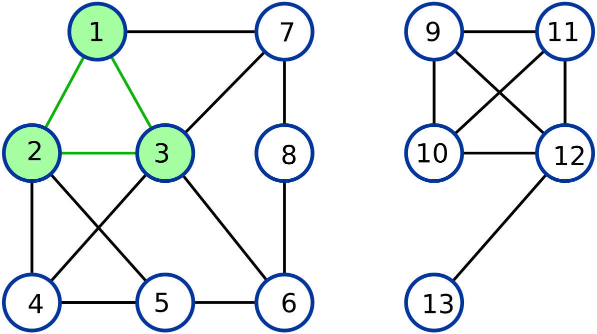
- How do we define a module in an undirected graph?
- In an undirected graph $G = (V, E)$, a subset of nodes $C \subseteq V$ is a *clique* or *complete subgraph* if for every pair of nodes $u, v \in C$, (u, v) is an edge in E.

Defining Modules



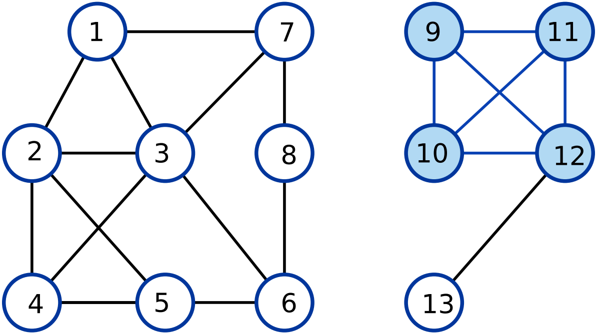
- How do we define a module in an undirected graph?
- In an undirected graph $G = (V, E)$, a subset of nodes $C \subseteq V$ is a *clique* or *complete subgraph* if for every pair of nodes $u, v \in C$, (u, v) is an edge in E.

Defining Modules



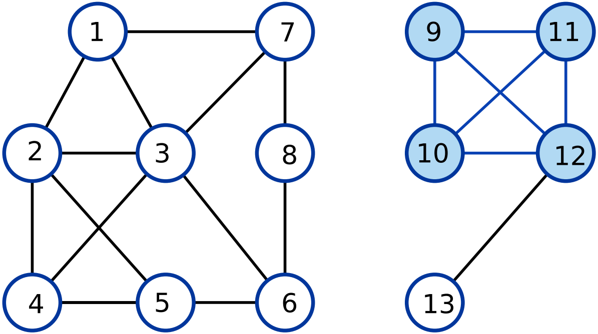
- How do we define a module in an undirected graph?
- In an undirected graph $G = (V, E)$, a subset of nodes $C \subseteq V$ is a *clique* or *complete subgraph* if for every pair of nodes $u, v \in C$, (u, v) is an edge in E.
  - A clique C is *maximal* if no node outside C can be added to it, i.e., for every node $x \in V - C$, x is not connected to at least one node in C.

Defining Modules



- How do we define a module in an undirected graph?
- In an undirected graph $G = (V, E)$, a subset of nodes $C \subseteq V$ is a *clique* or *complete subgraph* if for every pair of nodes $u, v \in C$, (u, v) is an edge in E.
  - A clique C is *maximal* if no node outside C can be added to it, i.e., for every node $x \in V - C$, x is not connected to at least one node in C.
  - A clique C is *maximum* if there is no clique C' in G with more nodes than C.

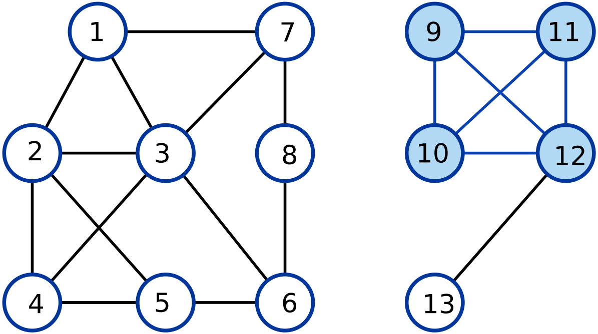
Computing a Maximum Clique



Maximum Clique

Given an undirected, unweighted graph $G(V, E)$, compute the largest clique in G.

Computing a Maximum Clique

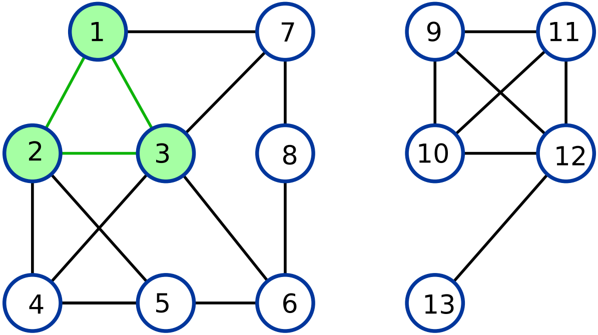


MAXIMUM CLIQUE

Given an undirected, unweighted graph $G(V, E)$, compute the largest clique in G.

- Computing a maximum clique is NP-hard.
- Any algorithm that can provably compute the maximum clique is likely to have a running time that is exponential in the size of the graph.

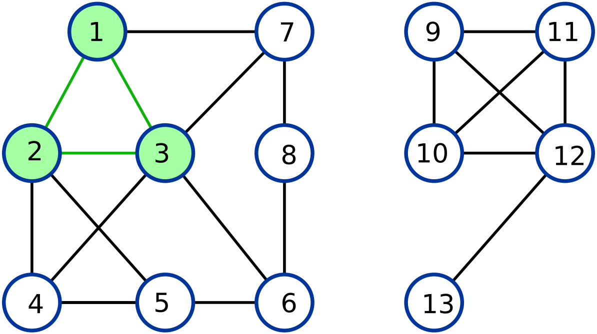
Computing a Maximal Clique



MAXIMAL CLIQUE

Given an undirected, unweighted graph $G(V, E)$, compute a maximal clique in G.

Computing a Maximal Clique

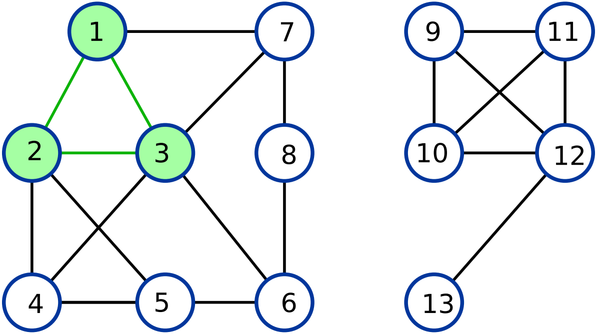


MAXIMAL CLIQUE

Given an undirected, unweighted graph $G(V, E)$, compute a maximal clique in G.

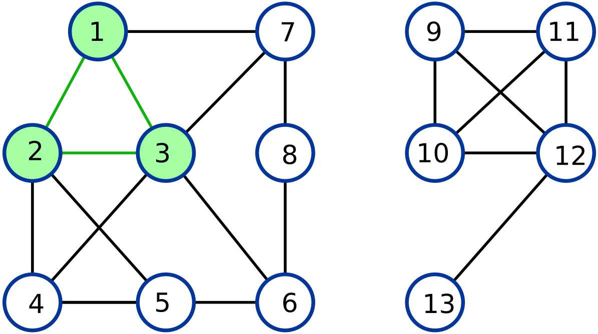
1. Select an arbitrary node v and add it to S (the clique we will output).
2. If there is a node u in $V - S$ that is connected to every node in S, add u to S.
3. Repeat the previous step until no such node u is found.

Running Time to Compute a Maximal Clique



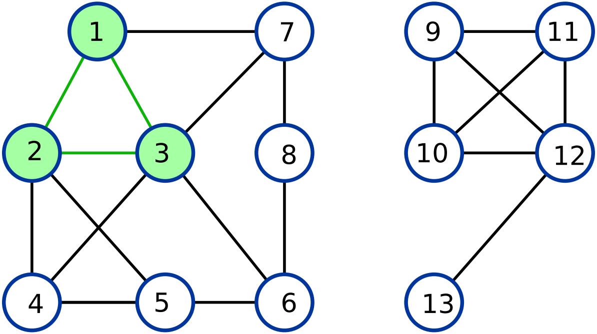
1. Select an arbitrary node v and add it to S (the clique we will output).
2. If there is a node u in $V - S$ that is connected to every node in S, add u to S.
3. Repeat the previous step until no such node u is found.

Running Time to Compute a Maximal Clique



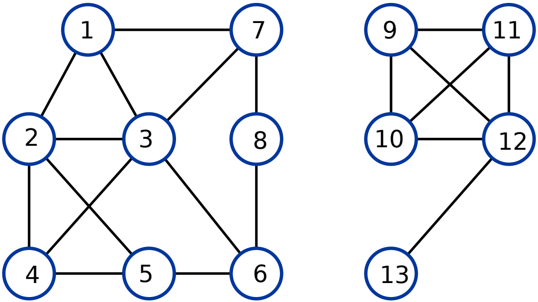
1. Select an arbitrary node v and add it to S (the clique we will output).
2. If there is a node u in $V - S$ that is connected to every node in S, add u to S. $O(n|S|)$ checks for edge existence.
3. Repeat the previous step until no such node u is found.

Running Time to Compute a Maximal Clique



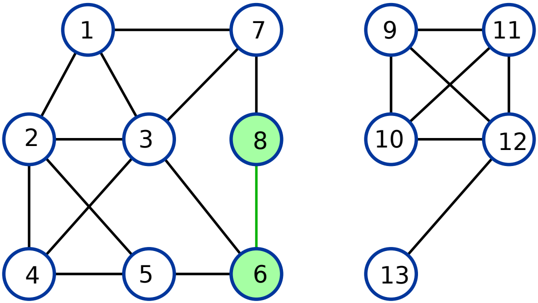
1. Select an arbitrary node v and add it to S (the clique we will output).
2. If there is a node u in $V - S$ that is connected to every node in S, add u to S. $O(n|S|)$ checks for edge existence.
3. Repeat the previous step until no such node u is found. $O(n|S|^2)$ checks for edge existence.

k-Cores



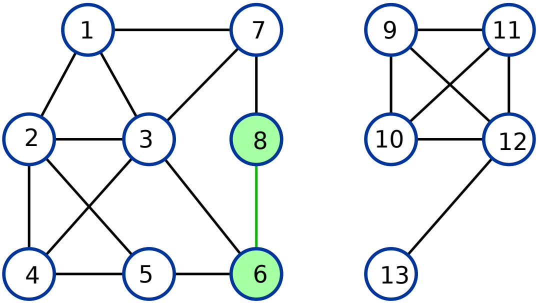
- In an undirected graph $G = (V, E)$, a subset of nodes $C \subseteq V$ is a *k-core* if every node $u \in C$ is connected in G to at least k nodes in C.

k-Cores



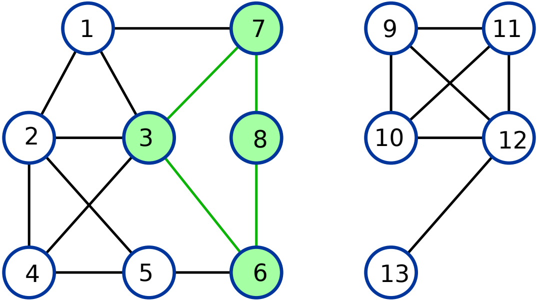
- In an undirected graph $G = (V, E)$, a subset of nodes $C \subseteq V$ is a *k-core* if every node $u \in C$ is connected in G to at least k nodes in C.
- What is largest the 1-core of G?

k-Cores



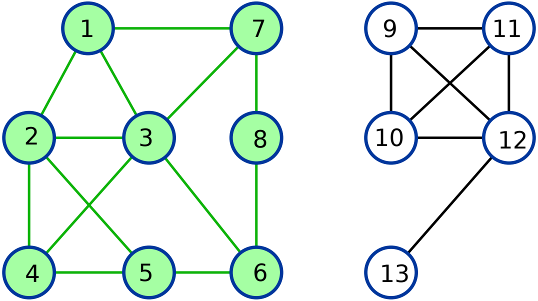
- In an undirected graph $G = (V, E)$, a subset of nodes $C \subseteq V$ is a *k-core* if every node $u \in C$ is connected in G to at least k nodes in C.
- What is largest the 1-core of G? G itself (without any nodes of degree zero).

k-Cores



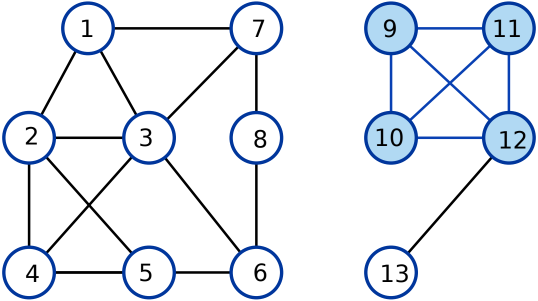
- In an undirected graph $G = (V, E)$, a subset of nodes $C \subseteq V$ is a *k-core* if every node $u \in C$ is connected in G to at least k nodes in C.
- What is largest the 1-core of G? G itself (without any nodes of degree zero).

k-Cores



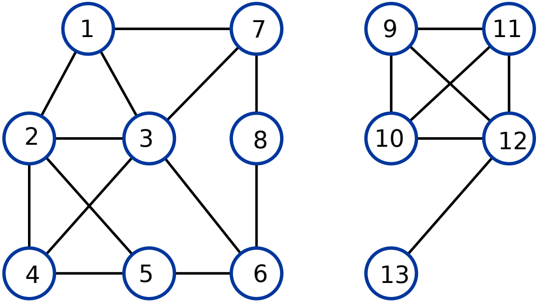
- In an undirected graph $G = (V, E)$, a subset of nodes $C \subseteq V$ is a *k-core* if every node $u \in C$ is connected in G to at least k nodes in C.
- What is largest the 1-core of G? G itself (without any nodes of degree zero).

k-Cores



- In an undirected graph $G = (V, E)$, a subset of nodes $C \subseteq V$ is a *k-core* if every node $u \in C$ is connected in G to at least k nodes in C.
- What is largest the 1-core of G? G itself (without any nodes of degree zero).
- Does this graph have a 4-core?

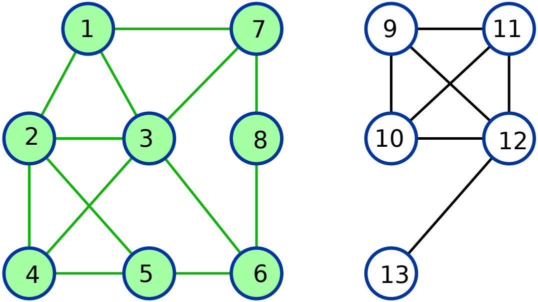
Problems related to k-cores



k-CORE EXISTENCE

Given an undirected, unweighted graph $G(V, E)$ and an integer k, compute the k-core with the largest number of nodes in G, if it exists.

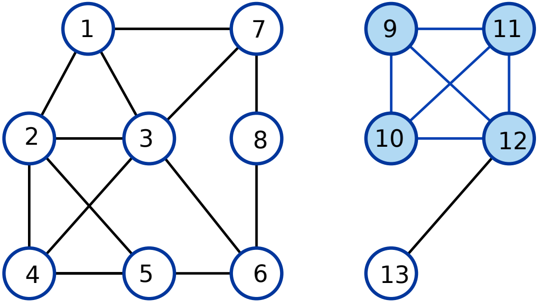
Problems related to k-cores



k-core Existence

Given an undirected, unweighted graph $G(V, E)$ and an integer k, compute the k-core with the largest number of nodes in G, if it exists.

Problems related to k-cores



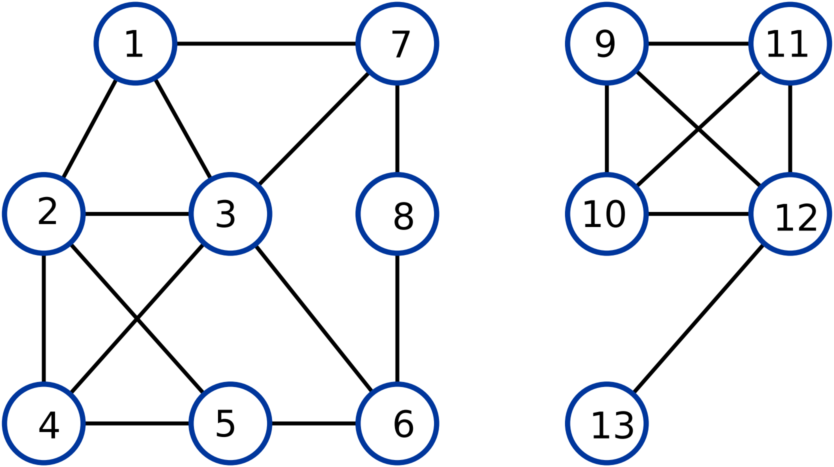
k-CORE EXISTENCE

Given an undirected, unweighted graph $G(V, E)$ and an integer k, compute the k-core with the largest number of nodes in G, if it exists.

LARGEST k-CORE

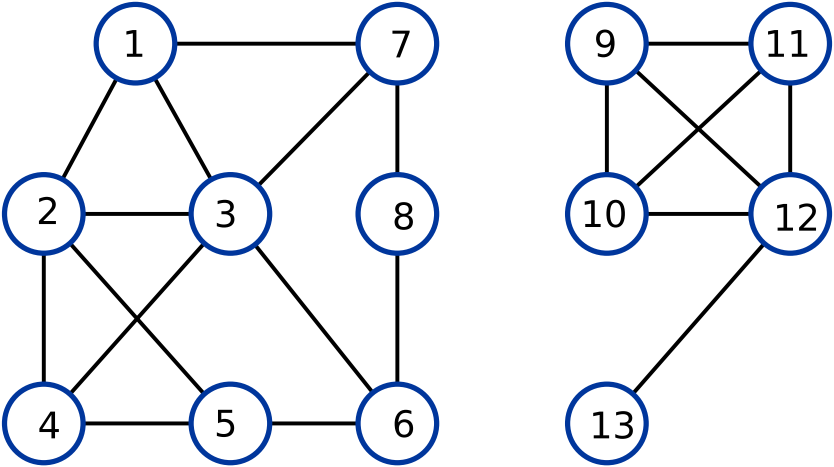
Given an undirected, unweighted graph $G(V, E)$,
compute the largest value of k for which G contains a k-core.

Algorithm for k-Core Existence



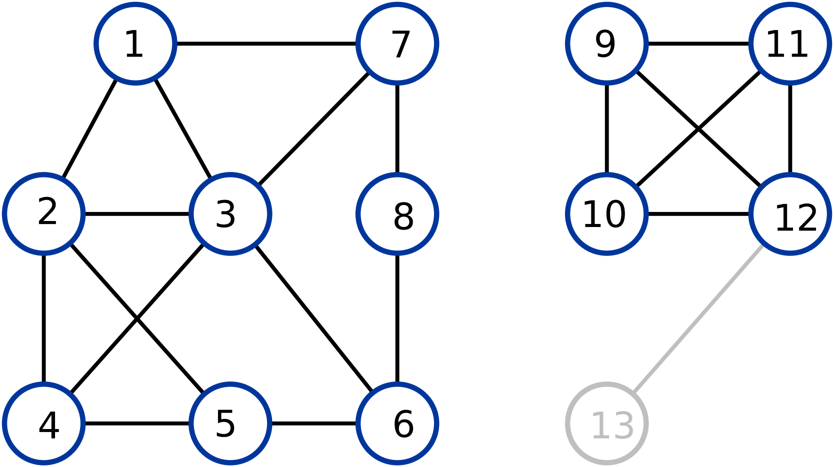
- Repeatedly delete all nodes of degree $< k$ until

Algorithm for k-Core Existence



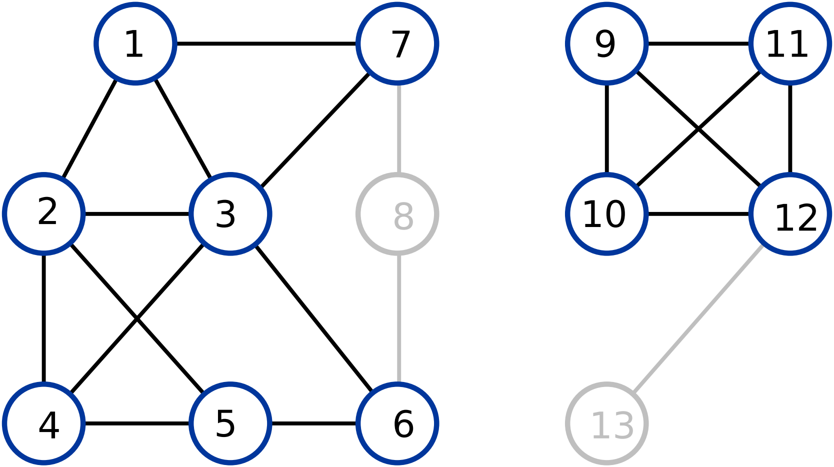
- Repeatedly delete all nodes of degree $< k$ until every remaining node has degree $\geq k$.
- Resulting graph is the largest k-core.

Algorithm for k-Core Existence



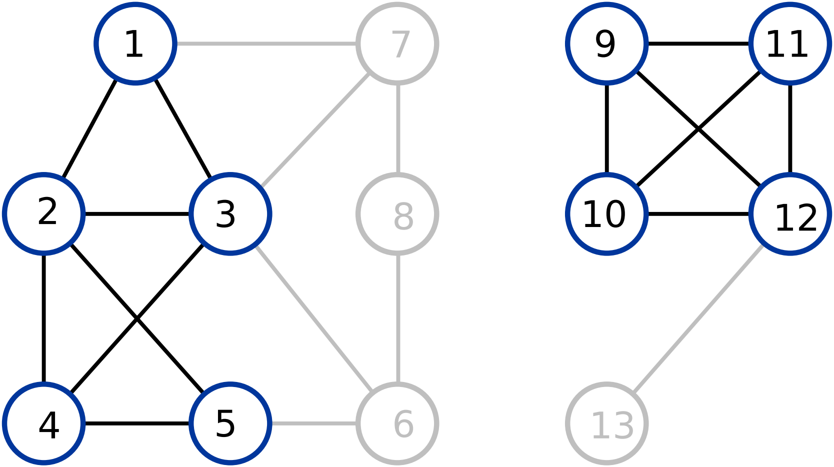
- Repeatedly delete all nodes of degree $< k$ until every remaining node has degree $\geq k$.
- Resulting graph is the largest k-core.

Algorithm for k-Core Existence



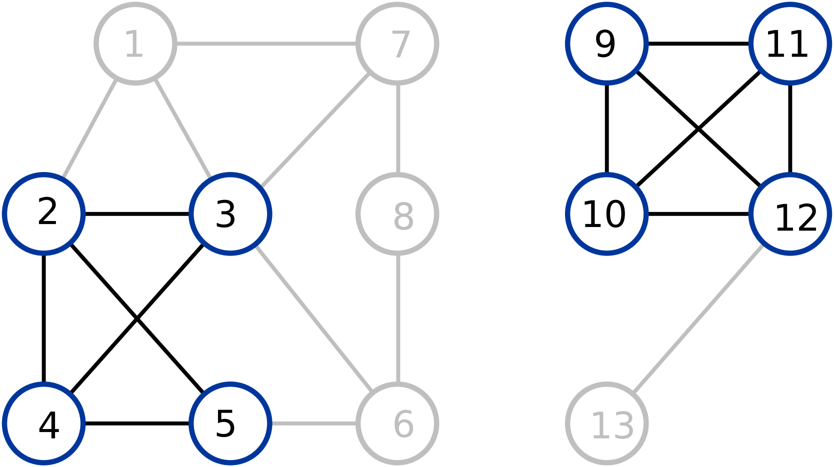
- Repeatedly delete all nodes of degree $< k$ until every remaining node has degree $\geq k$.
- Resulting graph is the largest k-core.

Algorithm for k-Core Existence



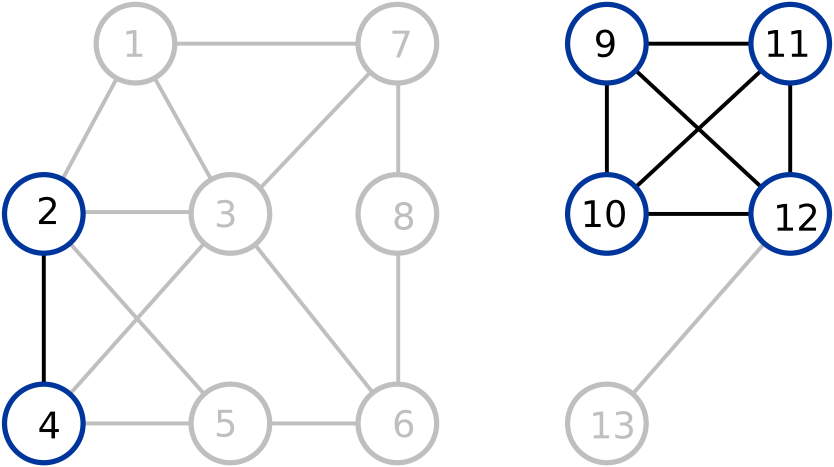
- Repeatedly delete all nodes of degree $< k$ until every remaining node has degree $\geq k$.
- Resulting graph is the largest k-core.

Algorithm for k-Core Existence



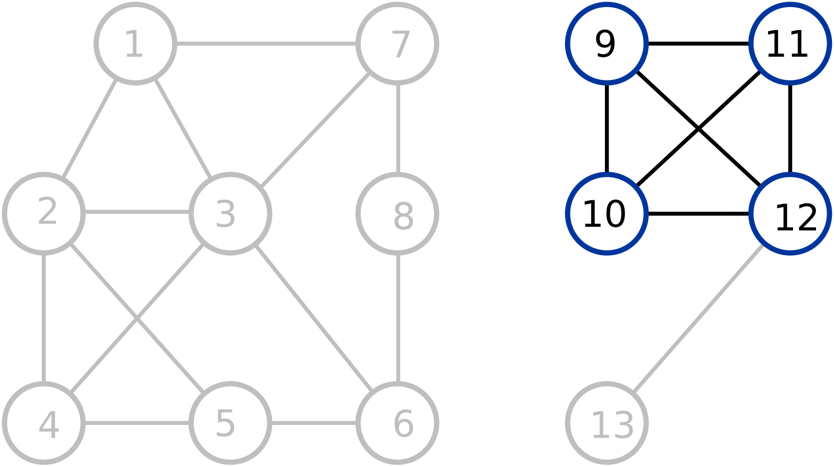
- Repeatedly delete all nodes of degree $< k$ until every remaining node has degree $\geq k$.
- Resulting graph is the largest k-core.

Algorithm for k-Core Existence



- Repeatedly delete all nodes of degree $< k$ until every remaining node has degree $\geq k$.
- Resulting graph is the largest k-core.

Algorithm for k-Core Existence



- Repeatedly delete all nodes of degree $< k$ until every remaining node has degree $\geq k$.
- Resulting graph is the largest k-core.

Correctness of k-Core Existence Algorithm

- Repeatedly delete all nodes of degree $< k$ until every remaining node has degree $\geq k$.
- Why should the resulting graph H be a k-core?
- Why should the resulting graph H be the k-core with the largest number of nodes?

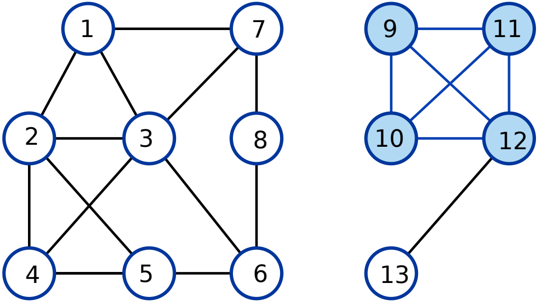
Correctness of k-Core Existence Algorithm

- Repeatedly delete all nodes of degree $< k$ until every remaining node has degree $\geq k$.
- Why should the resulting graph H be a k-core?
- Why should the resulting graph H be the k-core with the largest number of nodes?
- Proof by contradiction.
  - Suppose there is a k-core H' with more nodes than H.
  - Then $H \cup H'$ is also a k-core.
  - Moreover, no node in H' will be deleted by the algorithm.

Correctness of k-Core Existence Algorithm

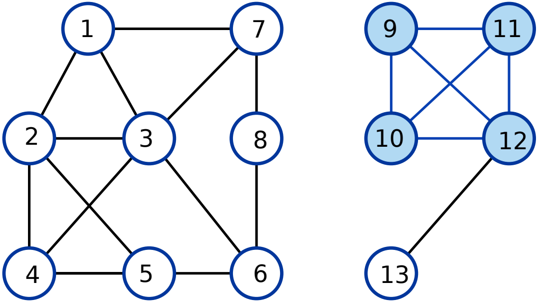
- Repeatedly delete all nodes of degree $< k$ until every remaining node has degree $\geq k$.
- Why should the resulting graph H be a k-core?
- Why should the resulting graph H be the k-core with the largest number of nodes?
- Proof by contradiction.
  - Suppose there is a k-core H' with more nodes than H.
  - Then $H \cup H'$ is also a k-core.
  - Moreover, no node in H' will be deleted by the algorithm.
- How do we implement k-core algorithm efficiently?

Cores vs. Cliques



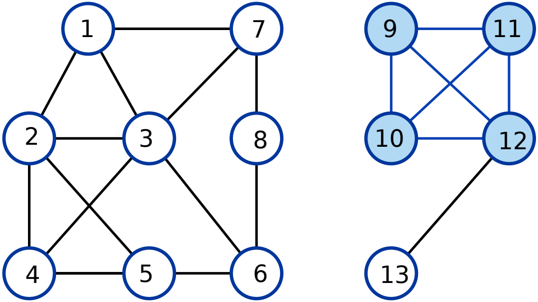
- A clique with k nodes is a $(k-1)$-core.
- Can we use the k-core algorithm to find maximum cliques?

Cores vs. Cliques



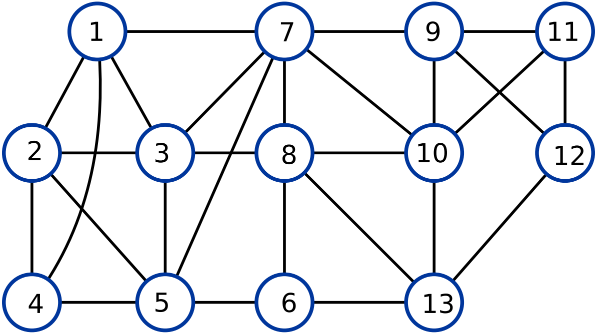
- A clique with k nodes is a $(k-1)$-core.
- Can we use the k-core algorithm to find maximum cliques?
- Idea: Compute the largest value of k for which a k-core H exists. If H is a clique, it must be the largest clique (of size $k+1$) in the graph.

Cores vs. Cliques



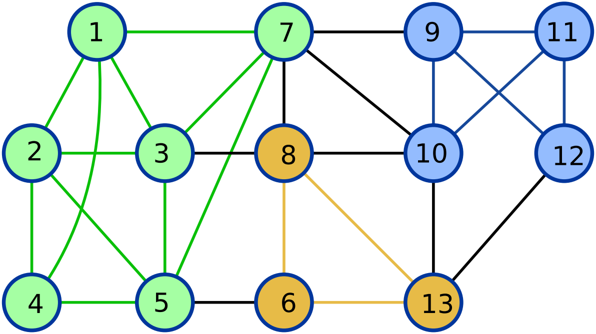
- A clique with k nodes is a $(k-1)$-core.
- Can we use the k-core algorithm to find maximum cliques?
- Idea: Compute the largest value of k for which a k-core H exists. If H is a clique, it must be the largest clique (of size $k+1$) in the graph.
- Flaw is that H may not be a clique, in general. The largest clique may be disjoint from H or be a subgraph of H.
- Moreover, the maximum clique may have l nodes while there may be a k-core where $k > l - 1$, e.g., $k = 3$ and $l = 3$. Create such an example.

Motivation



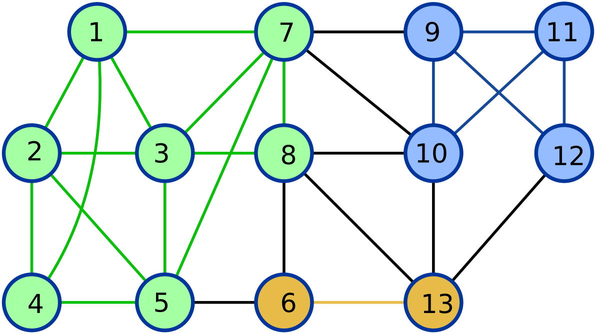
- Given an undirected, unweighted graph $G = (V, E)$ suppose we partition the nodes into k modules $\mathcal{C} = C_1, C_2, \ldots C_k$.
- How do we measure the "quality" of $\mathcal{C}$?
- Intuition: many more edges within modules than among modules.

Motivation



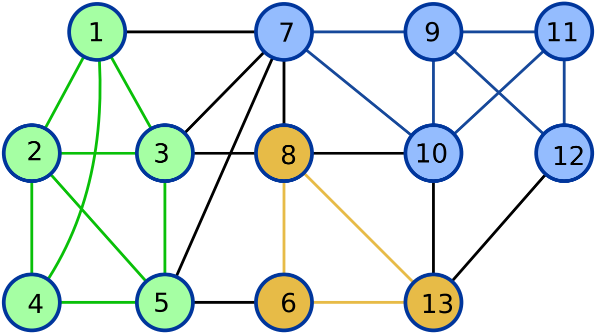
- Given an undirected, unweighted graph $G = (V, E)$ suppose we partition the nodes into k modules $\mathcal{C} = C_1, C_2, \ldots C_k$.
- How do we measure the "quality" of $\mathcal{C}$?
- Intuition: many more edges within modules than among modules.

Motivation



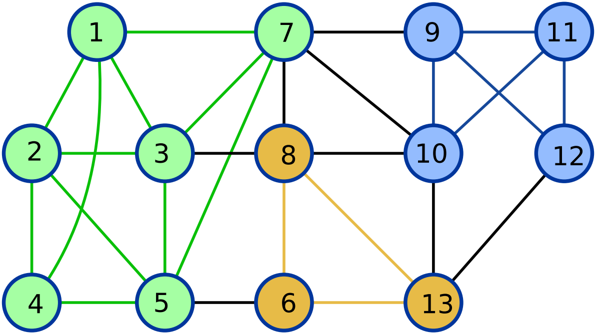
- Given an undirected, unweighted graph $G = (V, E)$ suppose we partition the nodes into k modules $\mathcal{C} = C_1, C_2, \ldots C_k$.
- How do we measure the "quality" of $\mathcal{C}$?
- Intuition: many more edges within modules than among modules.

Motivation



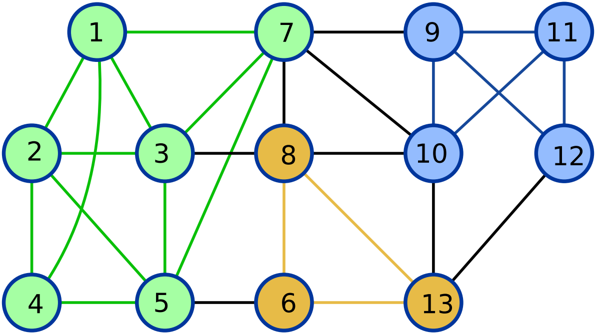
- Given an undirected, unweighted graph $G = (V, E)$ suppose we partition the nodes into k modules $\mathcal{C} = C_1, C_2, \ldots C_k$.
- How do we measure the "quality" of $\mathcal{C}$?
- Intuition: many more edges within modules than among modules.

Initial Definition of Modularity



- How do we count the number of edges within modules?

Initial Definition of Modularity

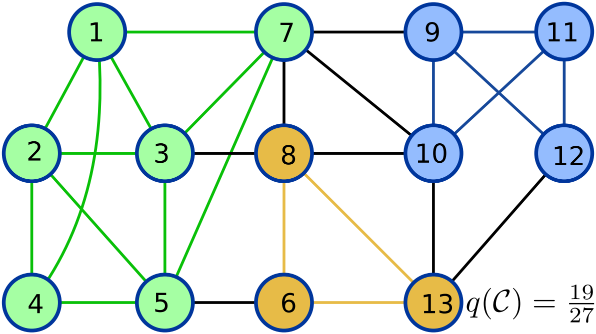


- How do we count the number of edges within modules?
- For every node $u \in V$, define $c(u)$ as the index of u's module.

$$q(\mathcal{C}) = \frac{1}{m} \sum_{(u,v) \in E} \delta(c(u), c(v)), \text{ where } \delta \text{ is the Kronecker delta function}$$

$$= \frac{1}{2m} \sum_{u,v \in V} a(u,v)\delta(c(u), c(v)), \text{ where } a(u,v) = 1 \text{ iff } (u,v) \text{ is an edge}$$

Initial Definition of Modularity



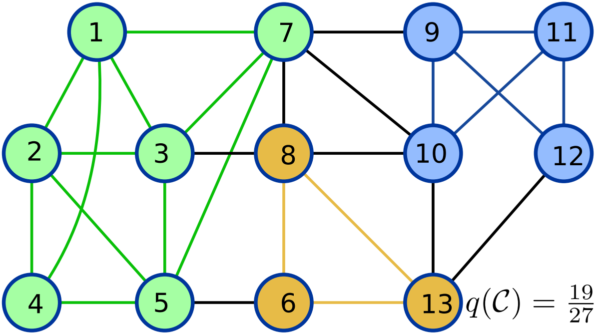
$q(\mathcal{C}) = \frac{19}{27}$

- How do we count the number of edges within modules?
- For every node $u \in V$, define $c(u)$ as the index of u's module.

$$q(\mathcal{C}) = \frac{1}{m} \sum_{(u,v) \in E} \delta(c(u), c(v)), \text{ where } \delta \text{ is the Kronecker delta function}$$

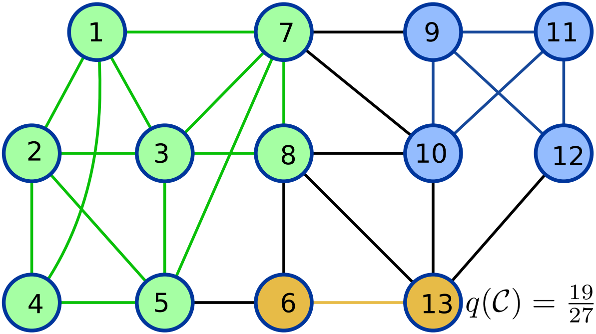
$$= \frac{1}{2m} \sum_{u,v \in V} a(u,v)\delta(c(u), c(v)), \text{ where } a(u,v) = 1 \text{ iff } (u,v) \text{ is an edge}$$

Optimising Modularity



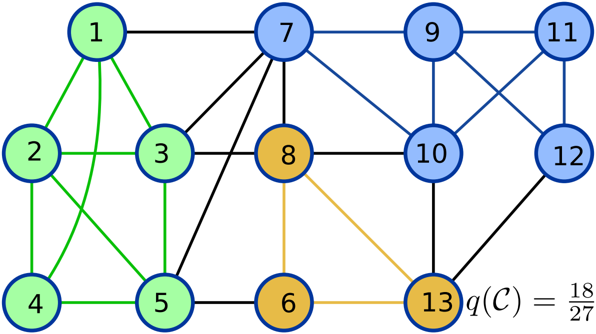
$$q(\mathcal{C}) = \frac{1}{2m} \sum_{u,v \in V} a(u,v)\delta(c(u), c(v))$$

Optimising Modularity



$$q(\mathcal{C}) = \frac{1}{2m} \sum_{u,v \in V} a(u,v)\delta(c(u), c(v))$$

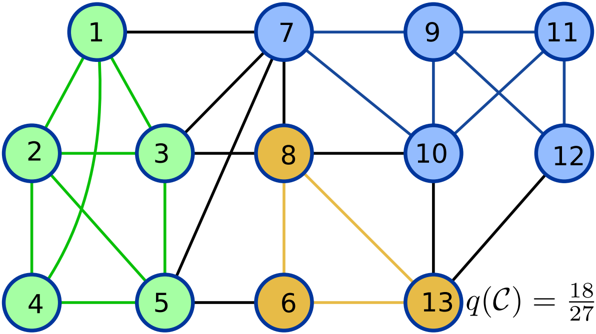
Optimising Modularity



$q(\mathcal{C}) = \frac{18}{27}$

$$q(\mathcal{C}) = \frac{1}{2m} \sum_{u,v \in V} a(u, v)\delta(c(u), c(v))$$

Optimising Modularity

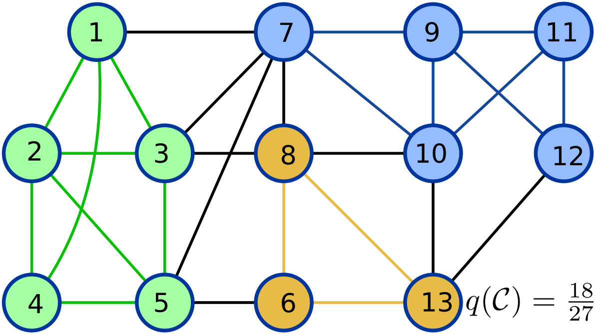


$q(\mathcal{C}) = \frac{18}{27}$

$$q(\mathcal{C}) = \frac{1}{2m} \sum_{u,v \in V} a(u,v)\delta(c(u), c(v))$$

- Should we maximise or minimise $q(\mathcal{C})$?

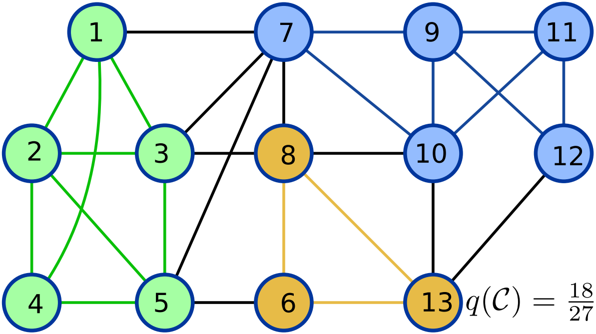
Optimising Modularity



$$q(\mathcal{C}) = \frac{1}{2m} \sum_{u,v \in V} a(u,v)\delta(c(u), c(v))$$

- Should we maximise or minimise $q(\mathcal{C})$? Maximise it.
- What is the value of $q(\mathcal{C})$ if we place all nodes in G in a single cluster?

Optimising Modularity



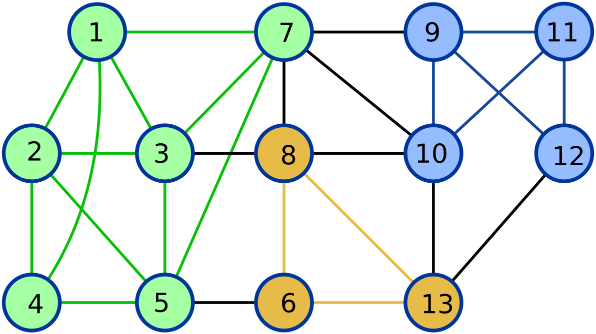
$$q(\mathcal{C}) = \frac{1}{2m} \sum_{u,v \in V} a(u,v)\delta(c(u), c(v))$$

- Should we maximise or minimise $q(\mathcal{C})$? Maximise it.
- What is the value of $q(\mathcal{C})$ if we place all nodes in G in a single cluster? 1!

Two Criteria for High Quality Partitions

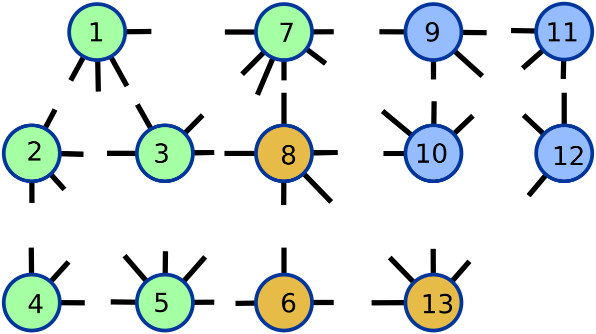
1. Nodes are in highly cohesive modules, i.e., nodes within the same module will be strongly connected with each other.
2. The amount of intramodule connectivity in a good partition will be greater than expected by chance, as defined by a network in which edges are placed between nodes at random.
3. Proposed by Newman and Girvan, 2004.

Configuration Model



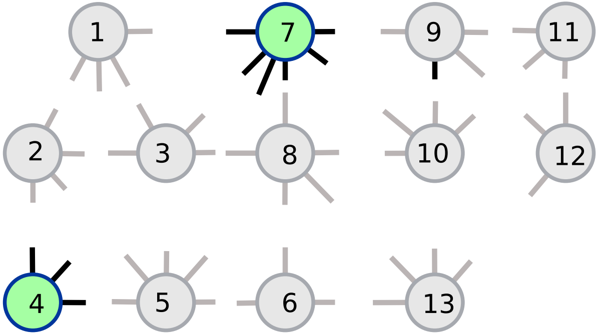
- Method to generate random graphs.
- Ensure that the random graphs have the same degree sequence as G, but allow self loops and multi-edges.

Configuration Model



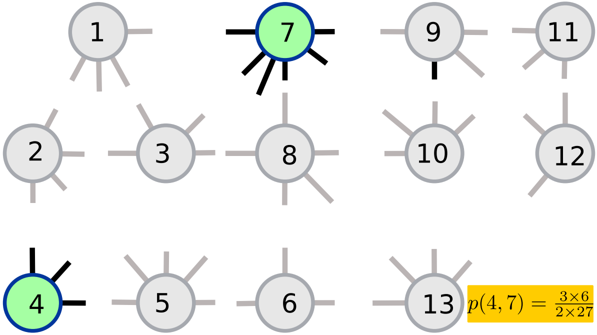
- Cut each edge in G in half.
- Each node u has $d(u)$ stubs; total number of stubs is $2m$.
- For each stub select another stub uniformly at random and connect them by an edge.

Configuration Model



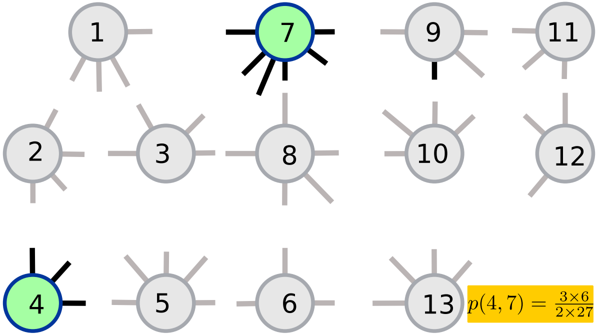
- What is the probability of an edge between nodes u and v?

Configuration Model



- What is the probability of an edge between nodes u and v? $\frac{d(u)d(v)}{2m}$.

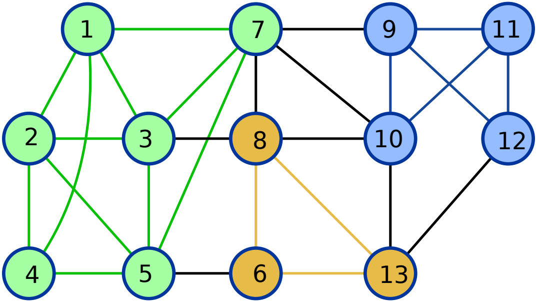
Configuration Model



- What is the probability of an edge between nodes u and v? $\frac{d(u)d(v)}{2m}$.
- Therefore modularity of the partition of a random graph in the configuration model into the same modules $\mathcal{C} = C_1, C_2, \ldots C_k$

$$q(\mathcal{C}) = \frac{1}{2m} \sum_{u,v \in V} \frac{d(u)d(v)}{2m} \delta(c(u), c(v))$$

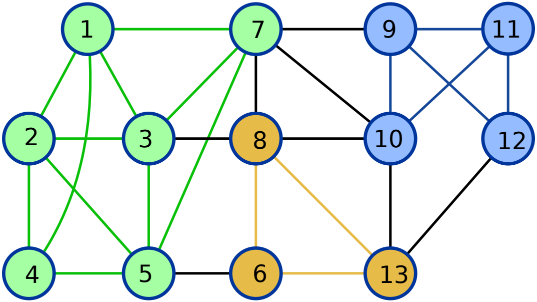
Final Definition of Modularity



$$q(\mathcal{C}) = \frac{1}{2m} \sum_{u,v \in V} \left(a(u,v) - \frac{d(u)d(v))}{2m} \right) \delta(C(u), C(v))$$

- What is the range of $q(\mathcal{C})$?

Final Definition of Modularity



$$q(\mathcal{C}) = \frac{1}{2m} \sum_{u,v \in V} \left(a(u,v) - \frac{d(u)d(v))}{2m} \right) \delta(C(u), C(v))$$

- What is the range of $q(\mathcal{C})$? Between -1/2 and 1.
  - $q(\mathcal{C}) > 0$: $\mathcal{C}$ has higher intramodule connectivity than expected by chance from configuration model.
  - $q(\mathcal{C}) = 0$: $\mathcal{C}$ has same intramodule connectivity as expected in a random graph.
  - $q(\mathcal{C}) < 0$: $\mathcal{C}$ has no modular structure.

Using Modularity

- Now that we have defined a nice measure for the quality of a partition, how do we use it?
- Definition of q does not specify the number of clusters.

Using Modularity

- Now that we have defined a nice measure for the quality of a partition, how do we use it?
- Definition of q does not specify the number of clusters.
- Hierarchical clustering: Compute modularity after every merge and output the clustering with the largest value.
- Any other clustering algorithm: compute the modularity of the result.

Using Modularity

- Now that we have defined a nice measure for the quality of a partition, how do we use it?
- Definition of q does not specify the number of clusters.
- Hierarchical clustering: Compute modularity after every merge and output the clustering with the largest value.
- Any other clustering algorithm: compute the modularity of the result.
- Develop a new algorithm to maximise modularity.
  - Maximising modularity is NP-hard.
  - We must rely on heuristics to make the modularity as large as possible.

Greedy Algorithm

- Proposed by Newman, 2004.

1. Start with every node in its own module.
2. While there are at least two modules
   1. Compute the pair of modules whose merger will result in the largest increase or smallest decrease in q.
   2. Merge this pair of modules into one.
3. Return the clustering with the largest value of q.

Greedy Algorithm

- Proposed by Newman, 2004.

1. Start with every node in its own module.
2. While there are at least two modules
   1. Compute the pair of modules whose merger will result in the largest increase or smallest decrease in q.
   2. Merge this pair of modules into one.
3. Return the clustering with the largest value of q.

- Hierarchical clustering algorithm built directly around maximisation of q.
- Allows q to decrease to preserve the principle of hierarchical clustering.
- Why is the algorithm "greedy"?

Greedy Algorithm

- Proposed by Newman, 2004.

1. Start with every node in its own module.
2. While there are at least two modules
   1. Compute the pair of modules whose merger will result in the largest increase or smallest decrease in q.
   2. Merge this pair of modules into one.
3. Return the clustering with the largest value of q.

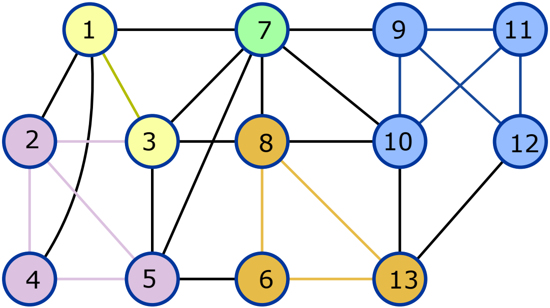
- Hierarchical clustering algorithm built directly around maximisation of q.
- Allows q to decrease to preserve the principle of hierarchical clustering.
- Why is the algorithm "greedy"? Merging of two modules cannot be undone.

Louvain Algorithm: Phase 1

- Proposed by Blondel *et al.*, 2008.

1. Start with every node in its own module.
2. For every node $u \in V$ and every neighbour v of u, evaluate the change in q when we remove u from its module and add it to v's module.
3. Move u to that neighbour's module for which increase in q is largest.
4. Repeat the previous two steps until q does not increase.

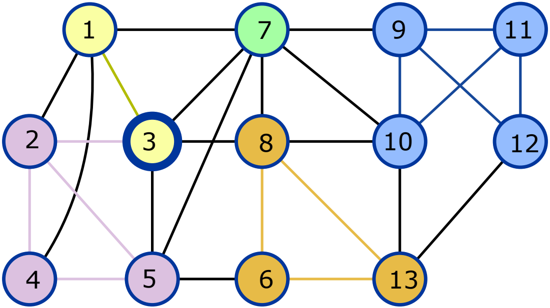
Louvain Algorithm: Phase 1



- Proposed by Blondel *et al.*, 2008.

1. Start with every node in its own module.
2. For every node $u \in V$ and every neighbour v of u, evaluate the change in q when we remove u from its module and add it to v's module.
3. Move u to that neighbour's module for which increase in q is largest.
4. Repeat the previous two steps until q does not increase.

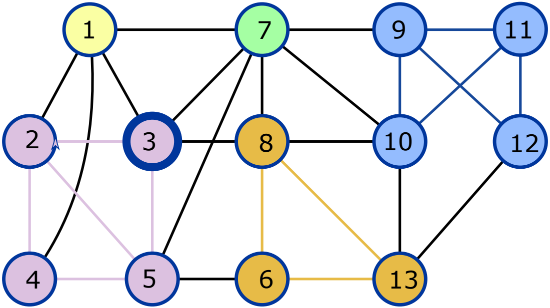
Louvain Algorithm: Phase 1



- Proposed by Blondel *et al.*, 2008.

1. Start with every node in its own module.
2. For every node $u \in V$ and every neighbour v of u, evaluate the change in q when we remove u from its module and add it to v's module.
3. Move u to that neighbour's module for which increase in q is largest.
4. Repeat the previous two steps until q does not increase.

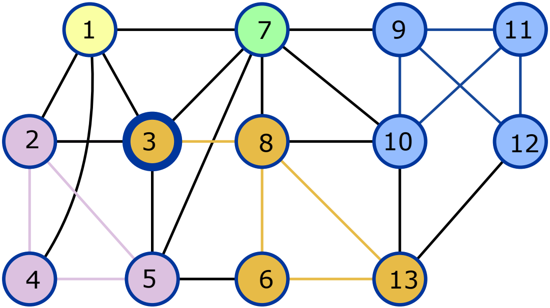
Louvain Algorithm: Phase 1



- Proposed by Blondel *et al.*, 2008.

1. Start with every node in its own module.
2. For every node $u \in V$ and every neighbour v of u, evaluate the change in q when we remove u from its module and add it to v's module.
3. Move u to that neighbour's module for which increase in q is largest.
4. Repeat the previous two steps until q does not increase.

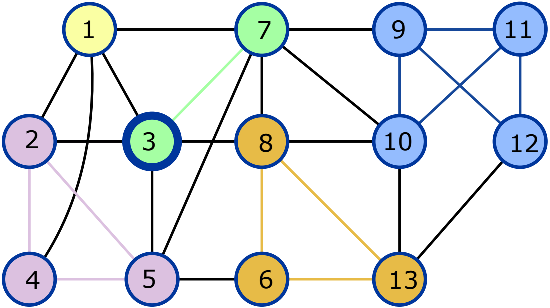
Louvain Algorithm: Phase 1



- Proposed by Blondel *et al.*, 2008.

1. Start with every node in its own module.
2. For every node $u \in V$ and every neighbour v of u, evaluate the change in q when we remove u from its module and add it to v's module.
3. Move u to that neighbour's module for which increase in q is largest.
4. Repeat the previous two steps until q does not increase.

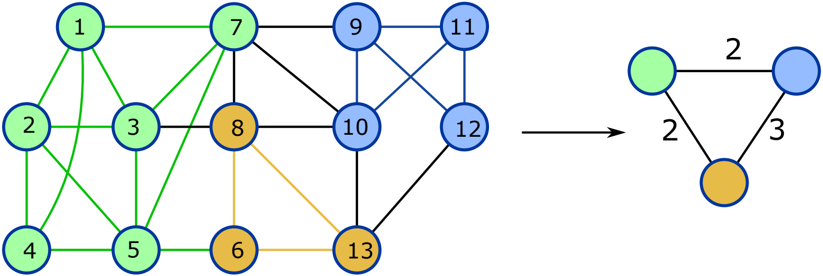
Louvain Algorithm: Phase 1



- Proposed by Blondel *et al.*, 2008.

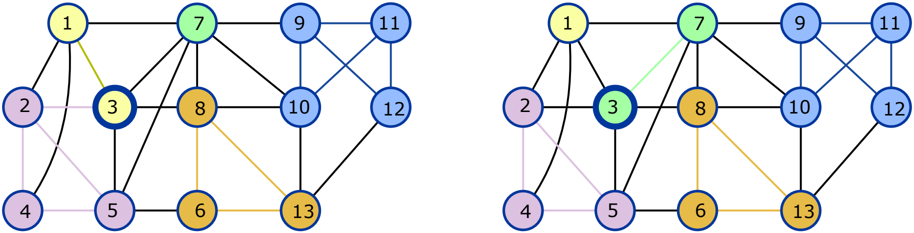
1. Start with every node in its own module.
2. For every node $u \in V$ and every neighbour v of u, evaluate the change in q when we remove u from its module and add it to v's module.
3. Move u to that neighbour's module for which increase in q is largest.
4. Repeat the previous two steps until q does not increase.

Louvain Algorithm: Phase 2



1. Construct a new graph where every module is a node and a weighted edge represents (multiple) connections between two modules.
2. Repeat Phases 1 and 2 until no further gains in q are possible.

Louvain Algorithm: Efficiency



- Efficient calculation of change in q upon swapping makes this algorithm very fast.

Limitations of Modularity

- Modularity generally increases as number of nodes and modules in a graph increase.
- Many very similar partitions have similar values of q.
- Modularity has a resolution limit: small modules may be combined simply to increase q.
- Random graph model is quite simple: assumes every node has an equal probability of connecting to every other node.

Limitations of Modularity

- Modularity generally increases as number of nodes and modules in a graph increase.
- Many very similar partitions have similar values of q.
- Modularity has a resolution limit: small modules may be combined simply to increase q.
- Random graph model is quite simple: assumes every node has an equal probability of connecting to every other node.
- Many alternatives proposed to address these limitations.