Script generated by TTT

Title: groh: profile1 (13.05.2014)

Date: Tue May 13 11:59:57 CEST 2014

Duration: 89:39 min

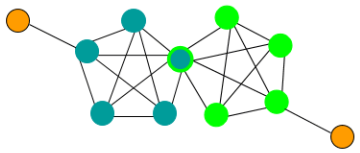
Pages: 83

Cliques

- A subset U⊆V of a Graph (V,E) is a clique if G([U]) is a complete graph; G([U]) is the sub-graph induced by U.
- A clique is maximal if there is no clique U' with U ⊂ U' in G
- A clique is a maximum clique if there is no clique with more vertices in G

Cliques

- If Graph is dense → cliques exist:
- (Turan 1941 (see [2])): If $|E| > |V|^2/2 \ (k-2)/(k-1)$ then G contains a clique of size k.
- Usually many different maximal cliques exist in a graph; they can even overlap without being identical



N-Cliques, N-Clubs, N-Clans

- Cliques are very "strict" → Alternative candidates for groups: Distance based structures:
  - U is N-clique iff $\forall u,v \in U : dist_G(u,v) \leq N$ (non-local def.!)
  - U is N-club iff $diam(G([U])) \leq N$
  - U is N-clan iff U is maximal N-clique and $diam(G([U])) \leq N$
- Criticisms:
  - Since dist is evaluated w.r.t. to G and not G([U]) (thus N-cliques are not local structures), N-cliques need not even be connected and can have a diameter diam(G([U]) > N

N-Cliques, N-Clubs, N-Clans

- Cliques are very "strict" → Alternative candidates for groups:
Distance based structures:
  - U is N-clique iff $\forall u,v \in U : \text{dist}_G(u,v) \leq N$ (non-local def.!)
  - U is N-club iff $\text{diam}(G([U])) \leq N$
  - U is N-clan iff U is maximal N-clique and $\text{diam}(G([U])) \leq N$
- Criticisms:
  - Since dist is evaluated w.r.t. to G and not G([U]) (thus N-cliques are not local structures), N-cliques need not even be connected and can have a diameter diam(G([U]) > N

N-Cliques, N-Clubs, N-Clans

- U is N-clique iff $\forall u,v \in V : \text{dist}_G(u,v) \leq N$
- U is N-club iff $\text{diam}(G([U])) \leq N$
- U is N-clan iff U is maximal N-clique and $\text{diam}(G([U])) \leq N$

- → N-clan: restrict dist-condition to paths of nodes within the structure: easy to find (just drop all n-cliques with diameter greater than N)
- → N-club: regard all induced graphs with diameter less than N: harder to find
- It can be shown / seen from the def.:
  -- all N-clans are N-cliques;
  -- all N-clubs are contained within N-cliques;
  -- all N-clans are n-clubs
  -- there are N-clubs that are not N-clans

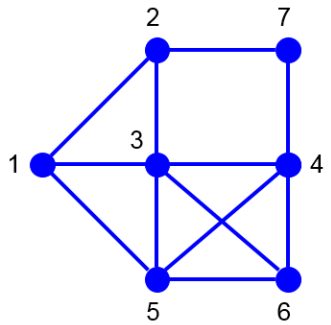
N-Cliques, N-Clubs, N-Clans

- U is N-clique iff $\forall u,v \in V : \text{dist}_G(u,v) \leq N$
- U is N-club iff $\text{diam}(G([U])) \leq N$
- U is N-clan iff U is maximal N-clique and $\text{diam}(G([U])) \leq N$

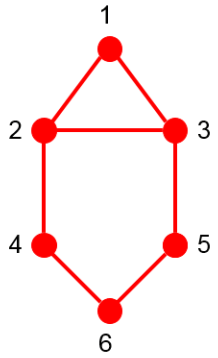
- → N-clan: restrict dist-condition to paths of nodes within the structure: easy to find (just drop all n-cliques with diameter greater than N)
- → N-club: regard all induced graphs with diameter less than N: harder to find
- It can be shown / seen from the def.:
  -- all N-clans are N-cliques;
  -- all N-clubs are contained within N-cliques;
  -- all N-clans are n-clubs
  -- there are N-clubs that are not N-clans

N-Cliques, N-Clubs, N-Clans

- U is N-clique iff $\forall u,v \in V : \text{dist}_G(u,v) \leq N$
- U is N-club iff $\text{diam}(G([U])) \leq N$
- U is N-clan iff U is maximal N-clique and $\text{diam}(G([U])) \leq N$



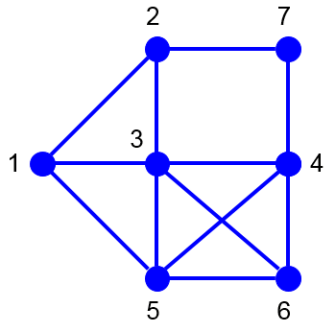
cliques: {1, 2, 3}, {1, 3, 5}, {3, 4, 5, 6}



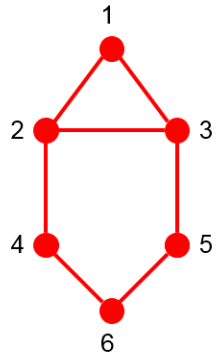
2-cliques: {1, 2, 3, 4, 5}, {2, 3, 4, 5, 6}
2-clubs: {1, 2, 3, 4}, {1, 2, 3, 5}, {2, 3, 4, 5, 6}
2-clan: {2, 3, 4, 5, 6}

N-Cliques, N-Clubs, N-Clans

- U is N-clique iff $\forall u,v \in V$: $dist_G(u,v) \leq N$
- U is N-club iff $diam(G([U])) \leq N$
- U is N-clan iff U is maximal N-clique and $diam(G([U])) \leq N$



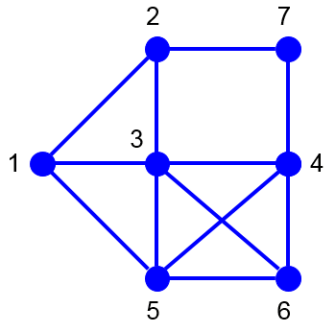
cliques: {1, 2, 3}, {1, 3, 5}, {3, 4, 5, 6}



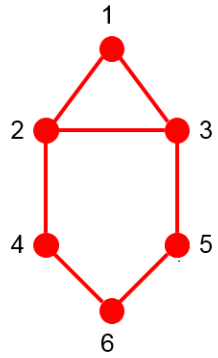
2-cliques: {1, 2, 3, 4, 5}, {2, 3, 4, 5, 6}
2-clubs: {1, 2, 3, 4}, {1, 2, 3, 5}, {2, 3, 4, 5, 6}
2-clan: {2, 3, 4, 5, 6}

N-Cliques, N-Clubs, N-Clans

- U is N-clique iff $\forall u,v \in V$: $dist_G(u,v) \leq N$
- U is N-club iff $diam(G([U])) \leq N$
- U is N-clan iff U is maximal N-clique and $diam(G([U])) \leq N$



cliques: {1, 2, 3}, {1, 3, 5}, {3, 4, 5, 6}

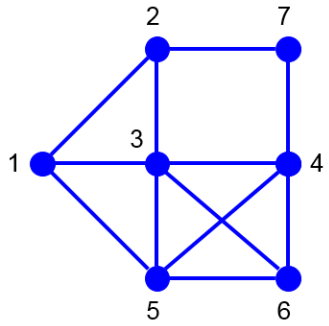


2-cliques: {1, 2, 3, 4, 5}, {2, 3, 4, 5, 6}
2-clubs: {1, 2, 3, 4}, {1, 2, 3, 5}, {2, 3, 4, 5, 6}
2-clan: {2, 3, 4, 5, 6}

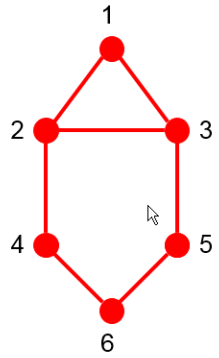
[11]

N-Cliques, N-Clubs, N-Clans

- U is N-clique iff $\forall u,v \in V$: $dist_G(u,v) \leq N$
- U is N-club iff $diam(G([U])) \leq N$
- U is N-clan iff U is maximal N-clique and $diam(G([U])) \leq N$



cliques: {1, 2, 3}, {1, 3, 5}, {3, 4, 5, 6}

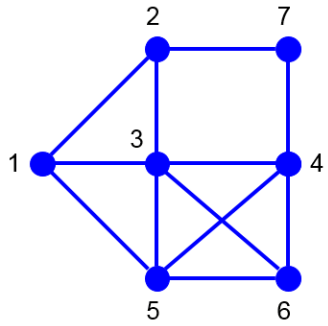


2-cliques: {1, 2, 3, 4, 5}, {2, 3, 4, 5, 6}
2-clubs: {1, 2, 3, 4}, {1, 2, 3, 5}, {2, 3, 4, 5, 6}
2-clan: {2, 3, 4, 5, 6}

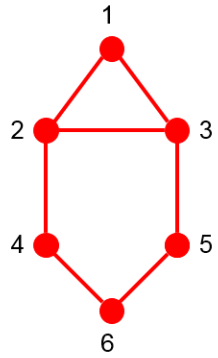
[11]

N-Cliques, N-Clubs, N-Clans

- U is N-clique iff $\forall u,v \in V$: $dist_G(u,v) \leq N$
- U is N-club iff $diam(G([U])) \leq N$
- U is N-clan iff U is maximal N-clique and $diam(G([U])) \leq N$



cliques: {1, 2, 3}, {1, 3, 5}, {3, 4, 5, 6}

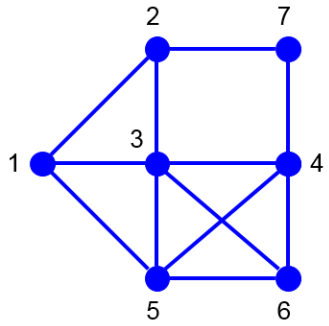


2-cliques: {1, 2, 3, 4, 5}, {2, 3, 4, 5, 6}
2-clubs: {1, 2, 3, 4}, {1, 2, 3, 5}, {2, 3, 4, 5, 6}
2-clan: {2, 3, 4, 5, 6}

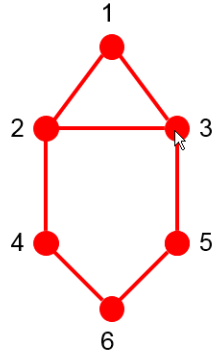
[11]

N-Cliques, N-Clubs, N-Clans

- U is N-clique iff $\forall u,v \in V : \text{dist}_G(u,v) \leq N$
- U is N-club iff $\text{diam}(G([U])) \leq N$
- U is N-clan iff U is maximal N-clique and $\text{diam}(G([U])) \leq N$

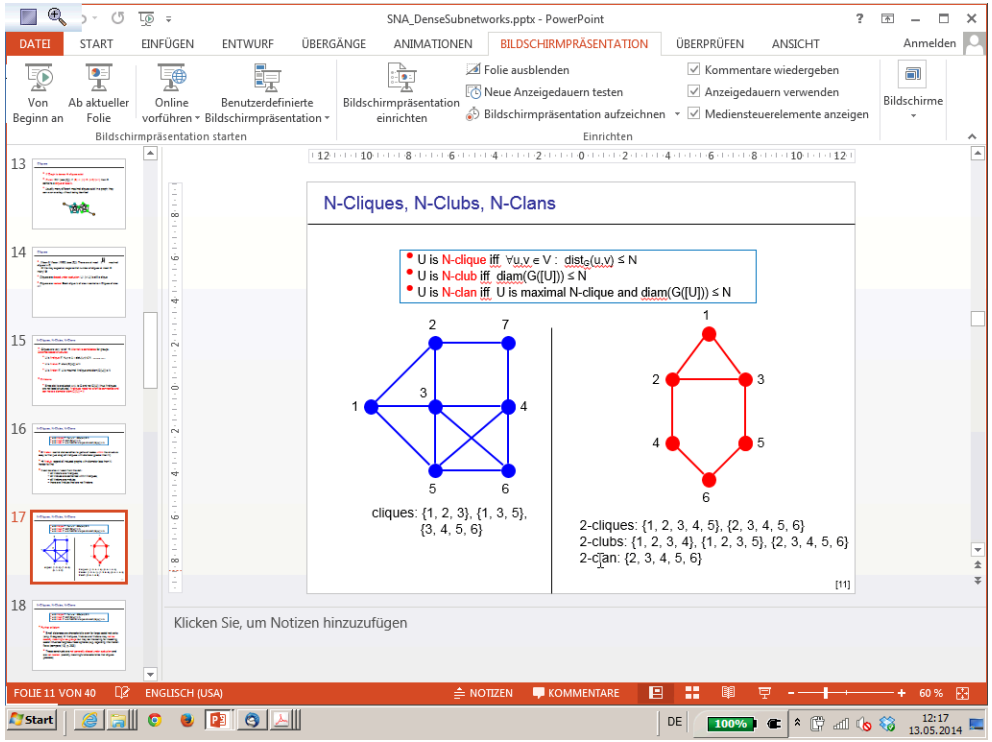


cliques: {1, 2, 3}, {1, 3, 5}, {3, 4, 5, 6}



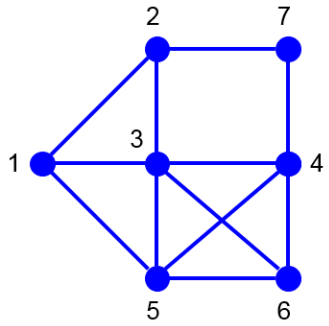
2-cliques: {1, 2, 3, 4, 5}, {2, 3, 4, 5, 6}
2-clubs: {1, 2, 3, 4}, {1, 2, 3, 5}, {2, 3, 4, 5, 6}
2-clan: {2, 3, 4, 5, 6}

[11]

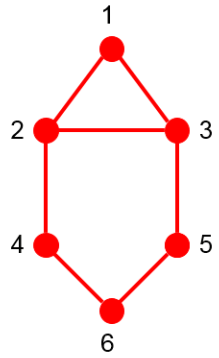


N-Cliques, N-Clubs, N-Clans

- U is N-clique iff $\forall u,v \in V : \text{dist}_G(u,v) \leq N$
- U is N-club iff $\text{diam}(G([U])) \leq N$
- U is N-clan iff U is maximal N-clique and $\text{diam}(G([U])) \leq N$



cliques: {1, 2, 3}, {1, 3, 5}, {3, 4, 5, 6}

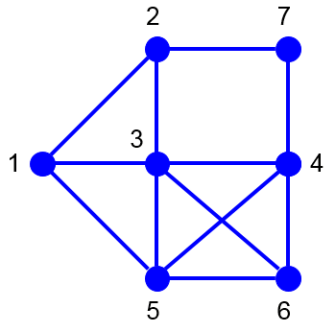


maximal 2-cliques: {1, 2, 3, 4, 5}, {2, 3, 4, 5, 6}
2-clubs: {1, 2, 3, 4}, {1, 2, 3, 5}, {2, 3, 4, 5, 6}
2-clan: {2, 3, 4, 5, 6}

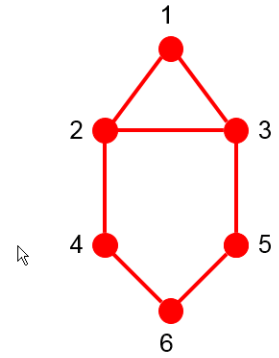
[11]

N-Cliques, N-Clubs, N-Clans

- U is N-clique iff $\forall u,v \in V : \text{dist}_G(u,v) \leq N$
- U is N-club iff $\text{diam}(G([U])) \leq N$
- U is N-clan iff U is maximal N-clique and $\text{diam}(G([U])) \leq N$



cliques: {1, 2, 3}, {1, 3, 5}, {3, 4, 5, 6}



maximal 2-cliques: {1, 2, 3, 4, 5}, {2, 3, 4, 5, 6}
2-clubs: {1, 2, 3, 4}, {1, 2, 3, 5}, {2, 3, 4, 5, 6}
2-clan: {2, 3, 4, 5, 6}

[11]

N-Cliques, N-Clubs, N-Clans

- U is N-clique iff $\forall u,v \in V : \text{dist}_G(u,v) \leq N$
- U is N-club iff $\text{diam}(G([U])) \leq N$
- U is N-clan iff U is maximal N-clique and $\text{diam}(G([U])) \leq N$

- → N-clan: restrict dist-condition to paths of nodes within the structure: easy to find (just drop all n-cliques with diameter greater than N)

- → N-club: regard all induced graphs with diameter less than N: harder to find

- It can be shown / seen from the def.:
  -- all N-clans are N-cliques;
  -- all N-clubs are contained within N-cliques;
  -- all N-clans are n-clubs
  -- there are N-clubs that are not N-clans

Cliques: Algorithms

- In connection with cliques: Algorithms with time complexity O(|E| + |V|) for the problems:

  - Determine if $U \subseteq V$ is a clique (Test pairs of vertices of U if they are in E. Although up to $\binom{|V|}{2}$ such pairs may exist: If |E| edges have been searched, search is over

  - Determine if clique U is maximal (Test all vertices in V-U if they are connected to all vertices in U; again: If |E| edges have been searched, search is over)

  - Compute lexicographically smallest maximal clique containing U: Assume vertices are sorted; Test all vertices in V-U in ascending order: if they are connected to all vertices in U, add to U; again: If |E| edges have been searched, search is over)

Cliques: Algorithms

- In connection with cliques: Algorithms with time complexity O(|E| + |V|) for the problems:

  - Determine if $U \subseteq V$ is a clique (Test pairs of vertices of U if they are in E. Although up to $\binom{|V|}{2}$ such pairs may exist: If |E| edges have been searched, search is over

  - Determine if clique U is maximal (Test all vertices in V-U if they are connected to all vertices in U; again: If |E| edges have been searched, search is over)

  - Compute lexicographically smallest maximal clique containing U: Assume vertices are sorted; Test all vertices in V-U in ascending order: if they are connected to all vertices in U, add to U; again: If |E| edges have been searched, search is over)

Cliques: Algorithms

- In connection with cliques: Algorithms with time complexity O(|E| + |V|) for the problems:

  - Determine if $U \subseteq V$ is a clique (Test pairs of vertices of U if they are in E. Although up to $\binom{|V|}{2}$ such pairs may exist: If |E| edges have been searched, search is over

  - Determine if clique U is maximal (Test all vertices in V-U if they are connected to all vertices in U; again: If |E| edges have been searched, search is over)

  - Compute lexicographically smallest maximal clique containing U: Assume vertices are sorted; Test all vertices in V-U in ascending order: if they are connected to all vertices in U, add to U; again: If |E| edges have been searched, search is over)

Cliques: Algorithms

- In connection with cliques: Algorithms with time complexity O(|E| + |V|) for the problems:
  - Determine if $U \subseteq V$ is a clique (Test pairs of vertices of U if they are in E. Although up to $\binom{|V|}{2}$ such pairs may exist: If |E| edges have been searched, search is over
  - Determine if clique U is maximal (Test all vertices in V-U if they are connected to all vertices in U; again: If |E| edges have been searched, search is over)
  - Compute lexicographically smallest maximal clique containing U: Assume vertices are sorted; Test all vertices in V-U in ascending order: if they are connected to all vertices in U, add to U; again: If |E| edges have been searched, search is over)

Cliques: Algorithms

- In connection with cliques: Algorithms with time complexity O(|E| + |V|) for the problems:
  - Determine if $U \subseteq V$ is a clique (Test pairs of vertices of U if they are in E. Although up to $\binom{|V|}{2}$ such pairs may exist: If |E| edges have been searched, search is over
  - Determine if clique U is maximal (Test all vertices in V-U if they are connected to all vertices in U; again: If |E| edges have been searched, search is over)
  - Compute lexicographically smallest maximal clique containing U: Assume vertices are sorted; Test all vertices in V-U in ascending order: if they are connected to all vertices in U, add to U; again: If |E| edges have been searched, search is over)

Cliques: Algorithms

- In connection with cliques: Algorithms with time complexity O(|E| + |V|) for the problems:
  - Determine if $U \subseteq V$ is a clique (Test pairs of vertices of U if they are in E. Although up to $\binom{|V|}{2}$ such pairs may exist: If |E| edges have been searched, search is over
  - Determine if clique U is maximal (Test all vertices in V-U if they are connected to all vertices in U; again: If |E| edges have been searched, search is over)
  - Compute lexicographically smallest maximal clique containing U: Assume vertices are sorted; Test all vertices in V-U in ascending order: if they are connected to all vertices in U, add to U; again: If |E| edges have been searched, search is over)

Cliques: Algorithms

- In connection with cliques: Algorithms with time complexity O(|E| + |V|) for the problems:
  - Determine if $U \subseteq V$ is a clique (Test pairs of vertices of U if they are in E. Although up to $\binom{|V|}{2}$ such pairs may exist: If |E| edges have been searched, search is over
  - Determine if clique U is maximal (Test all vertices in V-U if they are connected to all vertices in U; again: If |E| edges have been searched, search is over)
  - Compute lexicographically smallest maximal clique containing U: Assume vertices are sorted; Test all vertices in V-U in ascending order: if they are connected to all vertices in U, add to U; again: If |E| edges have been searched, search is over)

Cliques: Algorithms

- Naive algorithm for finding the maximum clique: Exhaustive Search: Compute all subsets of V and check for clique → $O(|V|^2 2^{|V|})$ worst case
- Is there a substantially better algorithm? Probably not: The decision problem CLIQUE(G,k): "Has G a clique of size at least k" is NP complete. (Solving the decision problem in time T(|V|) would yield an alg for determining the maximal k in O(T(|V|) log (|V|)) via binary search.

Proof: Reduce CLIQUE on SATISFIABILITY;

- → Unless P=NP there is no P algorithm X to compute clique of size k if G is guaranteed to contain such a clique.

Proof: Having X we could decide CLIQUE in P time;

Cliques: Algorithms

- Naive algorithm for finding the maximum clique: Exhaustive Search: Compute all subsets of V and check for clique → $O(n^2 2^n)$
- Is there a slightly better algorithm? Yes: We can improve the exponential function's base from 2 to approx. 1.38 (see [2])
- Can we expect to find the answer to the problem "how many cliques with exactly k nodes exist"? → Exhaustive search: $\Theta(|V|^k)$
- For triangles we can be better than $\Theta(|V|^3)$: An alg with $O(|V|^{2.376})$ exists ☺
- For other k: An analogous technique allows for alg with $O(|V|^{\beta(k)})$ with $\beta(k)<k$

Cliques: Algorithms

- Naive algorithm for finding the maximum clique: Exhaustive Search: Compute all subsets of V and check for clique → $O(n^2 2^n)$
- Is there a slightly better algorithm? Yes: We can improve the exponential function's base from 2 to approx. 1.38 (see [2])
- Can we expect to find the answer to the problem "how many cliques with exactly k nodes exist"? → Exhaustive search: $\Theta(|V|^k)$
- For triangles we can be better than $\Theta(|V|^3)$: An alg with $O(|V|^{2.376})$ exists ☺
- For other k: An analogous technique allows for alg with $O(|V|^{\beta(k)})$ with $\beta(k)<k$

Cliques: Algorithms

- Naive algorithm for finding the maximum clique: Exhaustive Search: Compute all subsets of V and check for clique → $O(n^2 2^n)$
- Is there a slightly better algorithm? Yes: We can improve the exponential function's base from 2 to approx. 1.38 (see [2])
- Can we expect to find the answer to the problem "how many cliques with exactly k nodes exist"? → Exhaustive search: $\Theta(|V|^k)$
- For triangles we can be better than $\Theta(|V|^3)$: An alg with $O(|V|^{2.376})$ exists ☺
- For other k: An analogous technique allows for alg with $O(|V|^{\beta(k)})$ with $\beta(k)<k$

Cliques: Algorithms

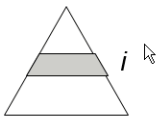
- Naive algorithm for finding the maximum clique: Exhaustive Search: Compute all subsets of V and check for clique → $O(n^2 2^n)$
- Is there a slightly better algorithm? Yes: We can improve the exponential function's base from 2 to approx. 1.38 (see [2])
- Can we expect to find the answer to the problem "how many cliques with exactly k nodes exist"? → Exhaustive search: $\Theta(|V|^k)$
- For triangles we can be better than $\Theta(|V|^3)$: An alg with $O(|V|^{2.376})$ exists ☺
- For other k: An analogous technique allows for alg with $O(|V|^{\beta(k)})$ with $\beta(k)<k$

Cliques: Algorithms

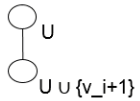
- Naive algorithm for finding the maximum clique: Exhaustive Search: Compute all subsets of V and check for clique → $O(n^2 2^n)$
- Is there a slightly better algorithm? Yes: We can improve the exponential function's base from 2 to approx. 1.38 (see [2])
- Can we expect to find the answer to the problem "how many cliques with exactly k nodes exist"? → Exhaustive search: $\Theta(|V|^k)$
- For triangles we can be better than $\Theta(|V|^3)$: An alg with $O(|V|^{2.376})$ exists ☺
- For other k: An analogous technique allows for alg with $O(|V|^{\beta(k)})$ with $\beta(k)<k$

Cliques: Enumeration

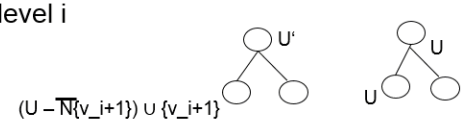
- Binary tree, n levels, leaves only at level n
- each level i ↔ vertex v_i
- nodes at level i: maximal cliques in G[v_1,v_2,…,v_i}]
- level i+1: determine children of node U at level i: two cases :



(1) v_i+1 adjacent to all nodes in U (U⊆N(v_i+1)):
→ U ∪ {v_i+1} is maximal clique in G[v_1,v_2,…,v_i, v_i+1}]
→ U ∪ {v_i+1} is only child of U



(2) ∃ vertex in U not adjacent to v_i+1 (U⊈N(v_i+1)):
→ U itself is a maximal clique in G[v_1,v_2,…,v_i, v_i+1}]
→ if (U – $\overline{N}${v_i+1}) ∪ {v_i+1} is maximal it is also a maximal clique in G[v_1,v_2,…,v_i, v_i+1}]
→ (U – $\overline{N}${v_i+1}) ∪ {v_i+1} is potentially child of many possible nodes → define it als child of lexiographically smallest clique (node) U' at level i



Cliques: Enumeration

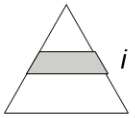
- Binary tree, n levels, leaves only at level n
- each level i ↔ vertex v_i
- nodes at level i: maximal cliques in G[v_1,v_2,…,v_i}]
- level i+1: determine children of node U at level i: two cases :

(1) v_i+1 adjacent to all nodes in U (U⊆N(v_i+1)):
→ U ∪ {v_i+1} is maximal clique in G[v_1,v_2,…,v_i, v_i+1}]
→ U ∪ {v_i+1} is only child of U

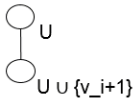
(2) ∃ vertex in U not adjacent to v_i+1 (U⊈N(v_i+1)):
→ U itself is a maximal clique in G[v_1,v_2,…,v_i, v_i+1}]
→ if (U – $\overline{N}${v_i+1}) ∪ {v_i+1} is maximal it is also a maximal clique in G[v_1,v_2,…,v_i, v_i+1}]
→ (U – $\overline{N}${v_i+1}) ∪ {v_i+1} is potentially child of many possible nodes → define it als child of lexiographically smallest clique (node) U' at level i

Cliques: Enumeration

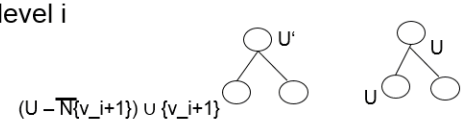
- Binary tree, n levels, leaves only at level n
- each level i ↔ vertex v_i
- nodes at level i: maximal cliques in G[v_1,v_2,…,v_i}]
- level i+1: determine children of node U at level i: two cases :



(1) v_i+1 adjacent to all nodes in U (U⊆N(v_i+1):
→ U ∪ {v_i+1} is maximal clique in G[v_1,v_2,…,v_i, v_i+1}]
→ U ∪ {v_i+1} is only child of U

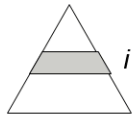


(2) ∃ vertex in U not adjacent to v_i+1 (U⊈N(v_i+1):
→ U itself is a maximal clique in G[v_1,v_2,…,v_i, v_i+1}]
→ if (U – $\overline{N}${v_i+1}) ∪ {v_i+1} is maximal it is also a maximal clique in G[v_1,v_2,…,v_i, v_i+1}]
→ (U – $\overline{N}${v_i+1}) ∪ {v_i+1} is potentially child of many possible nodes → define it als child of lexiographically smallest clique (node) U‘ at level i

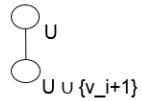


Cliques: Enumeration

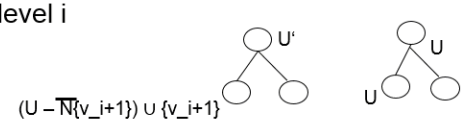
- Binary tree, n levels, leaves only at level n
- each level i ↔ vertex v_i
- nodes at level i: maximal cliques in G[v_1,v_2,…,v_i}]
- level i+1: determine children of node U at level i: two cases :



(1) v_i+1 adjacent to all nodes in U (U⊆N(v_i+1):
→ U ∪ {v_i+1} is maximal clique in G[v_1,v_2,…,v_i, v_i+1}]
→ U ∪ {v_i+1} is only child of U



(2) ∃ vertex in U not adjacent to v_i+1 (U⊈N(v_i+1):
→ U itself is a maximal clique in G[v_1,v_2,…,v_i, v_i+1}]
→ if (U – $\overline{N}${v_i+1}) ∪ {v_i+1} is maximal it is also a maximal clique in G[v_1,v_2,…,v_i, v_i+1}]
→ (U – $\overline{N}${v_i+1}) ∪ {v_i+1} is potentially child of many possible nodes → define it als child of lexiographically smallest clique (node) U‘ at level i

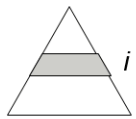


Cliques: Algorithms

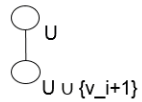
- Naive algorithm for finding the maximum clique: Exhaustive Search: Compute all subsets of V and check for clique → $O(n^2 2^n)$
- Is there a slightly better algorithm? Yes: We can improve the exponential function's base from 2 to approx. 1.38 (see [2])
- Can we expect to find the answer to the problem "how many cliques with exactly k nodes exist"? → Exhaustive search: $\Theta(|V|^k)$
- For triangles we can be better than $\Theta(|V|^3)$: An alg with $O(|V|^{2.376})$ exists ☺
- For other k: An analogous technique allows for alg with $O(|V|^{\beta(k)})$ with $\beta(k)<k$

Cliques: Enumeration

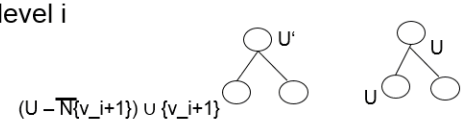
- Binary tree, n levels, leaves only at level n
- each level i ↔ vertex v_i
- nodes at level i: maximal cliques in G[v_1,v_2,…,v_i}]
- level i+1: determine children of node U at level i: two cases :



(1) v_i+1 adjacent to all nodes in U (U⊆N(v_i+1):
→ U ∪ {v_i+1} is maximal clique in G[v_1,v_2,…,v_i, v_i+1}]
→ U ∪ {v_i+1} is only child of U



(2) ∃ vertex in U not adjacent to v_i+1 (U⊈N(v_i+1):
→ U itself is a maximal clique in G[v_1,v_2,…,v_i, v_i+1}]
→ if (U – $\overline{N}${v_i+1}) ∪ {v_i+1} is maximal it is also a maximal clique in G[v_1,v_2,…,v_i, v_i+1}]
→ (U – $\overline{N}${v_i+1}) ∪ {v_i+1} is potentially child of many possible nodes → define it als child of lexiographically smallest clique (node) U‘ at level i



Cliques: Enumeration

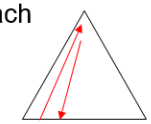
- → tree constructed in principle
- Algorithm: traverse and construct tree depth first, output leaves
- neccesary primitives:

  -- Parent(U,i): for node U at level i determine lexicographically smallest clique in G[v_1,v_2,…, v_i-1}] : O(m+n)

  -- LeftChild(U,i): either U or U ∪ {v_i+1} : O(m+n)

  -- RightChild(U,i):
  if case (1): no right child;
  if case (2): if X:=(U – N{v_i+1}) ∪ {v_i+1} is maximal (takes O(m+n) time to determine) then X is right child if U = Parent(X, i+1) (takes O(m+n))

- longest path between two leaves passes over 2n-1 nodes; for each node: O(m+n) time → maximal delay: $O(n^3)$



Cliques: Enumeration

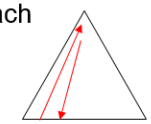
- → tree constructed in principle
- Algorithm: traverse and construct tree depth first, output leaves
- neccesary primitives:

  -- Parent(U,i): for node U at level i determine lexicographically smallest clique in G[v_1,v_2,…, v_i-1}] : O(m+n)

  -- LeftChild(U,i): either U or U ∪ {v_i+1} : O(m+n)

  -- RightChild(U,i):
  if case (1): no right child;
  if case (2): if X:=(U – N{v_i+1}) ∪ {v_i+1} is maximal (takes O(m+n) time to determine) then X is right child if U = Parent(X, i+1) (takes O(m+n))

- longest path between two leaves passes over 2n-1 nodes; for each node: O(m+n) time → maximal delay: $O(n^3)$



Cliques: Enumeration

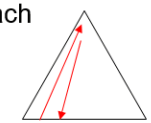
- → tree constructed in principle
- Algorithm: traverse and construct tree depth first, output leaves
- neccesary primitives:

  -- Parent(U,i): for node U at level i determine lexicographically smallest clique in G[v_1,v_2,…, v_i-1}] : O(m+n)

  -- LeftChild(U,i): either U or U ∪ {v_i+1} : O(m+n)

  -- RightChild(U,i):
  if case (1): no right child;
  if case (2): if X:=(U – N{v_i+1}) ∪ {v_i+1} is maximal (takes O(m+n) time to determine) then X is right child if U = Parent(X, i+1) (takes O(m+n))

- longest path between two leaves passes over 2n-1 nodes; for each node: O(m+n) time → maximal delay: $O(n^3)$



Plexes and Cores

- A N-Plex is maximal iff it is not strictly contained in another larger N-Plex; A N-Plex is a maximum N-Plex iff it has a maximum number of vertices among all N-Plexes in G
- N-Plexes are closed under exclusion; N-Plexes are nested → good candidates for social groups
- Furthermore: If the set of nodes V of G is an N-Plex we have: (Combinatorial Proof: see [2]):
  if N < (|V|+2) / 2 then diam(G) ≤ 2
  → If an N-Plex has a not too large "sloppyness" N "its" diameter is very small → Socially realistic
- → So far: wonderful but:

Plexes and Cores

- But the decision problem NPLEX(G,k,N): "Does G contain an N-Plex of size at least k?" is NP-complete for all N. Proof: Informal argument: CLIQUE(G,k)=NPLEX(G,k,1); formal: reduce CLIQUE(G,k) to N-PLEX (see [2]).
- Ok, so (instead of fixing how many edges can be missed at most): why not demand a minimum degree for every node in the structure?
- A subset $U \subseteq V$ is a N-Core iff $\delta(G([U])) \geq N$
- (Defs for maximal and maximum apply accordingly)
- If U is N-core, U is also a (|V|-N)-plex; Socially: If N is small compared to |U|, N-cores are not very meaningful

Plexes and Cores

- But the decision problem NPLEX(G,k,N): "Does G contain an N-Plex of size at least k?" is NP-complete for all N. Proof: Informal argument: CLIQUE(G,k)=NPLEX(G,k,1); formal: reduce CLIQUE(G,k) to N-PLEX (see [2]).
- Ok, so (instead of fixing how many edges can be missed at most): why not demand a minimum degree for every node in the structure?
- A subset $U \subseteq V$ is a N-Core iff $\delta(G([U])) \geq N$
- (Defs for maximal and maximum apply accordingly)
- If U is N-core, U is also a (|V|-N)-plex; Socially: If N is small compared to |U|, N-cores are not very meaningful

Plexes and Cores

- But the decision problem NPLEX(G,k,N): "Does G contain an N-Plex of size at least k?" is NP-complete for all N. Proof: Informal argument: CLIQUE(G,k)=NPLEX(G,k,1); formal: reduce CLIQUE(G,k) to N-PLEX (see [2]).
- Ok, so (instead of fixing how many edges can be missed at most): why not demand a minimum degree for every node in the structure?
- A subset $U \subseteq V$ is a N-Core iff $\delta(G([U])) \geq N$
- (Defs for maximal and maximum apply accordingly)
- If U is N-core, U is also a (|V|-N)-plex; Socially: If N is small compared to |U|, N-cores are not very meaningful

Plexes and Cores

- But the decision problem NPLEX(G,k,N): "Does G contain an N-Plex of size at least k?" is NP-complete for all N. Proof: Informal argument: CLIQUE(G,k)=NPLEX(G,k,1); formal: reduce CLIQUE(G,k) to N-PLEX (see [2]).
- Ok, so (instead of fixing how many edges can be missed at most): why not demand a minimum degree for every node in the structure?
- A subset $U \subseteq V$ is a N-Core iff $\delta(G([U])) \geq N$
- (Defs for maximal and maximum apply accordingly)
- If U is N-core, U is also a (|V|-N)-plex; Socially: If N is small compared to |U|, N-cores are not very meaningful

Plexes and Cores

- If U1 and U2 are N-cores, U1 U U2 is also an N-core; Socially: not very desirable; (math: → maximum N-core is unique)
- If U1 and U2 are connected maximal N-cores → U1 and U2 are disjoint; Socially: not very desirable
- N-cores are not closed under exclusion (Example: cycle is 2-core but subset not) and are generally not nested
- N-cores need not be connected
- → N-cores do not seem to be good candidates for social groups
- But: We can easily compute the maximum N-core (see [2])

Plexes and Cores

- If U1 and U2 are N-cores, U1 U U2 is also an N-core; Socially: not very desirable; (math: → maximum N-core is unique)
- If U1 and U2 are connected maximal N-cores → U1 and U2 are disjoint; Socially: not very desirable
- N-cores are not closed under exclusion (Example: cycle is 2-core but subset not) and are generally not nested
- N-cores need not be connected
- → N-cores do not seem to be good candidates for social groups
- But: We can easily compute the maximum N-core (see [2])

LS-Sets and Lambda Sets

- We will now look at some non-local concepts:
- LS-sets (Luccio-Sami-Sets): Formalize the paradigm „Intra-cluster coherence, inter-cluster decoherence“: An LS-set is a „*network region where internal ties are more significant than external ties*“ [2] (extreme version of paradigm: „strong alliance“: complete component (disconnected clique))
- A subset $U \subseteq V$ is a LS-set iff all proper subsets of U share more ties with the network outside than U does:

$$U' \subset U \rightarrow |E(U', V-U')| > |E(U, V-U)|$$

- LS-sets have some interesting properties (e.g. no trival overlaps: if $u1 \cap u2 \neq \emptyset \rightarrow u1 \subseteq u2$ or $u2 \subseteq u1$; min degree of LS set is at least half the number of outgoing edges) and may be good candidates for social groups (overlap property can be a counter argument)
- LS-sets can be computed with reasonable complexity ([2])

LS-Sets and Lambda Sets

- Lambda Sets are another related concept: In a lambda set, members are connected to other members by more (edge disjoint) paths than to outside nodes
- Let $\lambda(u,v)$ denote the number of edge-disjoint paths between nodes u and v;
A subset $U \subseteq V$ is a Lambda set iff

$$\min_{u,v \in U} \lambda(u,v) > \max_{u \in U,\, v \in V-U} \lambda(u,v)$$

- Lambda sets can be computed in P time [2]

LS-Sets and Lambda Sets

- Lambda Sets are another related concept: In a lambda set, members are connected to other members by more (edge disjoint) paths than to outside nodes

- Let λ(u,v) denote the number of edge-disjoint paths between nodes u and v;
A subset U ⊆ V is a Lambda set iff

$$\min_{u,v \in U} \lambda(u,v) > \max_{u \in U,\, v \in V-U} \lambda(u,v)$$

- Lambda sets can be computed in P time [2]

LS-Sets and Lambda Sets

- Lambda Sets are another related concept: In a lambda set, members are connected to other members by more (edge disjoint) paths than to outside nodes

- Let λ(u,v) denote the number of edge-disjoint paths between nodes u and v;
A subset U ⊆ V is a Lambda set iff

$$\min_{u,v \in U} \lambda(u,v) > \max_{u \in U,\, v \in V-U} \lambda(u,v)$$

- Lambda sets can be computed in P time [2]

Graph Clustering

- Usually: Clustering: Unsupervised Classification ("partition a set of patterns {v_i} into subsets (classes) {C_j} without training data")

- Contrast: Supervised Classifier: Train the parameters of a classifier system (e.g. the weights of a neural net, the probabilities of a naive Bayes classifier, the support weights of a support vector machine , etc.) with a pre-classified set of "training instances" {(v_train_i, C(v_train_i))}

- AI: Application of Supervised Classifiers manifold; e.g.: text-classification (Classification (supervised or unsupervised): one of the most elementary intelligent "actions"

- Usually: Clustering: In metric spaces: example: K-Means. We will come back to this in next lecture (Clustering of Profile Elements)

Graph Clustering

- Usually: Clustering: Unsupervised Classification ("partition a set of patterns {v_i} into subsets (classes) {C_j} without training data")

- Contrast: Supervised Classifier: Train the parameters of a classifier system (e.g. the weights of a neural net, the probabilities of a naive Bayes classifier, the support weights of a support vector machine , etc.) with a pre-classified set of "training instances" {(v_train_i, C(v_train_i))}

- AI: Application of Supervised Classifiers manifold; e.g.: text-classification (Classification (supervised or unsupervised): one of the most elementary intelligent "actions"

- Usually: Clustering: In metric spaces: example: K-Means. We will come back to this in next lecture (Clustering of Profile Elements)

Graph Clustering

- Usually: Clustering: Unsupervised Classification ("partition a set of patterns {v_i} into subsets (classes) {C_j} without training data")
- Contrast: Supervised Classifier: Train the parameters of a classifier system (e.g. the weights of a neural net, the probabilities of a naive Bayes classifier, the support weights of a support vector machine , etc.) with a pre-classified set of "training instances" {(v_train_i, C(v_train_i))}
- AI: Application of Supervised Classifiers manifold; e.g.: text-classification (Classification (supervised or unsupervised): one of the most elementary intelligent "actions"
- Usually: Clustering: In metric spaces: example: K-Means. We will come back to this in next lecture (Clustering of Profile Elements)

Graph Clustering

- Usually: Clustering: Unsupervised Classification ("partition a set of patterns {v_i} into subsets (classes) {C_j} without training data")
- Contrast: Supervised Classifier: Train the parameters of a classifier system (e.g. the weights of a neural net, the probabilities of a naive Bayes classifier, the support weights of a support vector machine , etc.) with a pre-classified set of "training instances" {(v_train_i, C(v_train_i))}
- AI: Application of Supervised Classifiers manifold; e.g.: text-classification (Classification (supervised or unsupervised): one of the most elementary intelligent "actions"
- Usually: Clustering: In metric spaces: example: K-Means. We will come back to this in next lecture (Clustering of Profile Elements)

Graph Clustering

- Given directed, weighted graph G=(V,E,w); A graph clustering **C**={C_1, C_2, ..., C_k} is a partition of V into non-empty subsets C_i;
- Notations:
  - E(C_i,C_j): Set of edges in G from C_i to C_j;
  - $E(\mathbf{C}) = U_{i=1,..,k}E(C_i)$: Set of intra-cluster edges;
  - $\overline{E(\mathbf{C})} = E \setminus E(\mathbf{C})$: Set of inter-cluster edges;
  - $m(\mathbf{C}) = |E(\mathbf{C})|$; $\overline{m}(\mathbf{C}) = |\overline{E(\mathbf{C})}|$;
  - G([C_i]): subgraph induced by C_i;
  - **C** with k=1: *1-clustering*; **C** with k=|V|: *singletons*; (both: *Trivial clustering);* **C** with k=2: *cut;*
  - **A**(G): Set of all possible clusterings on G;

Graph Clustering

- Given directed, weighted graph G=(V,E,w); A graph clustering **C**={C_1, C_2, ..., C_k} is a partition of V into non-empty subsets C_i;
- Notations:
  - E(C_i,C_j): Set of edges in G from C_i to C_j;
  - $E(\mathbf{C}) = U_{i=1,..,k}E(C_i)$: Set of intra-cluster edges;
  - $\overline{E(\mathbf{C})} = E \setminus E(\mathbf{C})$: Set of inter-cluster edges;
  - $m(\mathbf{C}) = |E(\mathbf{C})|$; $\overline{m}(\mathbf{C}) = |\overline{E(\mathbf{C})}|$;
  - G([C_i]): subgraph induced by C_i;
  - **C** with k=1: *1-clustering*; **C** with k=|V|: *singletons*; (both: *Trivial clustering);* **C** with k=2: *cut;*
  - **A**(G): Set of all possible clusterings on G;

Graph Clustering

- Given directed, weighted graph G=(V,E,w); A graph clustering **C**={C_1, C_2, ..., C_k} is a partition of V into non-empty subsets C_i;
- Notations:
  - E(C_i,C_j): Set of edges in G from C_i to C_j;
  - E(**C**) = $U_{i=1,..,k}$E(C_i): Set of intra-cluster edges;
  - $\overline{E(\mathbf{C})}$ = E \ E(**C**) : Set of inter-cluster edges;
  - m(**C**) = |E(**C**)|; $\overline{m}$(**C**) = $|\overline{E(\mathbf{C})}|$;
  - G([C_i]): subgraph induced by C_i;
  - **C** with k=1: *1-clustering*; **C** with k=|V|: *singletons*; (both: *Trivial clustering);* **C** with k=2: *cut;*
  - **A**(G): Set of all possible clusterings on G;

Graph Clustering

- Given directed, weighted graph G=(V,E,w); A graph clustering **C**={C_1, C_2, ..., C_k} is a partition of V into non-empty subsets C_i;
- Notations:
  - E(C_i,C_j): Set of edges in G from C_i to C_j;
  - E(**C**) = $U_{i=1,..,k}$E(C_i): Set of intra-cluster edges;
  - $\overline{E(\mathbf{C})}$ = E \ E(**C**) : Set of inter-cluster edges;
  - m(**C**) = |E(**C**)|; $\overline{m}$(**C**) = $|\overline{E(\mathbf{C})}|$;
  - G([C_i]): subgraph induced by C_i;
  - **C** with k=1: *1-clustering*; **C** with k=|V|: *singletons*; (both: *Trivial clustering);* **C** with k=2: *cut;*
  - **A**(G): Set of all possible clusterings on G;

Graph Clustering

- Given directed, weighted graph G=(V,E,w); A graph clustering **C**={C_1, C_2, ..., C_k} is a partition of V into non-empty subsets C_i;
- Notations:
  - E(C_i,C_j): Set of edges in G from C_i to C_j;
  - E(**C**) = $U_{i=1,..,k}$E(C_i): Set of intra-cluster edges;
  - $\overline{E(\mathbf{C})}$ = E \ E(**C**) : Set of inter-cluster edges;
  - m(**C**) = |E(**C**)|; $\overline{m}$(**C**) = $|\overline{E(\mathbf{C})}|$;
  - G([C_i]): subgraph induced by C_i;
  - **C** with k=1: *1-clustering*; **C** with k=|V|: *singletons*; (both: *Trivial clustering);* **C** with k=2: *cut;*
  - **A**(G): Set of all possible clusterings on G;

Graph Clustering

- Notations (continued):
  - Let **C_1**={C_1, C_2, ..., C_k} and **C_2**={C'_1, C'_2, ..., C'_l}: **C_1** ≤ **C_2** ↔ ∀i ∃j : C_i ⊆C'_j; **C_1:** *refinement of* **C_2; C_2:** *coarsening of* **C_1;**
  - *Chain* (comparable set) of clusterings: *hierarchy;*
  - Hierarchy is *total* ←→ Both trivial clusterings are contained;
  - Hierarchy that contains one clustering for each {1,2,...,|V|}: *complete;*
  - S(V): *(Proper) Cut function*: Cutting the node-set in two (non-empty) subsets: S(V) and V \ S(V);

Graph Clustering

- Notations (continued):
  - Let **C_1**={C_1, C_2, ..., C_k} and **C_2**={C'_1, C'_2, ..., C'_l}:
    C_1 ≤ **C_2** ↔ ∀i ∃j : C_i ⊆C'_j;
    C_1: *refinement of* **C_2; C_2:** *coarsening of* **C_1;**
  - *Chain* (comparable set) of clusterings: *hierarchy;*
  - Hierarchy is *total* ←→ Both trivial clusterings are contained;
  - Hierarchy that contains one clustering for each {1,2,...,|V|}: *complete;*
  - S(V): *(Proper) Cut function*: Cutting the node-set in two (non-empty) subsets: S(V) and V \ S(V);

Quality Measures for Clusterings

- Quality measure: Objective function **A**(G) → $\mathbb{R}$ that formalizes the clustering paradigm in a special way
- G= (V,E,w): Weight function w: E→ $\mathbb{R}^+$ is interpreted as "similarity" (higher weights correspond to more intense tie); also possible: negative weights = dissimilarity; or w: E → [0,1] or w: E → [-1,1] etc.
- Distinguish between no edge and edge with weight zero;
- Notation: $w(E) = \sum_{e \in E} w(e)$

Quality Measures for Clusterings

- Quality measure: Objective function **A**(G) → $\mathbb{R}$ that formalizes the clustering paradigm in a special way
- G= (V,E,w): Weight function w: E→ $\mathbb{R}^+$ is interpreted as "similarity" (higher weights correspond to more intense tie); also possible: negative weights = dissimilarity; or w: E → [0,1] or w: E → [-1,1] etc.
- Distinguish between no edge and edge with weight zero;
- Notation: $w(E) = \sum_{e \in E} w(e)$

6: Graph Clustering

- General framework for a quality index of a clustering:

$$index(\mathbf{C}) = \frac{f(\mathbf{C}) + g(\mathbf{C})}{\max\{f(\mathbf{C}') + g(\mathbf{C}') : \mathbf{C}' \in \mathbf{A}(G)\}}$$

- f: **A**(G) → $\mathbb{R}^+$ measures intra cluster density (coherence);
  g : **A**(G) → $\mathbb{R}^+$ measures inter cluster sparseness (decoherence);

- General framework for a quality index of a clustering:

$$index(\mathbf{C}) = \frac{f(\mathbf{C}) + g(\mathbf{C})}{\max\{f(\mathbf{C}') + g(\mathbf{C}') : \mathbf{C}' \in \mathbf{A}(G)\}}$$

- f : **A**(G) → $\mathbb{R}^+$ measures intra cluster density (coherence);
  g : **A**(G) → $\mathbb{R}^+$ measures inter cluster sparseness (decoherence);

6: Graph Clustering

- General framework for a quality index of a clustering:

$$index(\mathbf{C}) = \frac{f(\mathbf{C}) + g(\mathbf{C})}{\max\{f(\mathbf{C}') + g(\mathbf{C}') : \mathbf{C}' \in \mathbf{A}(G)\}}$$

- f : **A**(G) → $\mathbb{R}^+$ measures intra cluster density (coherence);
  g : **A**(G) → $\mathbb{R}^+$ measures inter cluster sparseness (decoherence);

Coverage

- First quality measure: Coverage

$$\gamma(\mathbf{C}) = \frac{w(E(\mathbf{C}))}{w(E)} = \frac{\sum_{e \in E(\mathbf{C})} w(e)}{\sum_{e \in E} w(e)}$$

- Thus: f = w(E(**C**)) and g = 0; → only accumulated intra cluster density is measured

- Maximum value 1 achieved for **C**={V} (1-clustering)

- A clustering has coverage γ(**C**)=1 iff $\overline{E(\mathbf{C})}$=∅ (clustering is composed of connected components of G) or $\overline{w(E(\mathbf{C}))}$=0

- **C** with k>1 can be transformed into **C**´ with k´<k and $\gamma(\mathbf{C}) \leq \gamma(\mathbf{C}')$ by merging two clusters in **C** → optimal non-trivial **C** is a minimum cut

- → „Monotonic“ behavior of coverage → coverage is not a good sole quality index

Coverage

- First quality measure: Coverage

$$\gamma(\mathbf{C}) = \frac{w(E(\mathbf{C}))}{w(E)} = \frac{\sum_{e \in E(\mathbf{C})} w(e)}{\sum_{e \in E} w(e)}$$

- Thus: f = w(E(**C**)) and g = 0; → only accumulated intra cluster density is measured

- Maximum value 1 achieved for **C**={V} (1-clustering)

- A clustering has coverage γ(**C**)=1 iff $\overline{E(\mathbf{C})}$=∅ (clustering is composed of connected components of G) or $\overline{w(E(\mathbf{C}))}$=0

- **C** with k>1 can be transformed into **C**´ with k´<k and $\gamma(\mathbf{C}) \leq \gamma(\mathbf{C}')$ by merging two clusters in **C** → optimal non-trivial **C** is a minimum cut

- → „Monotonic“ behavior of coverage → coverage is not a good sole quality index

Conductance

- Clustering paradigm reformulated: Clusters should be well connected (many edges need to be removed to make it unconnected); few inter cluster edges (ideally none)
- Conductance: Measure for bottlenecks (Bottleneck: Cut that separates V into roughly same size halves and “cuts across” relatively few edges)
- Let **C**={C_1, V \ C_1} be a cut. Conductance φ of **C** is defined as

$$\varphi(\mathbf{C}) = \begin{cases} 1 & \text{if } C\_1 \in \{\emptyset, V\} \\ 0 & \text{if } C\_1 \notin \{\emptyset, V\},\ w(\overline{E(C)})=0 \\ \dfrac{w(\overline{E(\mathbf{C})})}{\min(\sum_{e \in E(C\_1, V)} w(e), \sum_{e \in E(V \setminus C\_1, V)} w(e))} & \text{otherwise} \end{cases}$$

the smaller $\varphi(\mathbf{C})$, the more „bottlenecky“ is **C**

Conductance

- Clustering paradigm reformulated: Clusters should be well connected (many edges need to be removed to make it unconnected); few inter cluster edges (ideally none)
- Conductance: Measure for bottlenecks (Bottleneck: Cut that separates V into roughly same size halves and “cuts across” relatively few edges)
- Let **C**={C_1, V \ C_1} be a cut. Conductance φ of **C** is defined as

$$\varphi(\mathbf{C}) = \begin{cases} 1 & \text{if } C\_1 \in \{\emptyset, V\} \\ 0 & \text{if } C\_1 \notin \{\emptyset, V\},\ w(\overline{E(C)})=0 \\ \dfrac{w(\overline{E(\mathbf{C})})}{\min(\sum_{e \in E(C\_1, V)} w(e), \sum_{e \in E(V \setminus C\_1, V)} w(e))} & \text{otherwise} \end{cases}$$

the smaller $\varphi(\mathbf{C})$, the more „bottlenecky“ is **C**

Conductance

- Clustering paradigm reformulated: Clusters should be well connected (many edges need to be removed to make it unconnected); few inter cluster edges (ideally none)
- Conductance: Measure for bottlenecks (Bottleneck: Cut that separates V into roughly same size halves and “cuts across” relatively few edges)
- Let **C**={C_1, V \ C_1} be a cut. Conductance φ of **C** is defined as

$$\varphi(\mathbf{C}) = \begin{cases} 1 & \text{if } C\_1 \in \{\emptyset, V\} \\ 0 & \text{if } C\_1 \notin \{\emptyset, V\},\ w(\overline{E(C)})=0 \\ \dfrac{w(\overline{E(\mathbf{C})})}{\min(\sum_{e \in E(C\_1, V)} w(e), \sum_{e \in E(V \setminus C\_1, V)} w(e))} & \text{otherwise} \end{cases}$$

the smaller $\varphi(\mathbf{C})$, the more „bottlenecky“ is **C**

Conductance

- Clustering paradigm reformulated: Clusters should be well connected (many edges need to be removed to make it unconnected); few inter cluster edges (ideally none)
- Conductance: Measure for bottlenecks (Bottleneck: Cut that separates V into roughly same size halves and “cuts across” relatively few edges)
- Let **C**={C_1, V \ C_1} be a cut. Conductance φ of **C** is defined as

$$\varphi(\mathbf{C}) = \begin{cases} 1 & \text{if } C\_1 \in \{\emptyset, V\} \\ 0 & \text{if } C\_1 \notin \{\emptyset, V\},\ w(\overline{E(C)})=0 \\ \dfrac{w(\overline{E(\mathbf{C})})}{\min(\sum_{e \in E(C\_1, V)} w(e), \sum_{e \in E(V \setminus C\_1, V)} w(e))} & \text{otherwise} \end{cases}$$

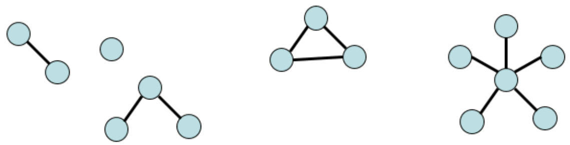
the smaller $\varphi(\mathbf{C})$, the more „bottlenecky“ is **C**

Conductance

- Conductance φ of G is defined as

$$\varphi(G) = \min_{C\_1 \subseteq V} \varphi(C\_1,\ V \setminus C\_1)$$

- Small conductance ←→"good cut possible"
- All unconnected graphs have conductance 0
- Theorem: If G is undirected and positively weighted, G has maximum conductance φ(G)=1 iff G is connected and has at most three nodes or is a star.

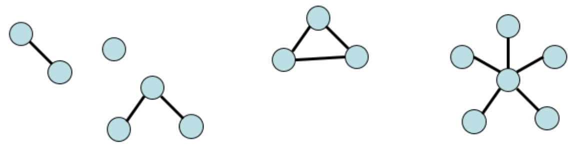


Conductance

- Conductance φ of G is defined as

$$\varphi(G) = \min_{C\_1 \subseteq V} \varphi(C\_1,\ V \setminus C\_1)$$

- Small conductance ←→"good cut possible"
- All unconnected graphs have conductance 0
- Theorem: If G is undirected and positively weighted, G has maximum conductance φ(G)=1 iff G is connected and has at most three nodes or is a star.

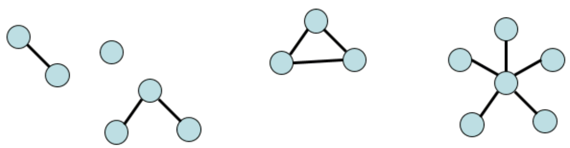


Conductance

- Conductance φ of G is defined as

$$\varphi(G) = \min_{C\_1 \subseteq V} \varphi(C\_1,\ V \setminus C\_1)$$

- Small conductance ←→"good cut possible"
- All unconnected graphs have conductance 0
- Theorem: If G is undirected and positively weighted, G has maximum conductance φ(G)=1 iff G is connected and has at most three nodes or is a star.

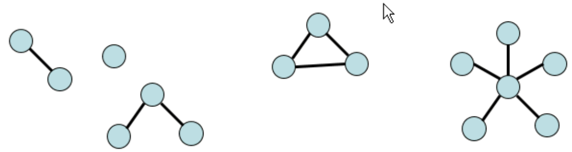


Conductance

- Conductance φ of G is defined as

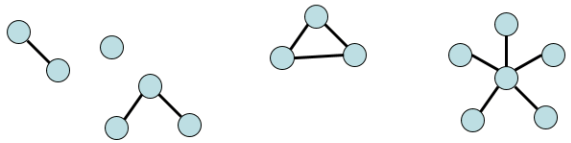
$$\varphi(G) = \min_{C\_1 \subseteq V} \varphi(C\_1,\ V \setminus C\_1)$$

- Small conductance ←→"good cut possible"
- All unconnected graphs have conductance 0
- Theorem: If G is undirected and positively weighted, G has maximum conductance φ(G)=1 iff G is connected and has at most three nodes or is a star.



Conductance

- Theorem: If G is undirected and positively weighted, G has maximum conductance φ(G)=1 iff G is connected and has at most three nodes or is a star. (Proof: see [1])



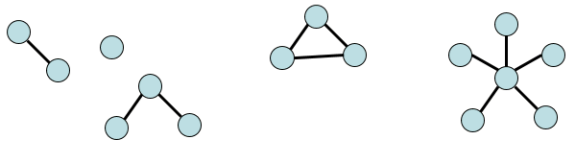
- Proof `←‘ $\sum_{e \in E(C\_1, V)} w(e) = w(E(C\_1)) + w(\overline{E(C)})$ →

$$\frac{w(\overline{E(C)})}{\min(\sum_{e \in E(C\_1, V)} w(e), \sum_{e \in E(V \setminus C\_1, V)} w(e))} =$$

$$\frac{w(\overline{E(C)})}{w(\overline{E(C)}) + \min(\underbrace{w(E(C\_1))}_{=0 \text{ if star or at most 3 nodes}}, w(E(V \setminus C\_1)))} = 1$$

Conductance

- Theorem: If G is undirected and positively weighted, G has maximum conductance φ(G)=1 iff G is connected and has at most three nodes or is a star. (Proof: see [1])



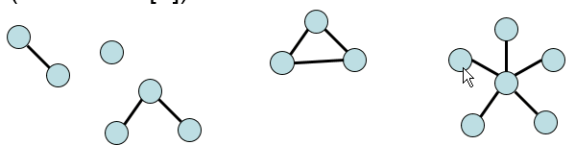
- Proof `←‘ $\sum_{e \in E(C\_1, V)} w(e) = w(E(C\_1)) + w(\overline{E(C)})$ →

$$\frac{w(\overline{E(C)})}{\min(\sum_{e \in E(C\_1, V)} w(e), \sum_{e \in E(V \setminus C\_1, V)} w(e))} =$$

$$\frac{w(\overline{E(C)})}{w(\overline{E(C)}) + \min(\underbrace{w(E(C\_1))}_{=0 \text{ if star or at most 3 nodes}}, w(E(V \setminus C\_1)))} = 1$$

Conductance

- Theorem: If G is undirected and positively weighted, G has maximum conductance φ(G)=1 iff G is connected and has at most three nodes or is a star. (Proof: see [1])



- Proof `←‘ $\sum_{e \in E(C\_1, V)} w(e) = w(E(C\_1)) + w(\overline{E(C)})$ →

$$\frac{w(\overline{E(C)})}{\min(\sum_{e \in E(C\_1, V)} w(e), \sum_{e \in E(V \setminus C\_1, V)} w(e))} =$$

$$\frac{w(\overline{E(C)})}{w(\overline{E(C)}) + \min(\underbrace{w(E(C\_1))}_{=0 \text{ if star or at most 3 nodes}}, w(E(V \setminus C\_1)))} = 1$$

Conductance

- Conductance appears to be a good element for a quality measure, but: calculating it is NP hard ☹ but: it can be approximated with guarantee $O((\log |V|)^{1/2})$ (see [1]) (This means: approximation * $O((\log |V|)^{1/2})$ = true value)

- The conductance of a complete graph is asymptotically 0.5: Let n be an integer:

$$\varphi(K_n) = \begin{cases} 0.5\dfrac{n}{n-1} & \text{if } n \text{ is even} \\ 0.5 + \dfrac{1}{n-1} & \text{if } n \text{ is odd} \end{cases}$$

Conductance

- Conductance appears to be a good element for a quality measure, but: calculating it is NP hard ☹ but: it can be approximated with guarantee $O((\log |V|)^{1/2})$ (see [1]) (This means: approximation * $O((\log |V|)^{1/2})$ = true value)

- The conductance of a complete graph is asymptotically 0.5: Let n be an integer:

$$\varphi(K_n) = \begin{cases} 0.5\dfrac{n}{n-1} & \text{if } n \text{ is even} \\ 0.5+\dfrac{1}{n-1} & \text{if } n \text{ is odd} \end{cases}$$

Conductance

- With conductance we can define two appropriate quality measures for clusterings:

- First measure: g=0 and $f(\mathbf{C}) = \min_{1\le i\le k} \varphi(G[\text{C\_i}])$

- If the measure is small: At least one of the clusters (more precisely: the induced subgraph) contains at least one bottleneck → This cluster is too coarse →Use minimum conductance cut to cut this cluster in "halves"

- From theorem before: Only clusterings where the clusters induce subgraphs that are stars or have size at most three have f=1 (f is called *intra cluster conductance*)

Conductance

- With conductance we can define two appropriate quality measures for clusterings:

- First measure: g=0 and $f(\mathbf{C}) = \min_{1\le i\le k} \varphi(G[\text{C\_i}])$

- If the measure is small: At least one of the clusters (more precisely: the induced subgraph) contains at least one bottleneck → This cluster is too coarse →Use minimum conductance cut to cut this cluster in "halves"

- From theorem before: Only clusterings where the clusters induce subgraphs that are stars or have size at most three have f=1 (f is called *intra cluster conductance*)

Conductance

- With conductance we can define two appropriate quality measures for clusterings:

- First measure: g=0 and $f(\mathbf{C}) = \min_{1\le i\le k} \varphi(G[\text{C\_i}])$

- If the measure is small: At least one of the clusters (more precisely: the induced subgraph) contains at least one bottleneck → This cluster is too coarse →Use minimum conductance cut to cut this cluster in "halves"

- From theorem before: Only clusterings where the clusters induce subgraphs that are stars or have size at most three have f=1 (f is called *intra cluster conductance*)