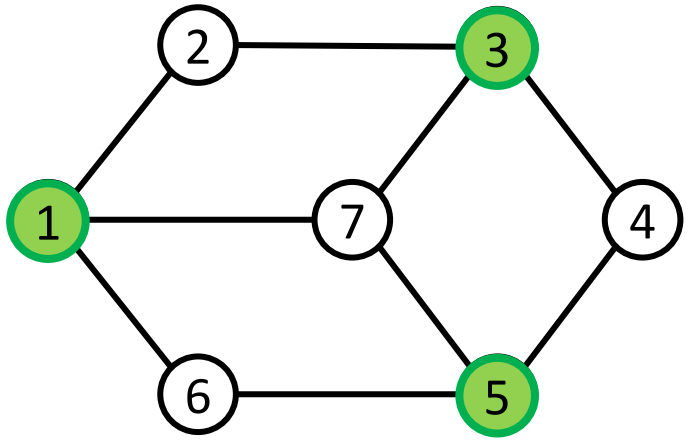
Algoritmi approssimanti per problemi NP Completi: Vertex Cover

Corso Ottimizzazione Combinatoria (IN440)

Prof. M. Liverani

Vertex cover

- Dato un grafo casuale $G = (V, E)$ non orientato con n vertici, vogliamo calcolare con un procedimento di ricerca esaustiva definito con l'algoritmo **VertexCoverEsaustivo** una **copertura di vertici di cardinalità minima** per G e poi confrontare questa soluzione con la soluzione ottenuta con l'algoritmo approssimante **VertexCoverApprossimato**
- Una copertura di vertici C per G è un sottoinsieme di vertici di $V(G)$ tale che ogni spigolo del grafo ha almeno uno dei due estremi in C



Ricerca esaustiva della soluzione ottima

L'algoritmo di ricerca esaustiva della soluzione è il seguente; l'enumerazione di tutti i sottoinsiemi C di V si ottiene con l'algoritmo di costruzione delle stringhe binarie con n cifre:

se $S_i = 1$ allora $v_i \in C$, altrimenti $v_i \notin C$

Algoritmo 53 VERTEXCOVERESAUSTIVO(G)

Input: Un grafo $G, = (V, E)$
Output: Una copertura di vertici $C \subseteq V$ di cardinalità minima

```
1: C_min = V
2: per ogni C ∈ P(V) ripeti
3:     flag = true
4:     per ogni e ∈ E(G) e flag = true ripeti
5:         sia e = (u, v)
6:         se u ∉ C e v ∉ C allora
7:             flag = false
8:         fine-condizione
9:     fine-ciclo
10:    se flag = true e |C| < |C_min| allora
11:        C_min = C
12:    fine-condizione
13: fine-ciclo
14: return C_min
```

Algoritmo 5 STRINGHEBINARIE(n)

Input: Un numero intero $n > 0$
Output: Tutte le stringhe binarie $S := (s_1, s_2, \ldots, s_n)$ con n cifre

```
1: sia S := (0, 0, ..., 0) (s_i := 0 per ogni i = 1, 2, ..., n)
2: i := 1
3: fintanto che i ≤ n ripeti
4:     i := 1
5:     fintanto che i ≤ n e s_i ≠ 0 ripeti
6:         s_i := 0
7:         i := i + 1
8:     fine-ciclo
9:     se i ≤ n allora
10:        s_i := 1
11:    fine-condizione
12:    scrivi la stringa S = (s_1, s_2, ..., s_n)
13: fine-ciclo
```

Ricerca esaustiva della soluzione ottima

```
C = np.zeros((n+1), dtype=int)
Cmin = np.full((n+1), 1, dtype=int)
min = n
i = 1
while i <= n:
  i = 1
  while i <= n and C[i] != 0:
    C[i] = 0
    i = i+1
  if i <= n: C[i] = 1
  flag = True
  for u in g:
    for v in u.getConnections():
      if C[u.getId()] == 0 and C[v.getId()] == 0:
        flag = False
  if flag == True:
    m = 0
    for k in range(n+1):
      m = m + C[k]
    if m < min:
      for k in range(n+1):
        Cmin[k] = C[k]
      min = m
```

[... segue ...]

```
print("Copertura ottima:")
for k in range(1, n+1):
  if Cmin[k] == 1:
    print(k, end=" ")
print()
```

Ricerca soluzione approssimata

- La soluzione approssimata (di cardinalità al più doppia di quella ottima) può essere calcolata con il seguente algoritmo

Algoritmo 41 VertexCover(G)

1: $C = \emptyset$, $E' = E(G)$
2: **fintanto che** $E' \neq \emptyset$ **ripeti**
3: sia $(u, v) \in E'$ uno spigolo arbitrario
4: $C = C \cup \{u, v\}$
5: rimuovi da E' ogni spigolo incidente su u o v
6: **fine-ciclo**
7: **per ogni** $u \in C$ in ordine di grado non decrescente su G **ripeti**
8: **se** ogni v adiacente a u è in C **allora**
9: rimuovi il vertice u da C
10: **fine-condizione**
11: **fine-ciclo**

Ricerca soluzione approssimata

- La soluzione approssimata (di cardinalità al più doppia di quella ottima) può essere calcolata con il seguente algoritmo

```
c = list()
e = list()
d = np.zeros((n+1), dtype=int)
for u in g:
  for v in u.getConnections():
    e.append((u.getId(), v.getId()))
    d[u.getId()] += 1
e1 = list()
while len(e) > 0:
  (u,v) = e.pop()
  c.append(u)
  c.append(v)
  e1 = e.copy()
  for (h,k) in e:
    if h == u or h == v or k == u or k == v:
      e1.remove((h,k))
      e = e1.copy()
print("c: ", c)
```

```
for i in range(1,n):
  for j in range(1, n+1):
    if j in c and d[j] == i:
      u = g.getVertex(j)
      flag = True
      for v in u.getConnections():
        if v.getId() not in c:
          flag = False
      if flag:
        c.remove(j)
print("c: ", c)
```

Mettiamo insieme i pezzi

```
from in440 import *
g = Graph()
n = int(input("Numero di vertici: "))
p = float(input("Probabilita': "))
randomGraph(g, n, p)
c = list()
e = list()
d = np.zeros((n+1), dtype=int)
C = np.zeros((n+1), dtype=int)
Cmin = np.full((n+1), 1, dtype=int)
min = n
i = 1
while i <= n:
  i = 1
  while i <= n and C[i] != 0:
    C[i] = 0
    i = i+1
  if i <= n: C[i] = 1
  flag = True
  for u in g:
    for v in u.getConnections():
      if C[u.getId()] == 0 and C[v.getId()] == 0:
        flag = False
  if flag == True:
    m = 0
    for k in range(n+1):
      m = m + C[k]
    if m < min:
      Cmin = C.copy()
      min = m
```

[...segue...]

```
print("Copertura ottima:")
for k in range(1, n+1):
  if Cmin[k] == 1:
    print(k, end=" ")
print()
for u in g:
  for v in u.getConnections():
    e.append((u.getId(), v.getId()))
    d[u.getId()] += 1
e1 = list()
while len(e) > 0:
  (u,v) = e.pop()
  c.append(u)
  c.append(v)
  e1 = e.copy()
  for (h,k) in e:
    if h == u or h == v or k == u or k == v:
      e1.remove((h,k))
      e = e1.copy()
print("c: ", c)
for i in range(1,n):
  for j in range(1, n+1):
    if j in c and d[j] == i:
      u = g.getVertex(j)
      flag = True
      for v in u.getConnections():
        if v.getId() not in c:
          flag = False
      if flag:
        c.remove(j)
print("c: ", c)
printGraph(g)
plotGraph(g, "Grafo", "non orientato", True)
```