# Algoritmi di approssimazione per trasversali connessi minimi

Relatore: Marco Di Summa
Correlatore: Manuel Francesco Aprile
Laureando: Marco Furlan
15 dicembre 2021


Università degli Studi di Padova

## Trasversale/Vertex Cover/VC

Un trasversale di un grafo $G = (V, E)$ è un insieme di vertici $C \subset V$ tale che per ogni arco $(u, v) \in E$ uno dei due estremi appartiene a C ($u \in C$ oppure $v \in C$).

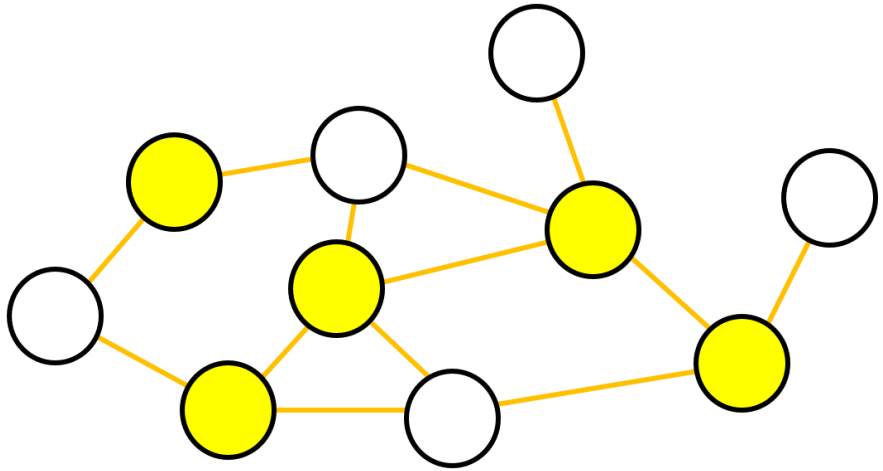


Figura – Trasversale (non connesso)

## Trasversale connesso/Connected Vertex Cover/CVC

Un trasversale il cui sottografo indotto è connesso.

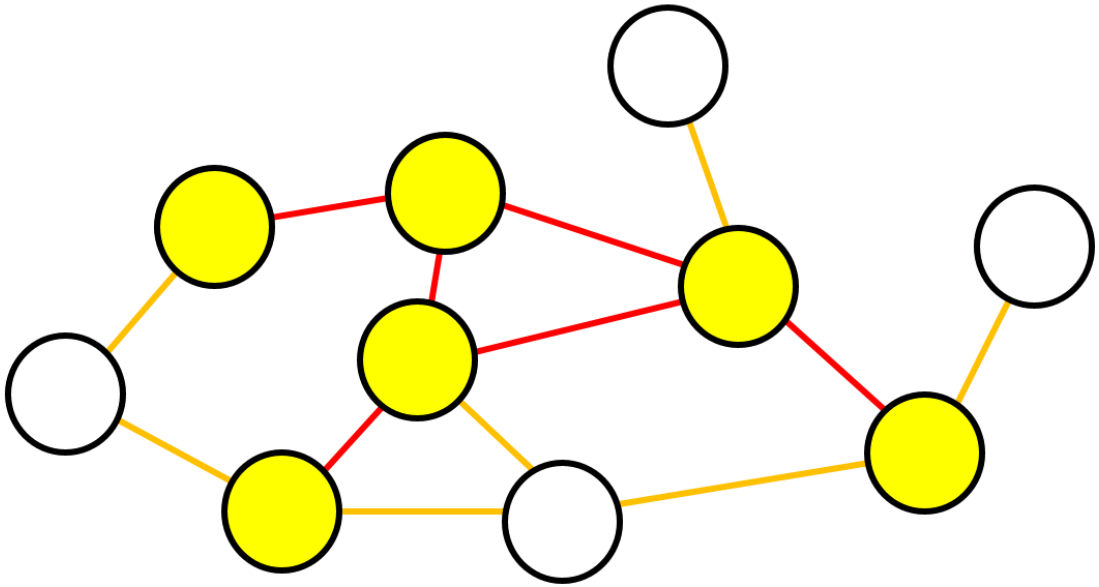


Figura – Trasversale connesso

# Introduzione



Minimizzare VC :

- NP-completo
- ammette algoritmi di 2-approssimazione
- largamente studiato in informatica teorica

Minimizzare CVC :

- NP-completo
- ammette algoritmi di 2-approssimazione
- finora meno studiato

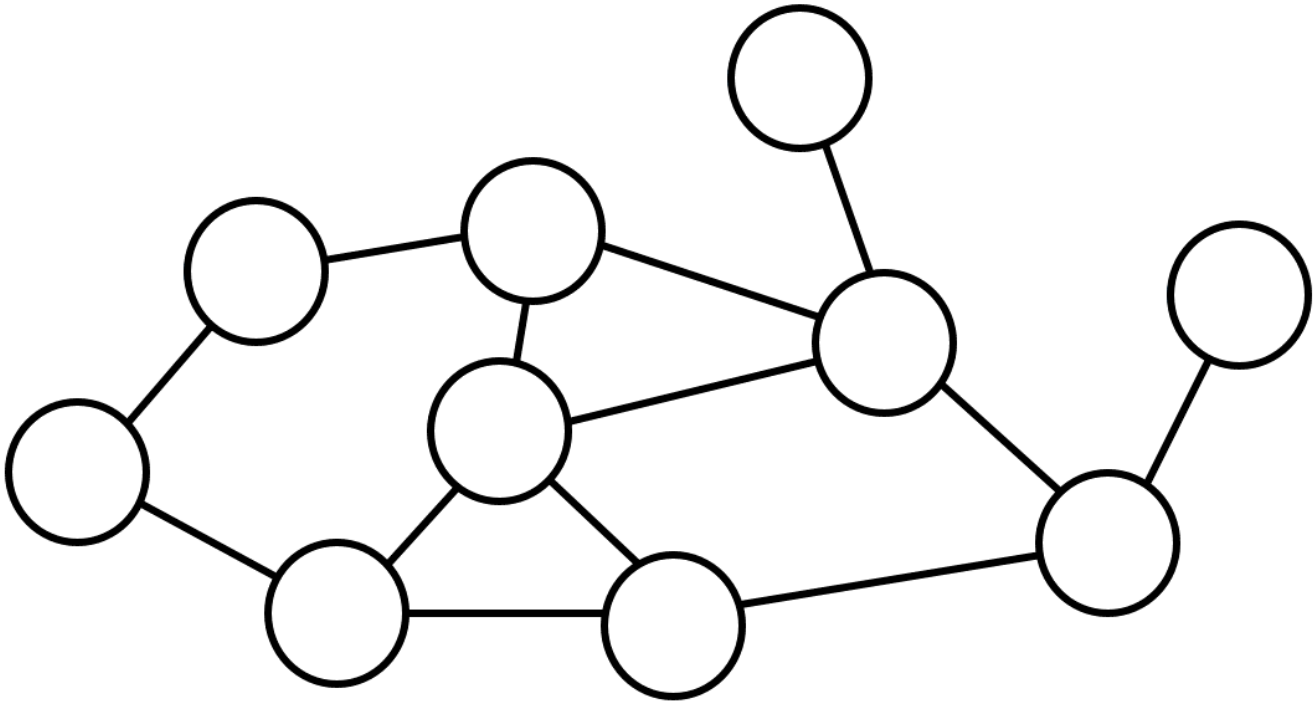
# DFS-CVC



Vediamo un primo algoritmo di 2-approssimazione per CVC, il DFS-CVC (depth-first search CVC) di *C.Savage*[1].

[1] C.Savage, *Depth-First Search and the Vertex Cover Problem*, 1982

Prendiamo come esempio il seguente grafo connesso :



Applichiamo il DFS :

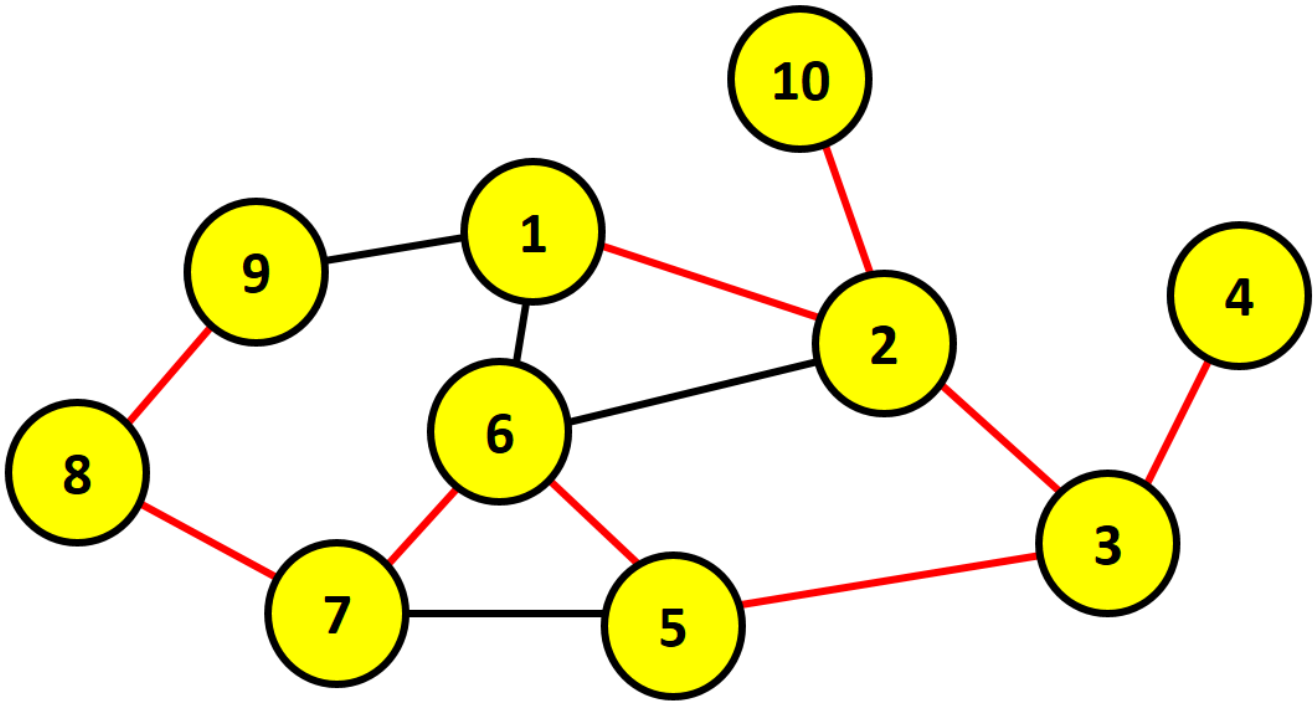
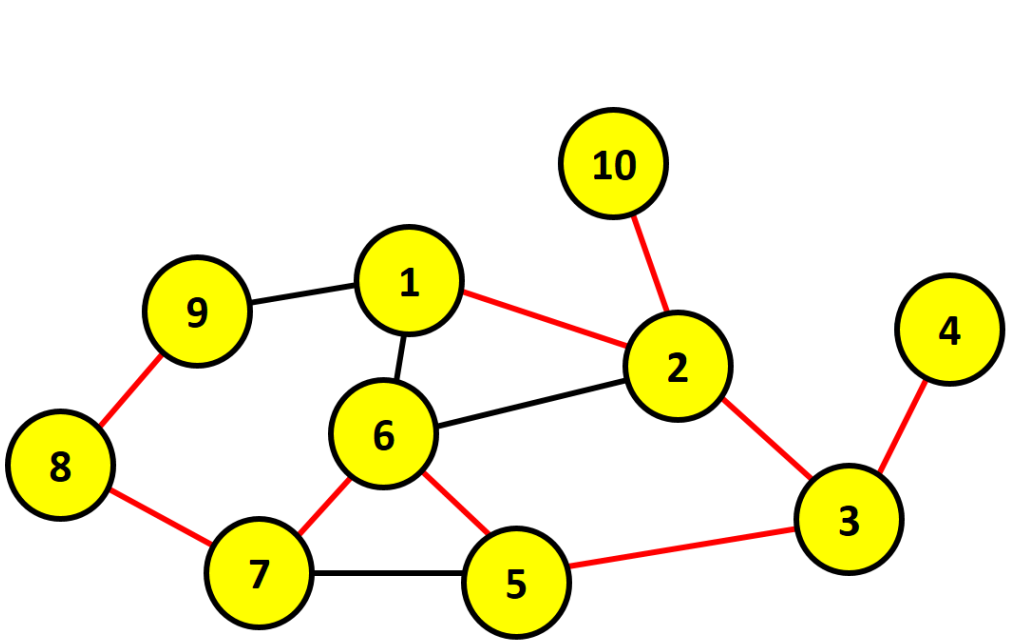
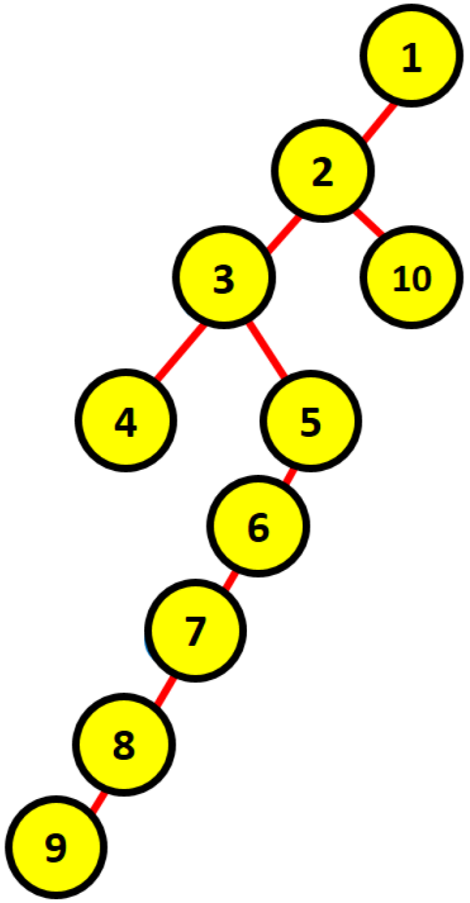


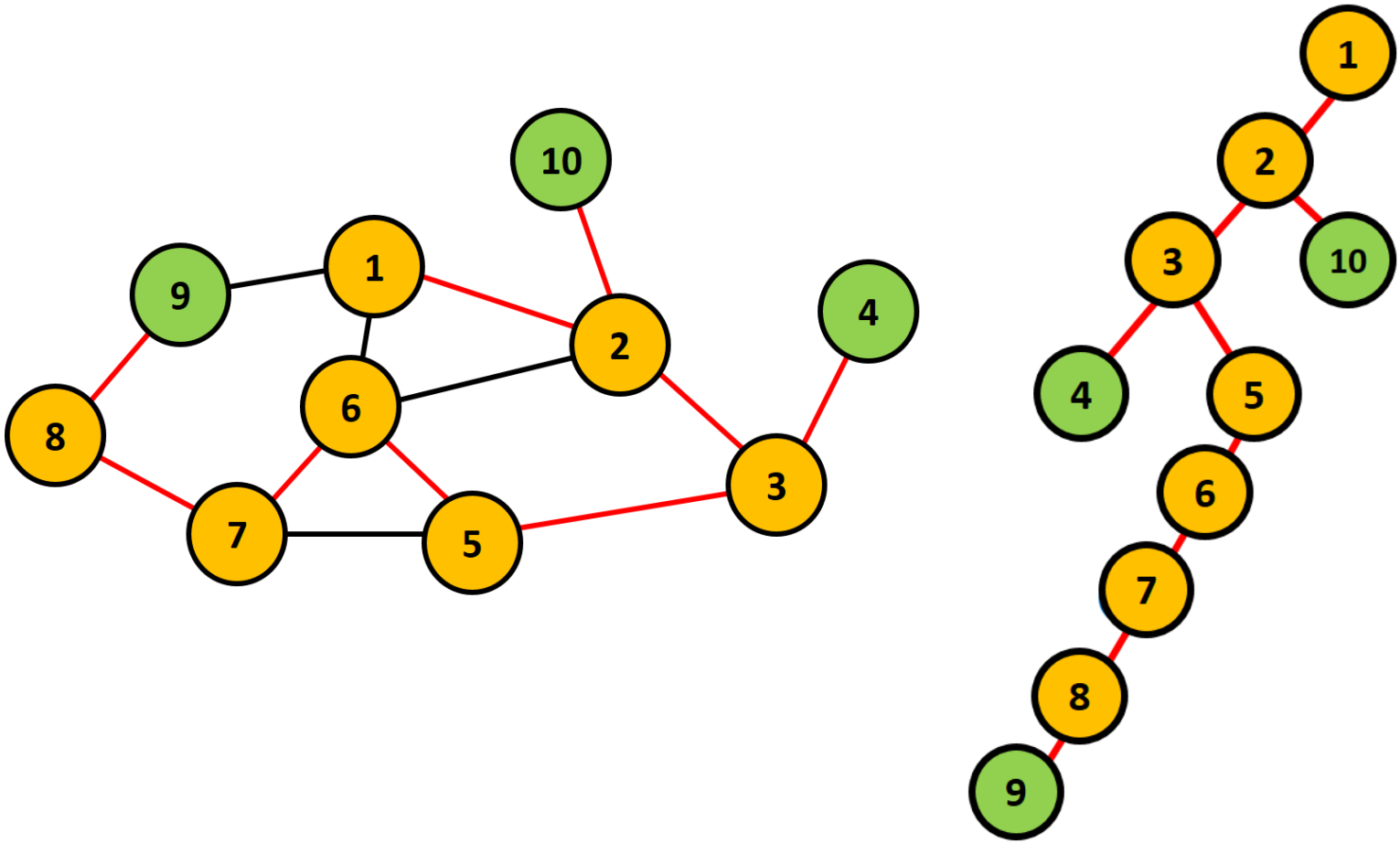

Figura – I nodi sono numerati in ordine di visita : si parte dal nodo 1 e si procede tenendo il nodo adiacente più a destra

Consideriamo l'albero ricoprente DFS

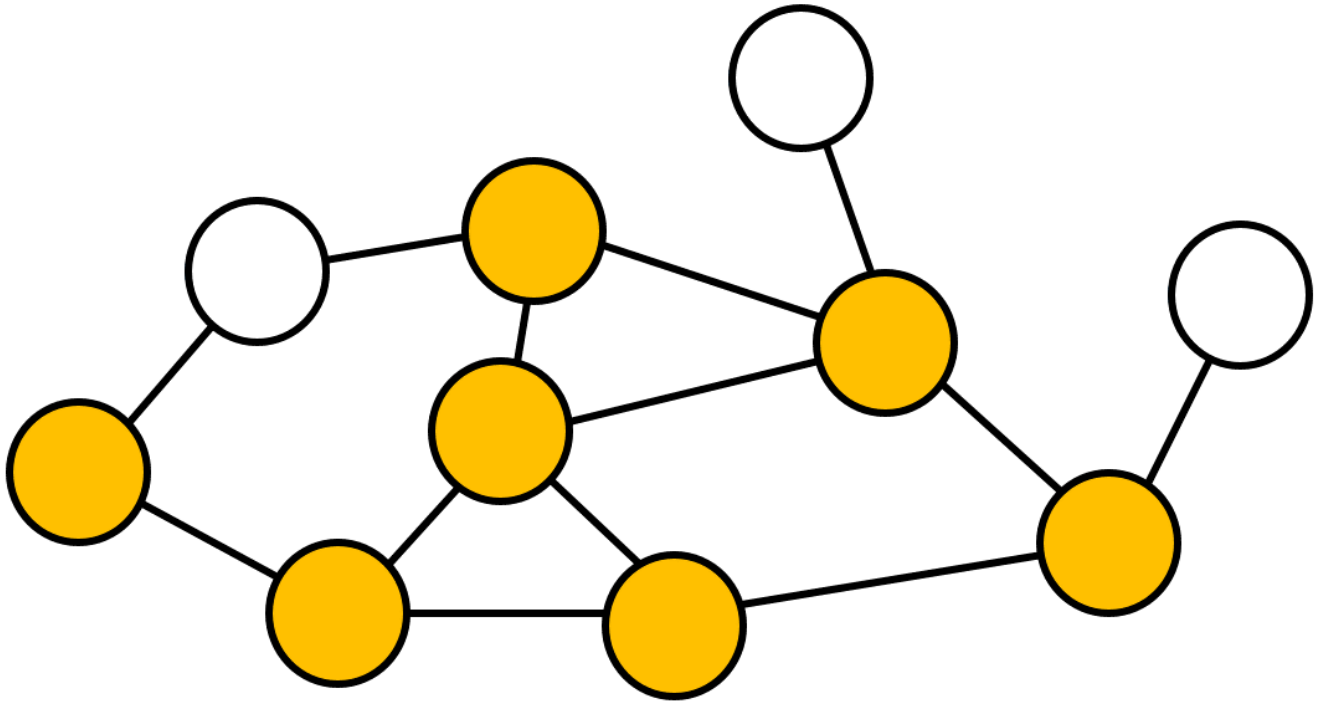


Selezioniamo i vertici interni (arancione, in verde le foglie)



L'insieme dei nodi interni forma una CVC !



Questo algoritmo ha la stesso tempo di esecuzione del DFS, cioè $O(|V| + |E|)$

# GRASP-CVC



Il GRASP-CVC è un'euristica descritta nell'articolo [2] che ci consente di trovare CVC

- ha tempo di esecuzione maggiore del DFS-CVC, cioè $O(|V| \cdot |E|)$
- è molto più efficace nel trovare CVC più piccole

Più avanti confronteremo tempi e risultati dei due algoritmi

[2] Y.Zhang, J.Wu, L.Zhang, P.Zhao, J.Zhou, M.Yin, *An Efficient Heuristic Algorithm for Solving Connected Vertex Cover Problems*, 2018

# GRASP-CVC



Il GRASP-CVC è composto da due parti, *GreedyConstruction* e *LocalSearch*. Funziona così :

- La *GreedyConstruction* costruisce una soluzione greedy,
- La *LocalSearch* cerca soluzioni migliori a partire da quella trovata con la *GreedyConstruction*,
- si eseguono *GreedyConstruction* e *LocalSearch* per diverse iterazioni e si ritorna la soluzione migliore.

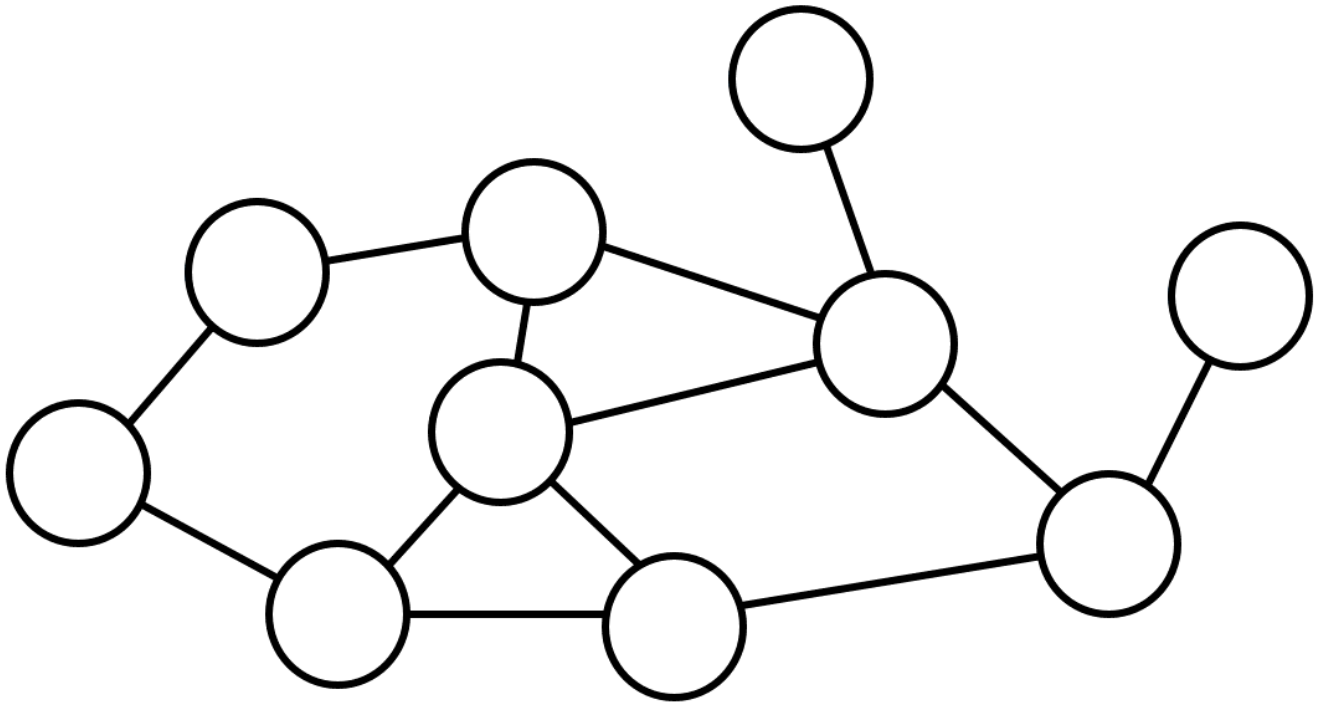
# GRASP-CVC, *GreedyConstruction*



La *GreedyConstruction* è un algoritmo che costruisce una soluzione greedy. Parte da un vertice di grado massimo, e aggiunge di volta in volta vertici vicini che coprono più archi, fino ad ottenere una VC.

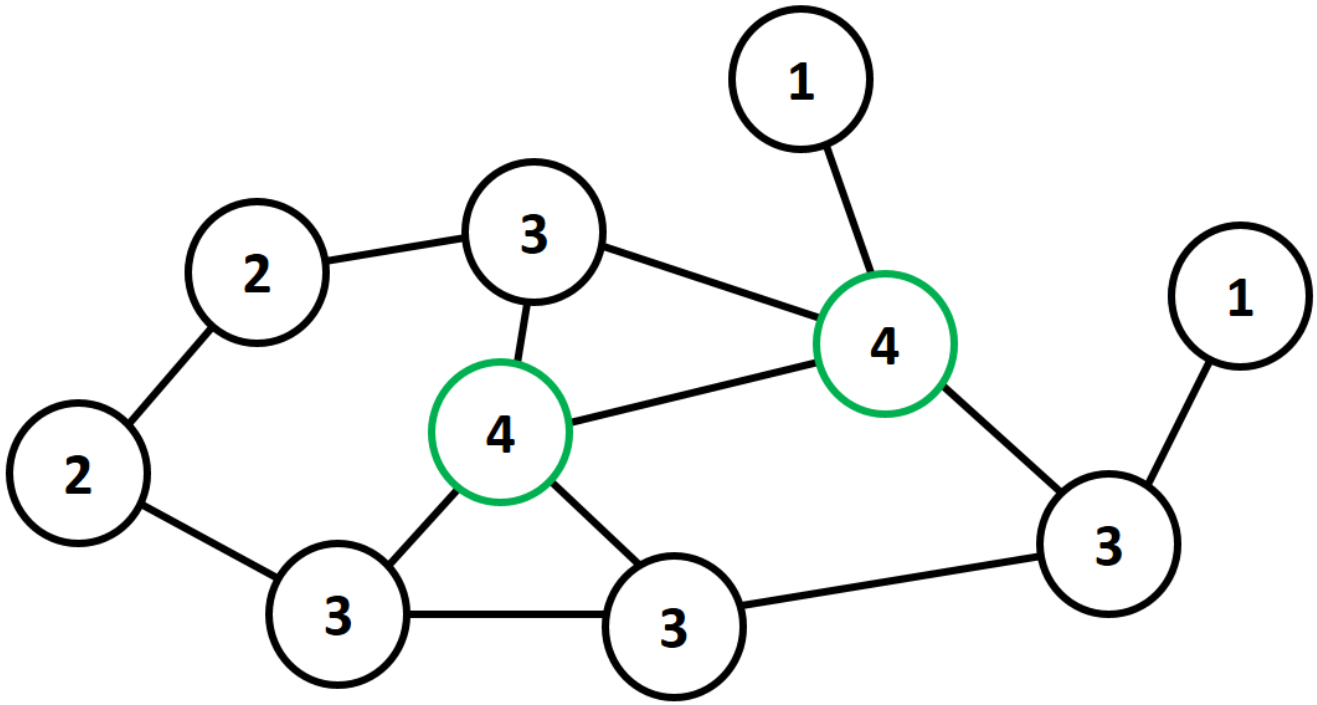
**Esempio di GreedyConstruction**

Vediamo la *GreedyConstruction* applicata a questo grafo :



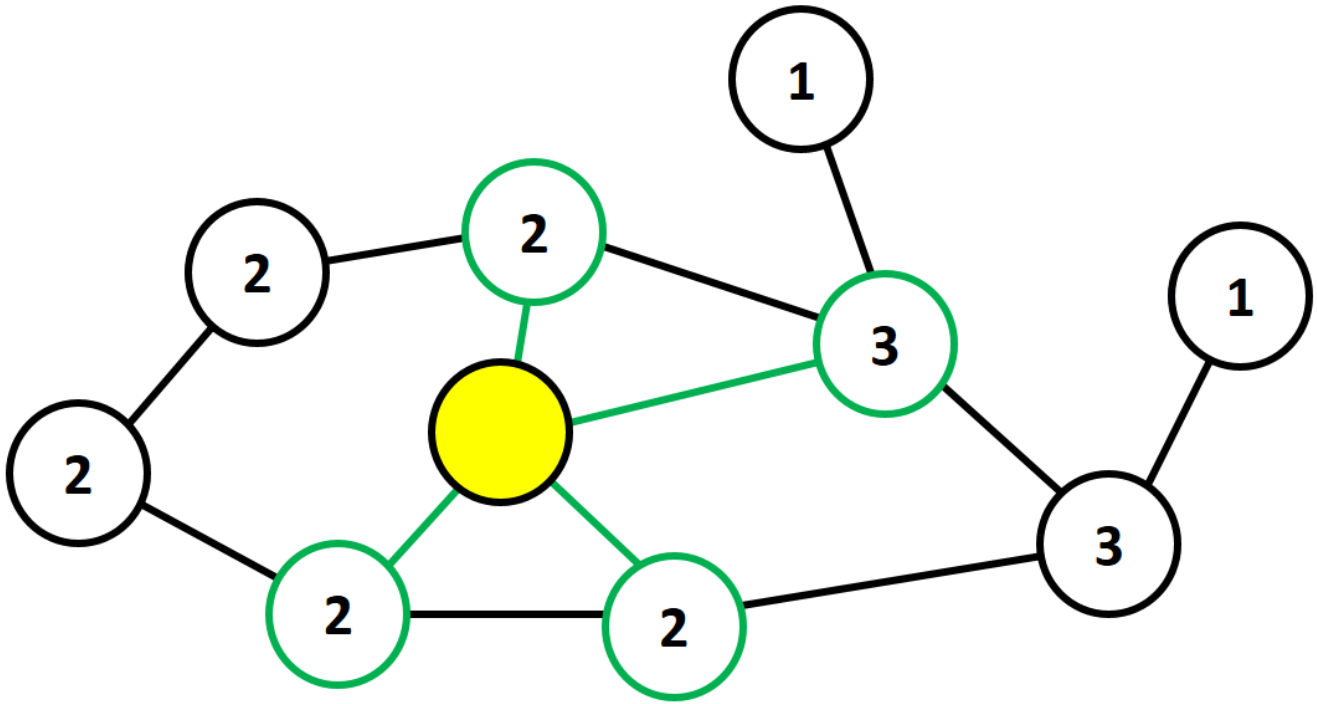
**Esempio di GreedyConstruction**

Selezioniamo i vertici di grado massimo :



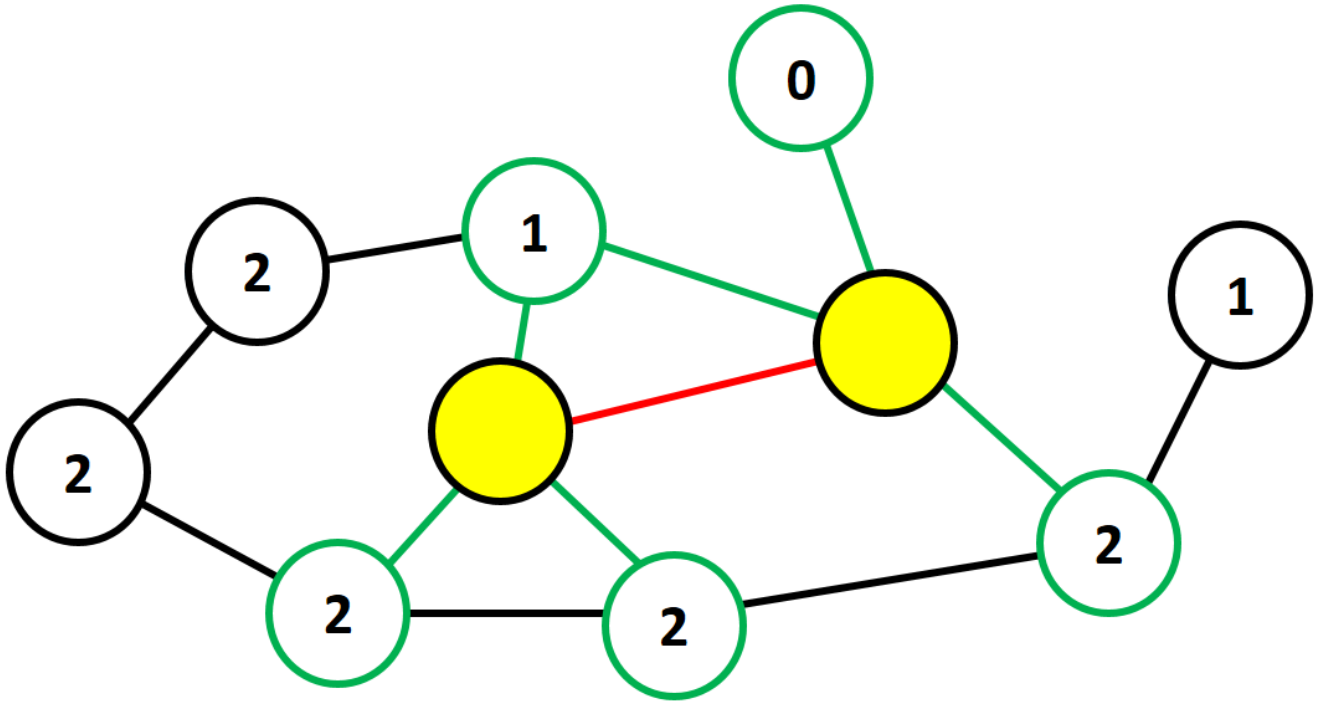
**Esempio di GreedyConstruction**

Ne scegliamo uno a caso e aggiorniamo *score* :



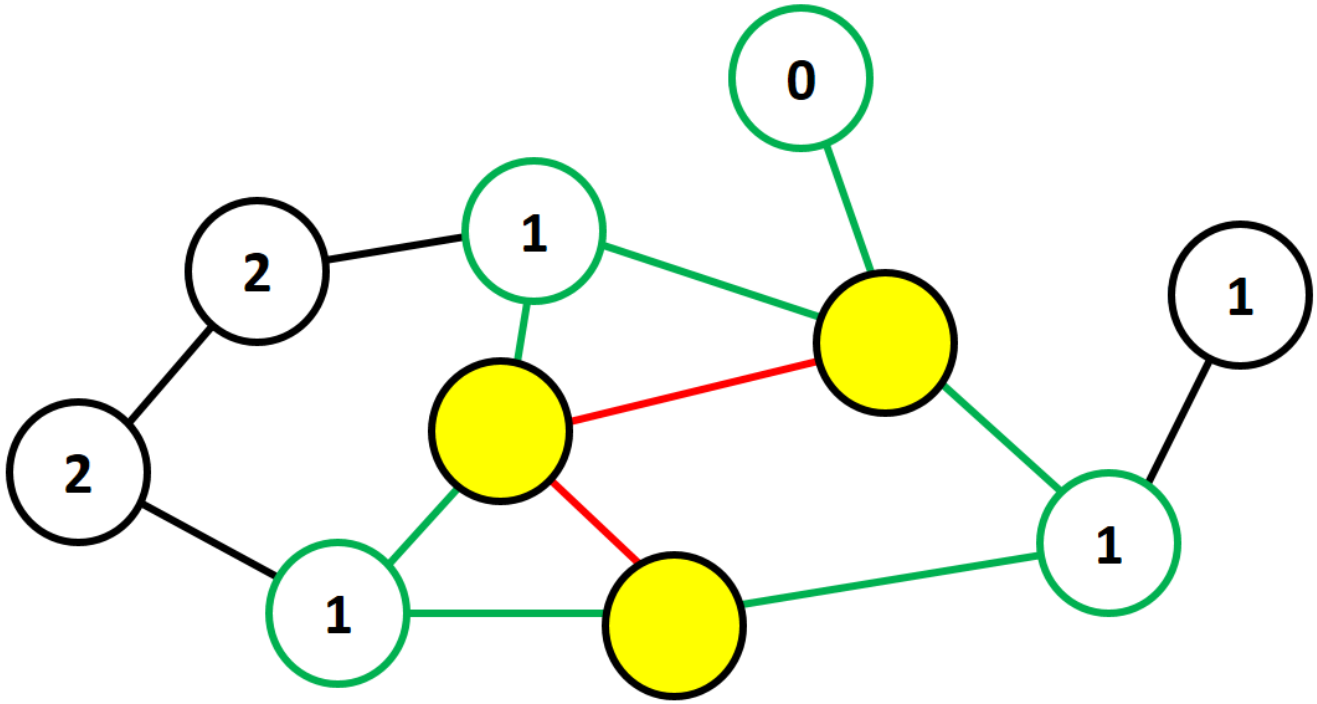
**Esempio di GreedyConstruction**

Abbiamo un vertice di *score* massimo, lo scegliamo e aggiorniamo *score* :



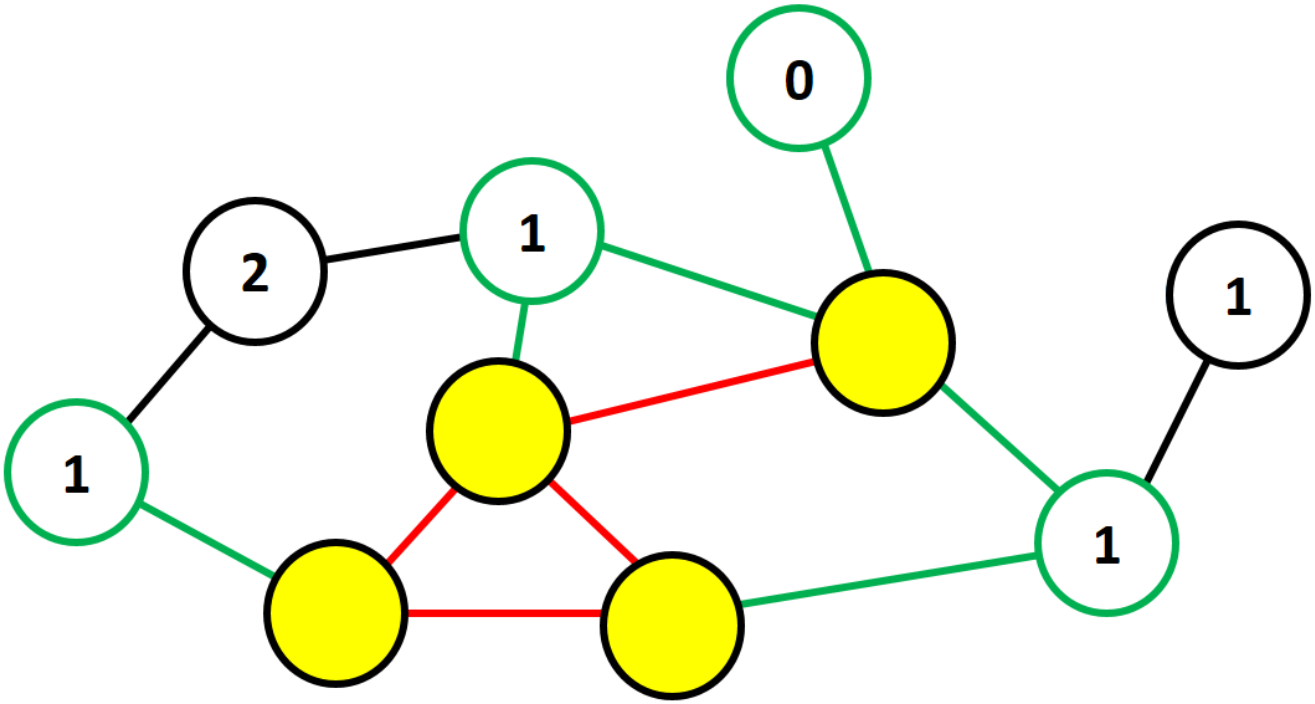
**Esempio di GreedyConstruction**

Abbiamo tre vertici di *score* massimo, ne scegliamo uno a caso e aggiorniamo *score* :



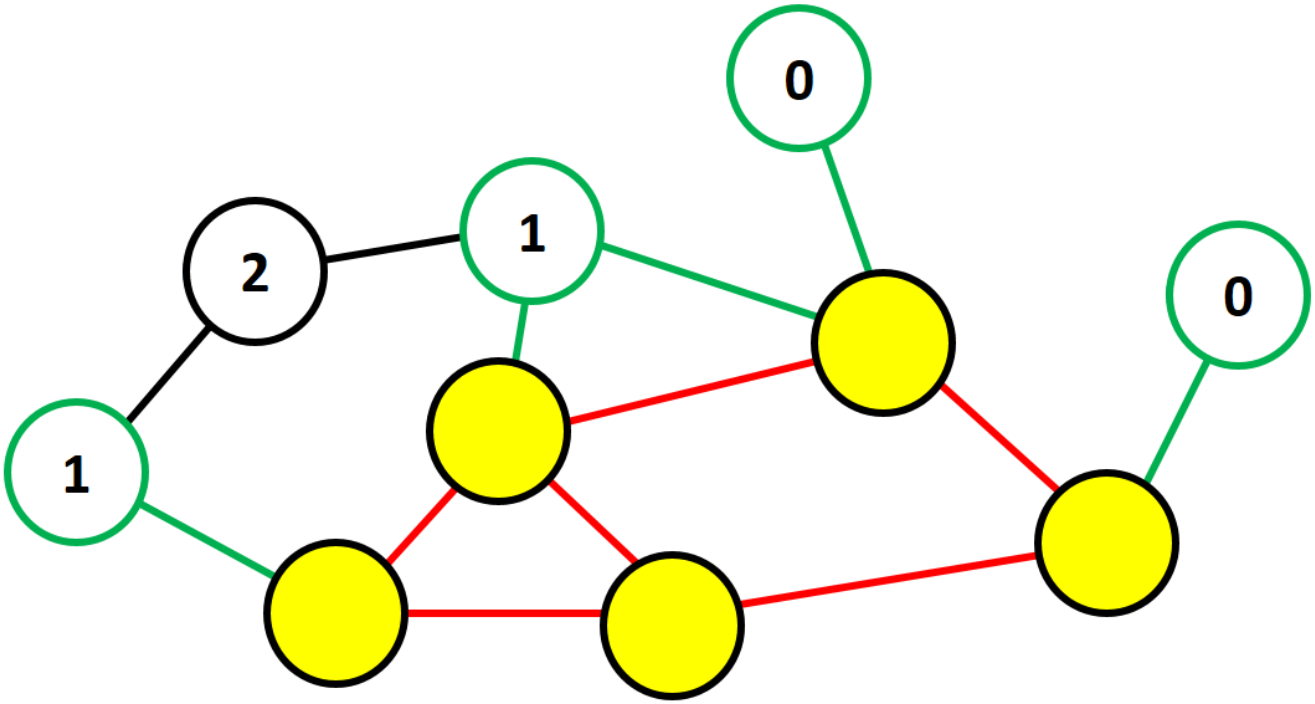
**Esempio di GreedyConstruction**

Abbiamo ancora tre vertici di *score* massimo, ne scegliamo uno a caso e aggiorniamo *score* :



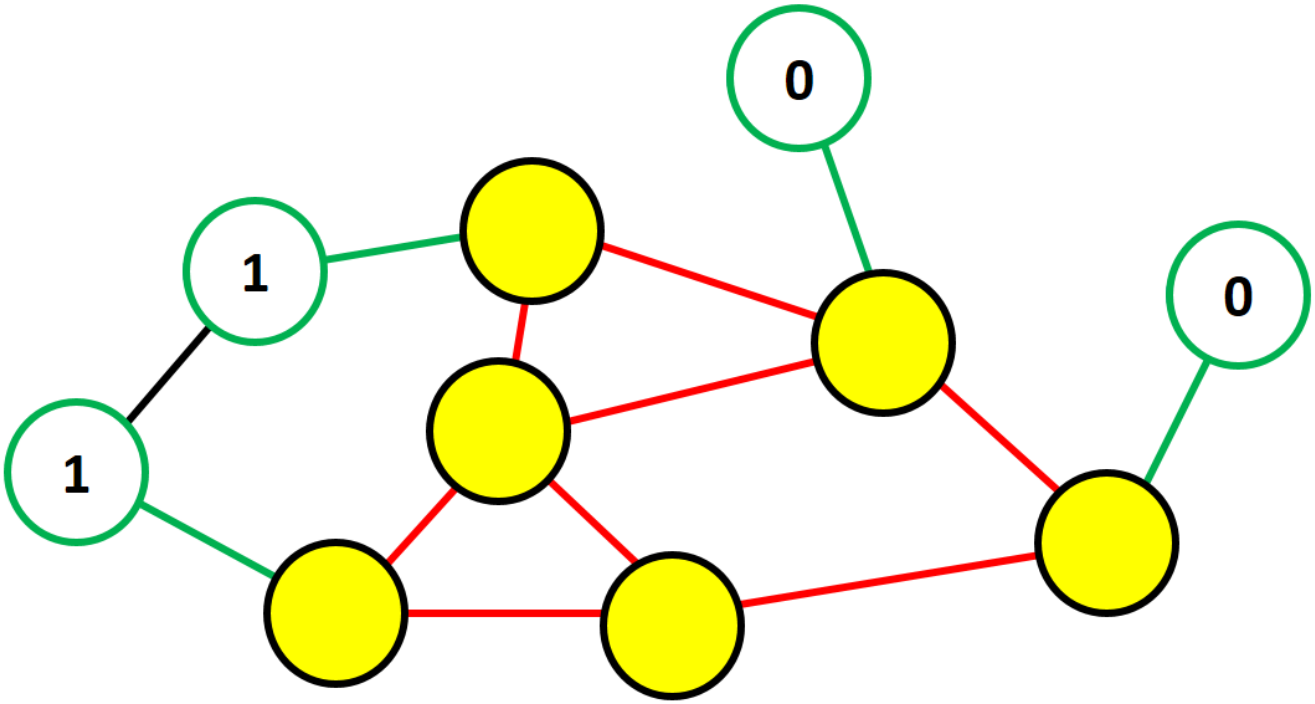
**Esempio di GreedyConstruction**

Eccetera... :



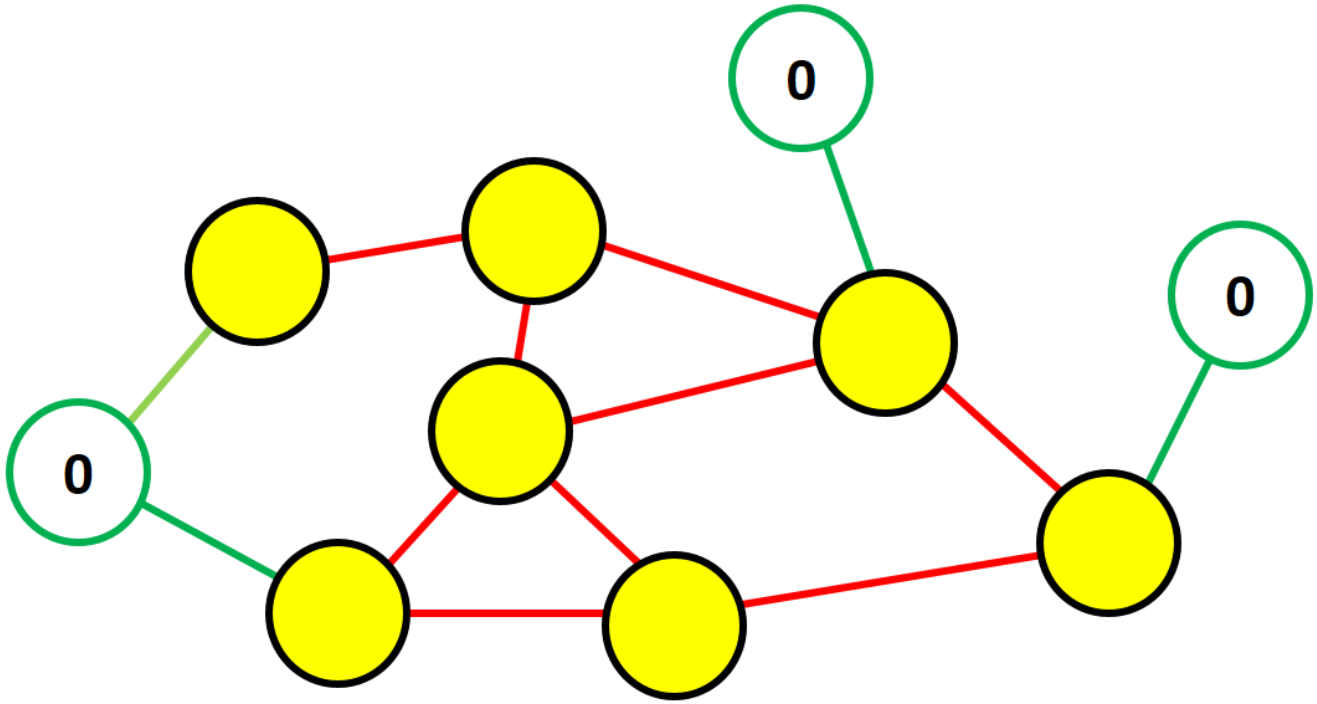
**Esempio di GreedyConstruction**

Eccetera... :



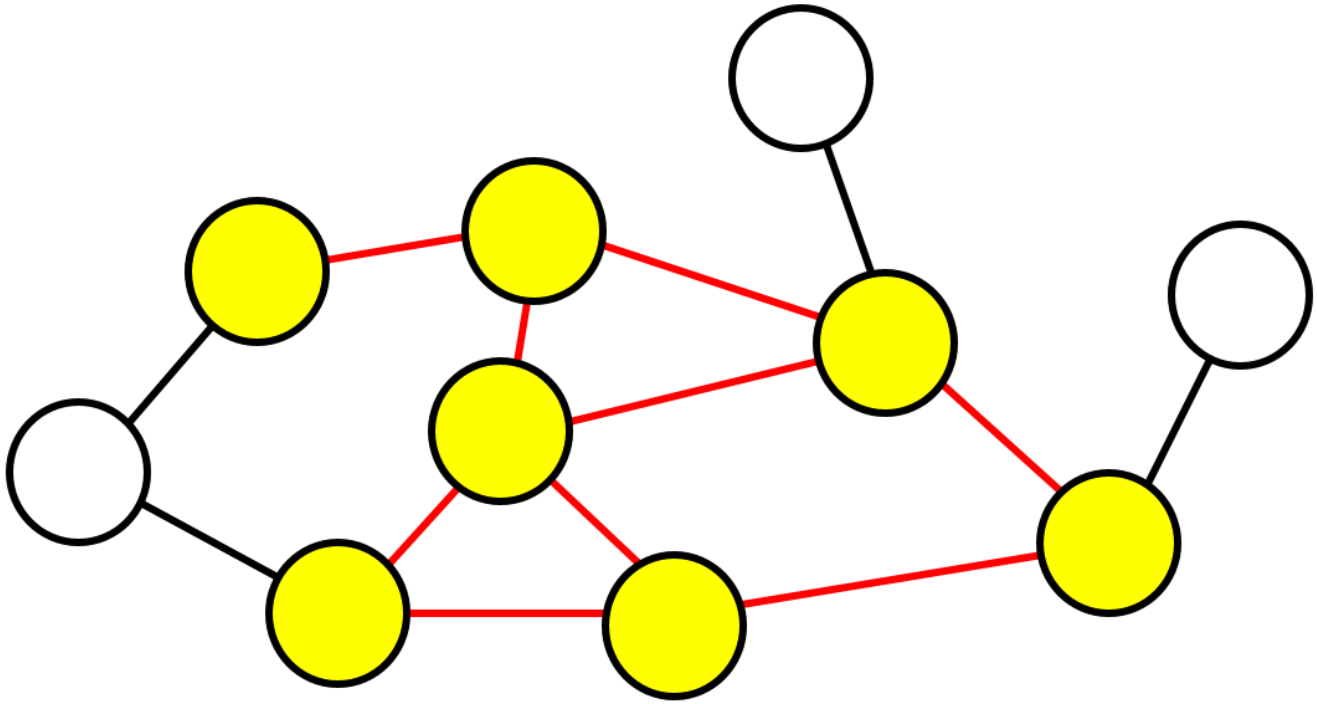
## Esempio di GreedyConstruction

...



**Esempio di GreedyConstruction**

Abbiamo una CVC !

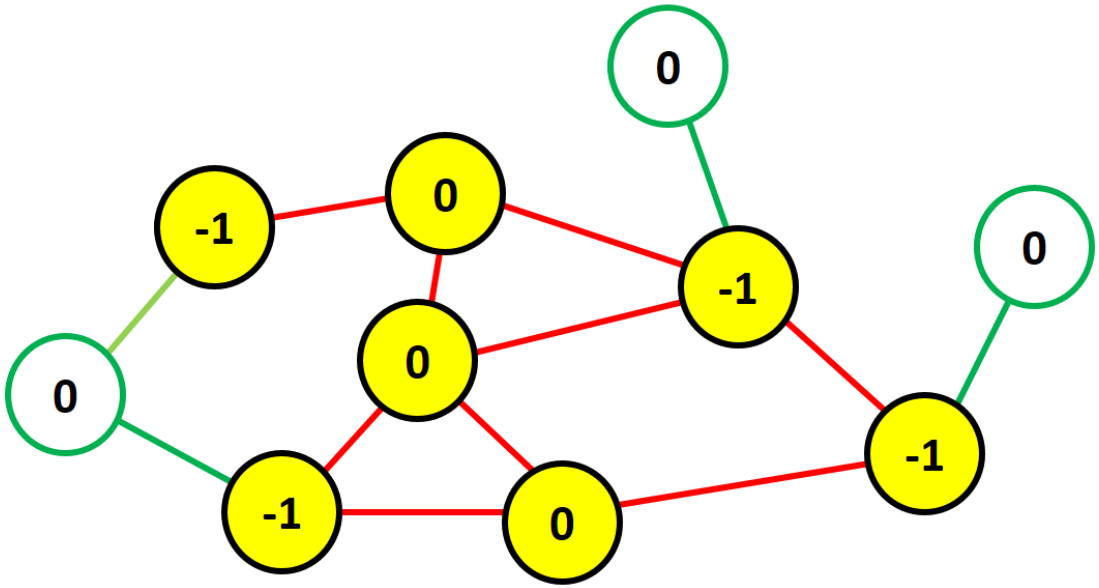


# GRASP-CVC, *LocalSearch*



La *LocalSearch* ha lo scopo di ricercare soluzioni migliori "vicine" alla soluzione trovata dalla *GreedyConstruction*. Ovvero, cerca soluzioni migliori partendo dalla CVC ottenuta e rimuovendo alcuni vertici (e aggiungendone altri se necessario).

Ad esempio, rivediamo la CVC trovata prima :




Notiamo che ci sono tre vertici in $C$ il cui *score* è 0 : questo vuol dire che se li togliamo abbiamo ancora una VC (e forse una CVC !)

# GRASP-CVC vs DFS-CVC



| Graph | $\|V\|$ | $\rho$ | MVC | GRASP-CVC | time | DFS-CVC | time |
|---|---|---|---|---|---|---|---|
| brock200-2 | 200 | 0.5 | 188 | 190 | 60.0 | 198 | 1.67 |
| brock200-4 | 200 | 0.34 | 183 | 184 | 36.63 | 196 | 1.17 |
| brock400-2 | 400 | 0.25 | 371 | 376 | 132.52 | 395 | 4.01 |
| brock400-4 | 400 | 0.25 | 367 | 376 | 132.05 | 394 | 3.98 |
| brock800-2 | 800 | 0.34 | 776 | 780 | 620.04 | 795 | 19.14 |
| brock800-4 | 800 | 0.35 | 774 | 780 | 614.08 | 795 | 19.42 |
| C125-9 | 125 | 0.1 | 91 | 91 | 10.69 | 111 | 0.21 |
| C250-9 | 250 | 0.1 | 206 | 207 | 30.41 | 236 | 0.73 |
| C500-9 | 500 | 0.09 | 443 | 448 | 97.22 | 484 | 2.89 |
| DSJC500-5 | 500 | 0.49 | 487 | 487 | 322.5 | 497 | 10.25 |
| gen200-p0-9-44 | 200 | 0.1 | 156 | 164 | 22.15 | 187 | 0.55 |
| gen200-p0-9-55 | 200 | 0.1 | 145 | 156 | 21.18 | 187 | 0.5 |
| gen400-p0-9-55 | 400 | 0.1 | 345 | 358 | 68.87 | 388 | 1.99 |
| gen400-p0-9-65 | 400 | 0.1 | 335 | 354 | 69.47 | 386 | 1.67 |
| gen400-p0-9-75 | 400 | 0.1 | 325 | 357 | 68.89 | 382 | 1.98 |
| hamming8-4 | 256 | 0.36 | 240 | 240 | 58.97 | 255 | 1.98 |
| keller4 | 171 | 0.35 | 160 | 160 | 25.81 | 169 | 0.78 |

# GRASP-CVC*



Il GRASP-CVC* è la mia versione del GRASP-CVC. L'algoritmo è uguale al GRASP-CVC, eccetto per due cambiamenti :

1. si fa una leggera modifica alla funzione *score*
2. si aggiunge un elemento di randomizzazione nella *LocalSearch*
   Si verifica empiricamente che il GRASP-CVC* dà risultati migliori del GRASP-CVC.

# GRASP-CVC vs GRASP-CVC*



| Graph | $\|V\|$ | $\rho$ | MVC | GRASP-CVC | GRASP-CVC* |
|---|---|---|---|---|---|
| brock200-2 | 200 | 0.5 | 188 | 190 | 189 |
| brock200-4 | 200 | 0.34 | 183 | 184 | 184 |
| brock400-2 | 400 | 0.25 | 371 | 376 | 375 |
| brock400-4 | 400 | 0.25 | 367 | 376 | 375 |
| brock800-2 | 800 | 0.34 | 776 | 780 | 780 |
| brock800-4 | 800 | 0.35 | 774 | 780 | 779 |
| C125-9 | 125 | 0.1 | 91 | 91 | 91 |
| C250-9 | 250 | 0.1 | 206 | 207 | 206 |
| C500-9 | 500 | 0.09 | 443 | 448 | 445 |
| DSJC500-5 | 500 | 0.49 | 487 | 487 | 487 |
| gen200-p0-9-44 | 200 | 0.1 | 156 | 164 | 160 |
| gen200-p0-9-55 | 200 | 0.1 | 145 | 156 | 145 |
| gen400-p0-9-55 | 400 | 0.1 | 345 | 358 | 351 |
| gen400-p0-9-65 | 400 | 0.1 | 335 | 354 | 353 |
| gen400-p0-9-75 | 400 | 0.1 | 325 | 357 | 348 |
| hamming8-4 | 256 | 0.36 | 240 | 240 | 240 |
| keller4 | 171 | 0.35 | 160 | 160 | 160 |