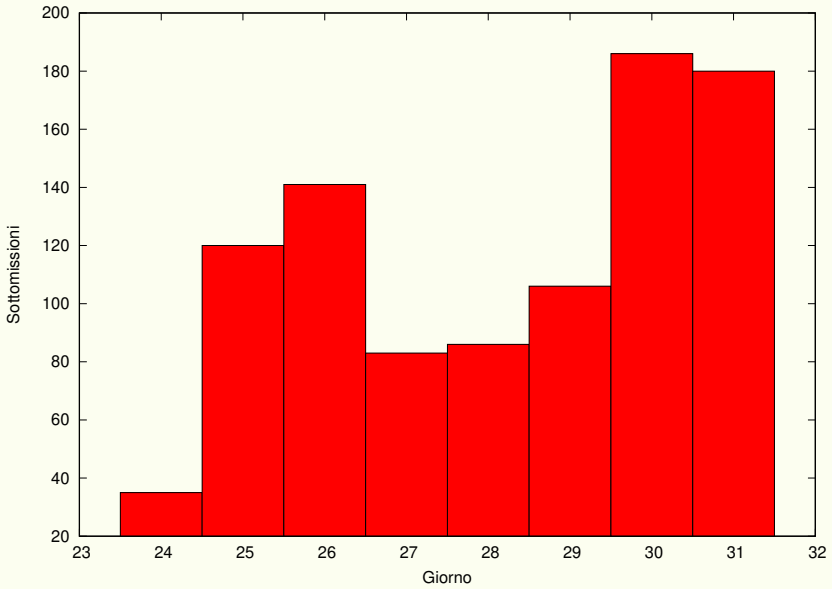
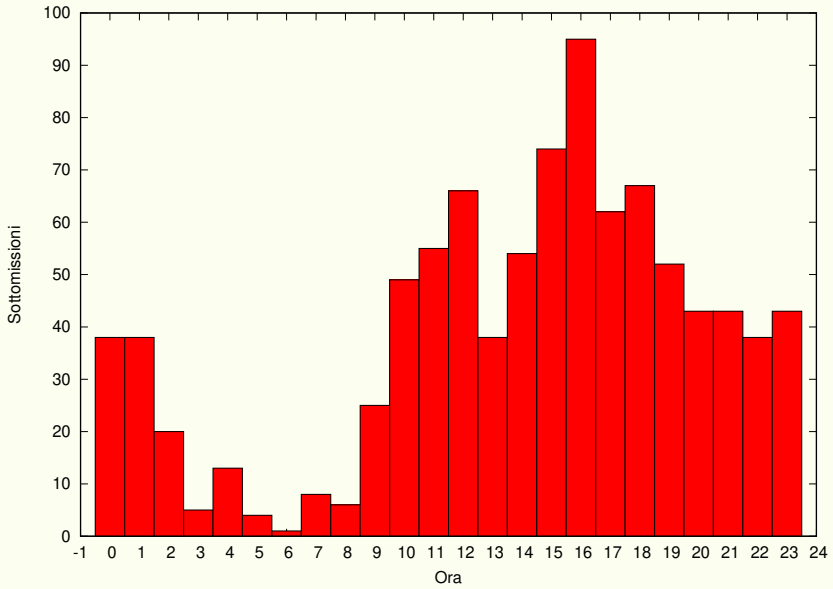
Soluzione Secondo Progetto ASD 2022/2023

Statistiche

Numero sottoposizioni: 937





- 90 gruppi iscritti, di cui 56 hanno raggiunto la sufficienza (57 gruppi che hanno fatto almeno una sottoposizione);
- 203 studenti iscritti, di cui 147 appartenenti a gruppi che hanno fatto almeno una sottoposizione;

Risultati

Punteggi

- $P < 15$ $\longrightarrow$ progetto non passato
- $15 \leq P < 40$ $\longrightarrow$ 1 punto bonus (15 gruppi)
- $40 \leq P < 65$ $\longrightarrow$ 2 punti bonus (22 gruppi)
- $P \geq 65$ $\longrightarrow$ 3 punti bonus (19 gruppi)

Classifiche e sorgenti sul sito (controllate i numeri di matricola):
`https://judge.science.unitn.it/slides/asd22/classifica_prog2.pdf`

Il problema I

Riassumiamo brevemente il problema proposto in questo progetto.

- Abbiamo un grafo indiretto non pesato, che rappresenta la struttura delle alleanze tra i nobili.
- Ogni nobile corrisponde a un nodo e la presenza di un arco tra due nodi indica l'esistenza di una alleanza.
- Ogni tavolo rappresenta una componente connessa e il vincolo secondo cui i nobili seduti ad un tavolo devono essere **tutti e solamente** alleati tra loro significa che la componente connessa deve essere una cricca.
- Dobbiamo aggiungere e/o eliminare archi in modo da rendere ogni componente connessa una cricca.

Obiettivo

L'obiettivo è **minimizzare** il numero modifiche (ovvero di archi aggiunti/tolti).

Il problema II

Cluster Editing

Il progetto è ispirato ad un problema conosciuto come Cluster Editing. È un problema NP-HARD, quindi trovare la soluzione ottima per questo problema diventa rapidamente intrattabile.

- Formalmente il problema consiste in transformare un grafo in un unione disgiunta di cricche usando il minimo numero di modifiche sugli archi[1].
- Il problema è stato di ispirazione per numerosi algoritmi nel campo della bio-informatica per il clustering di entità quali geni, proteine, fenotipi o pazienti.

[1] Böcker, S. and Baumbach, J., 2013. Cluster editing. In Conference on Computability in Europe, pp. 33-44.

I CASI SEMPLICI

La soluzione minima richiedeva di ottenere 15 punti, ovvero risolvere in modo ottimo i tre input per cui **è sufficiente solamente creare alleanze**.

Se è sufficiente creare allenze significa che il grafo è già "diviso" correttamente. In altre parole la divisione in componenti connesse è già ottima ma in questo stato non rispetta i requisiti imposti dai tavoli, ovvero le componenti non sono cricche.

OSSERVAZIONE

- Non conviene mai unire due componenti connesse.

Dobbiamo quindi trasformare in cricche le componenti connesse, aggiungendo gli archi mancanti.

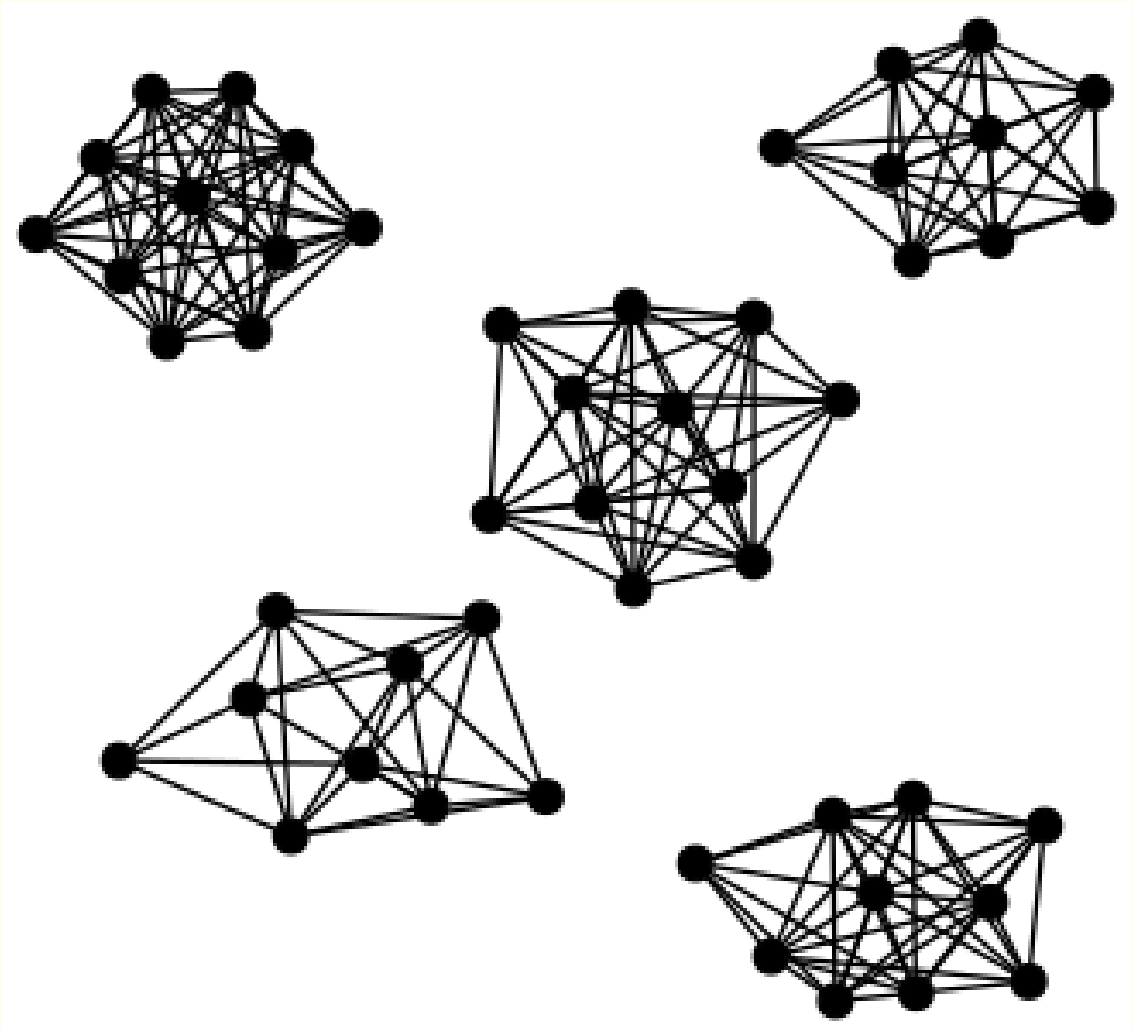
Quasi-cricche

L'algoritmo è molto semplice:

- Calcoliamo le componenti connesse con una semplice visita DFS o BFS.
- Per ogni coppia di nodi di una componente, controllo se esiste un arco nel grafo. Se non esiste lo aggiungo alla soluzione.
- ⚠ Bisogna fare attenzione a non aggiungere più volte lo stesso arco (il grafo è indiretto) e non aggiungere un arco da un nodo a sé stesso.

Punteggio: **15.00**

Testcase - Quasi cricche



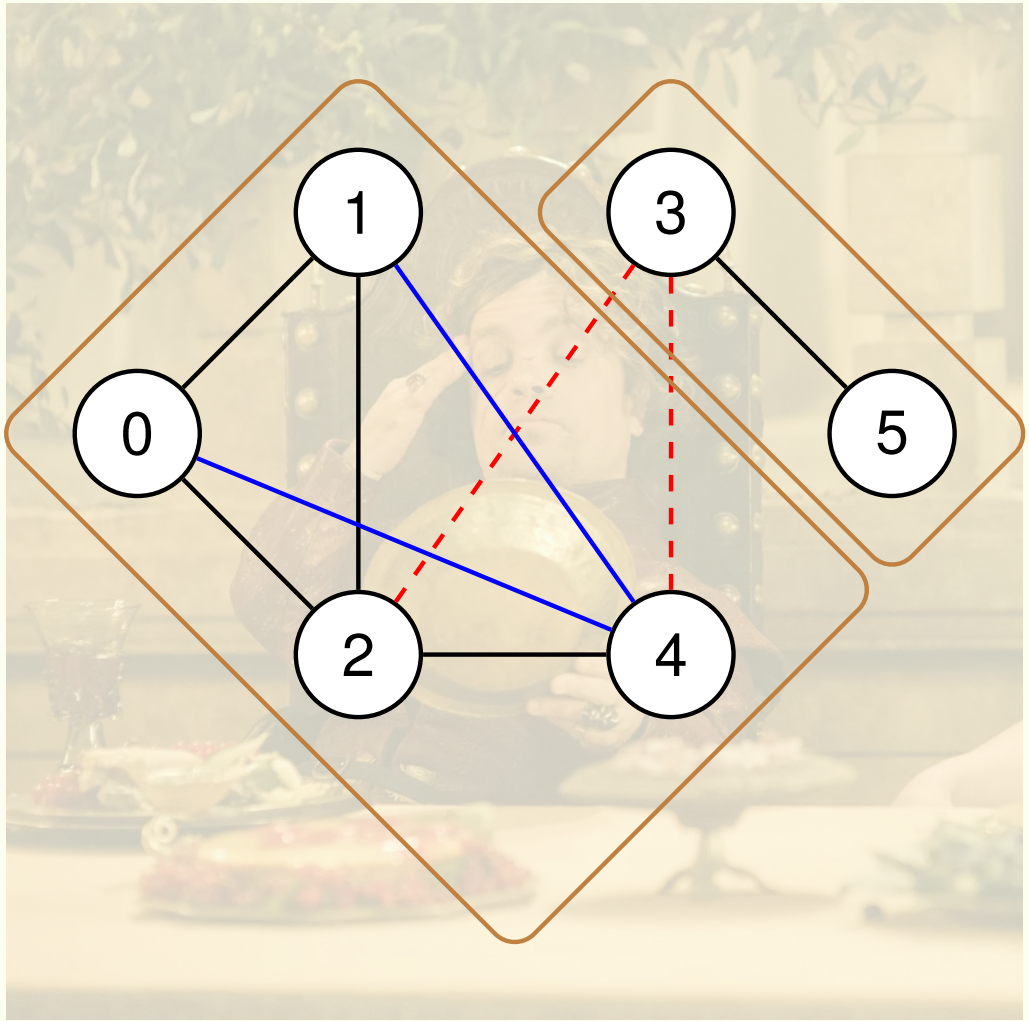
Verso la soluzione generale

Vediamo il problema da un'altra prospettiva:

- non simuliamo le aggiunte e rimozioni modificando il grafo originale, ma...
- raggruppiamo i nodi in "cluster", assegnando a ciascun nodo un *cluster_id*
- teniamo conto del numero di modifiche (costo) richiesto dall'assegnamento corrente
- in questo modo, è facile fare modifiche alla soluzione attuale per vedere se esistono soluzioni "vicine" migliori:
  - una soluzione vicina si può ottenere ad esempio spostando un nodo in un cluster adiacente
  - possiamo facilmente calcolare il costo della nuova soluzione a partire dal costo della soluzione corrente
- alla fine calcoliamo gli archi da aggiungere e rimuovere per far diventare i cluster cricche disgiunte

Come aggiornare il costo attuale (I)

- Supponiamo di essere arrivati a questo clustering
- Vogliamo spostare il nodo 4 dal cluster in cui si trova all'altro cluster
- Per farlo dobbiamo:
  1. isolarlo dal suo cluster attuale
  2. inserirlo nell'altro cluster



$costo = 4$

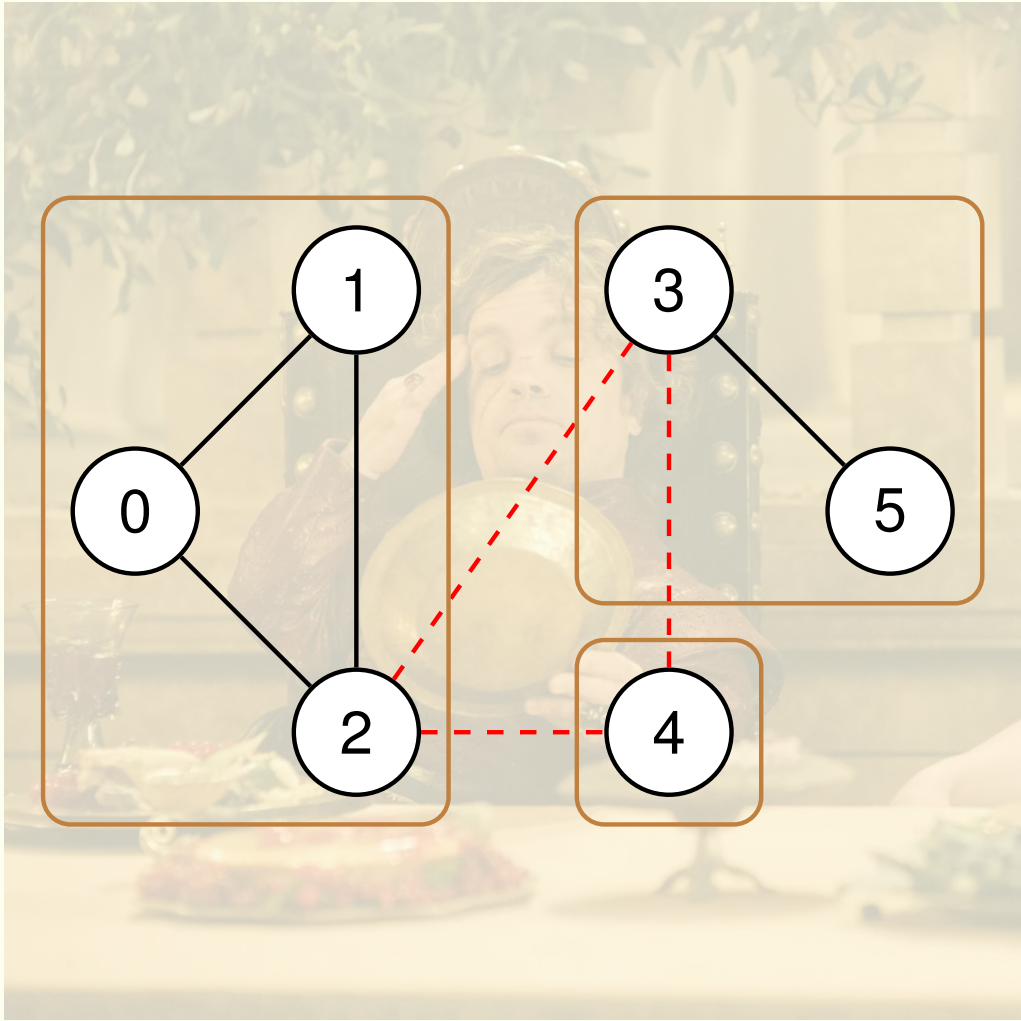
Come aggiornare il costo attuale (II)

Isolare un nodo dal suo cluster

$$costo_{new} = costo_{curr} + x - y$$

dove

- x è il numero di vicini nel cluster "vecchio" → dovrò eliminare gli archi corrispondenti
- y è il numero di non-vicini nel cluster "vecchio" → annullo l'aggiunta degli archi corrispondenti



$$costo_{new} = 4 + 1 - 2 = 3$$

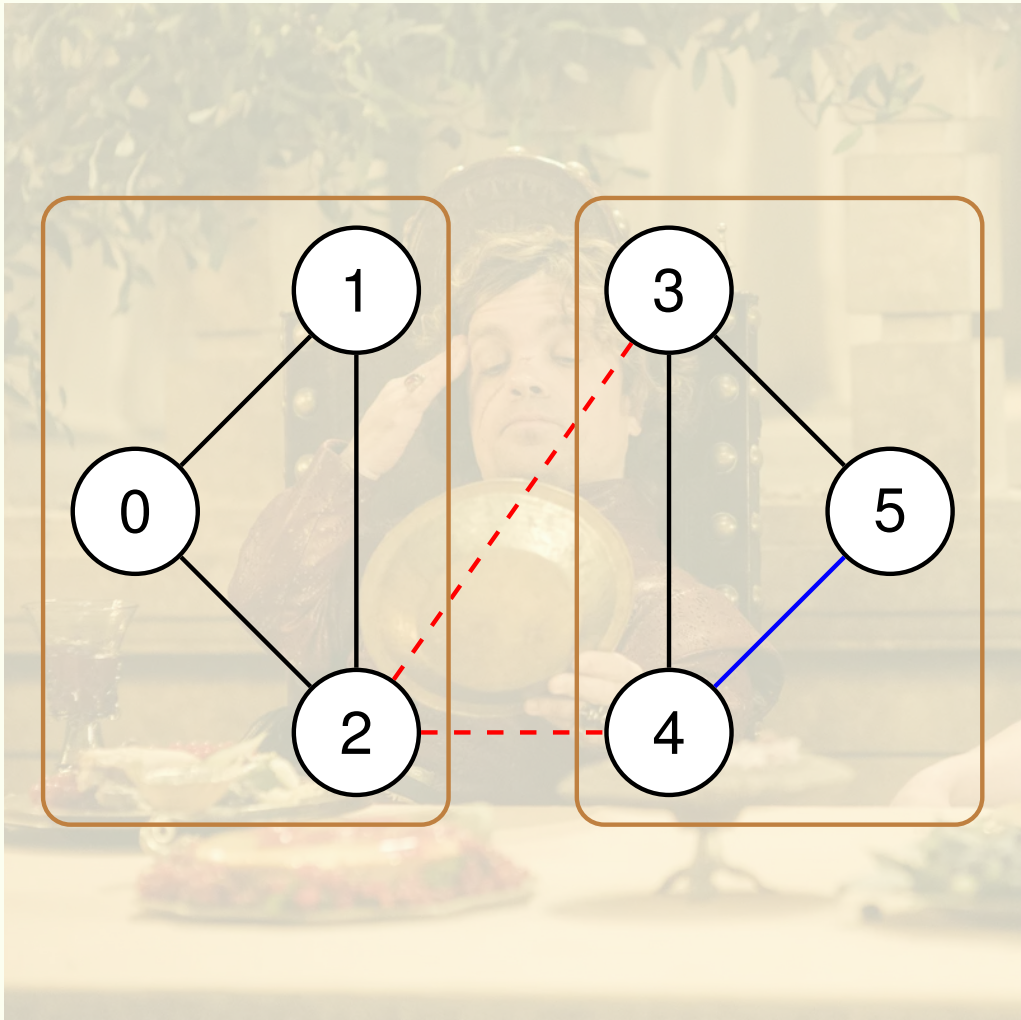
Come aggiornare il costo attuale (III)

Inserire un nodo isolato in un cluster

$$costo_{new} = costo_{curr} + x - y$$

dove

- x è il numero di non-vicini nel cluster "nuovo" → dovrò aggiungere gli archi corrispondenti
- y è il numero di vicini nel cluster "nuovo" → annullo la rimozione degli archi corrispondenti



$$costo_{new} = 3 + 1 - 1 = 3$$

Come ottenere le modifiche da fare

Scorriamo tutte le coppie di nodi:

- se i due nodi sono stati assegnati allo stesso cluster ma non esiste un arco tra di loro nel grafo originale, allora bisogna creare quell'arco
- se i due nodi sono stati assegnati a cluster diversi ma esiste un arco tra di loro nel grafo originale, allora bisogna eliminare quell'arco

L'algoritmo

- Partiamo da una soluzione valida
  - ogni nodo è in un cluster a sé stante
  - questa soluzione ha costo M: dovrò rimuovere tutti gli archi
- Ripetiamo finché c'è tempo:
  1. Perturbiamo la soluzione corrente
  2. Miglioriamo la soluzione corrente con ricerca locale

Punteggio: **77.7979**

Perturbazione

Consideriamo un sottoinsieme casuale di nodi (max 25%), e per ognuno applichiamo una tra queste due modifiche:

- **Node Isolation**[1]: ogni nodo diventa un cluster a sé stante
- **Node Mover**[1]: ogni nodo viene spostato in un cluster adiacente 'random'

[1] KaPoCE: A Heuristic Cluster Editing Algorithm

Ricerca locale (I): Label Propagation[1]

- Iteriamo i nodi in ordine casuale:
  - ogni nodo viene assegnato al cluster più “conveniente”, ossia quello che minimizza il numero di modifiche al grafo iniziale considerando il clustering attuale
  - consideriamo anche l’opzione di isolare il nodo
- Ripetiamo fino alla convergenza (o raggiunta del numero massimo di iterazioni scelto)

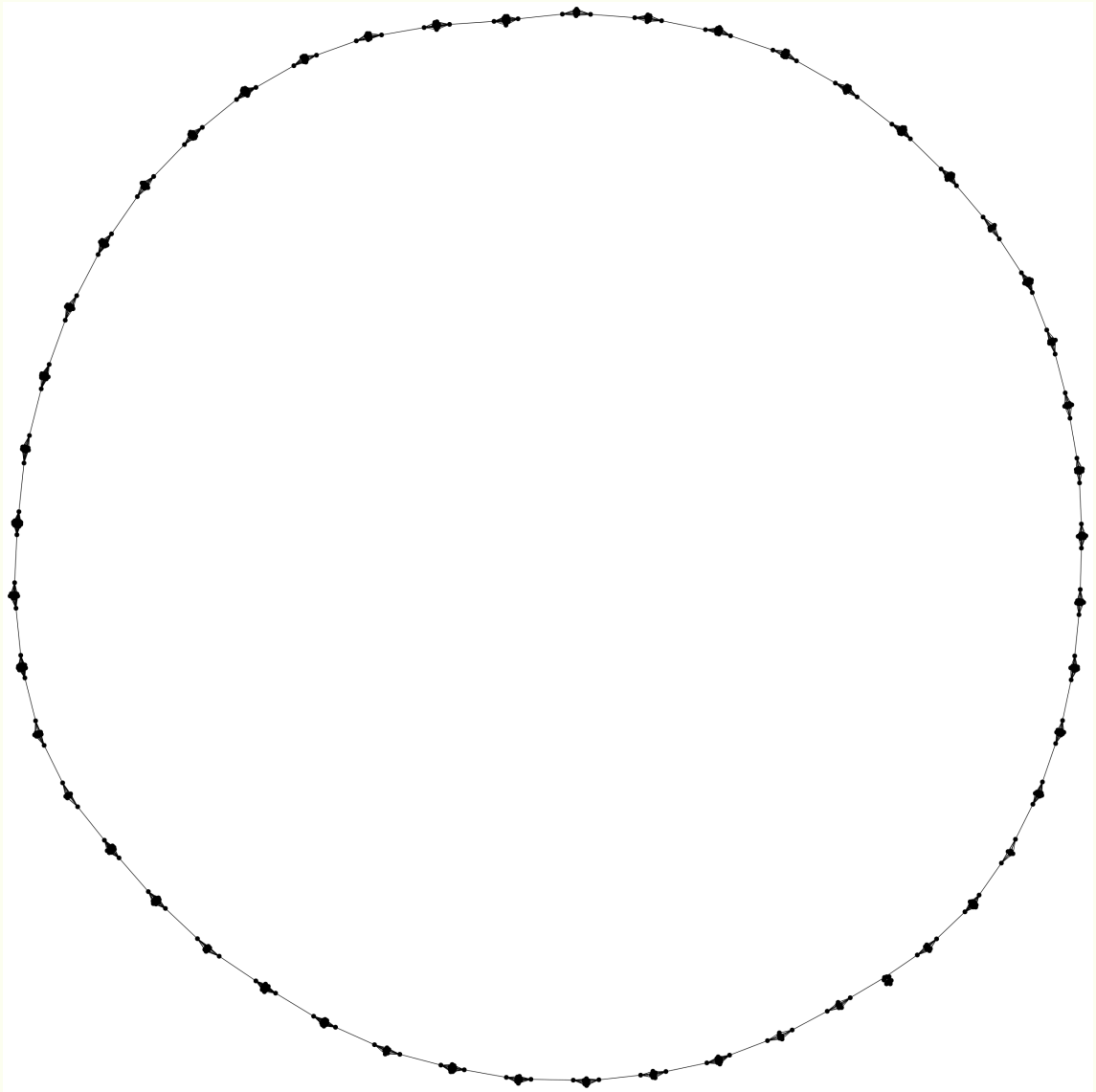
[1] KaPoCE: A Heuristic Cluster Editing Algorithm

Ricerca locale (II): Clique Removal[1]

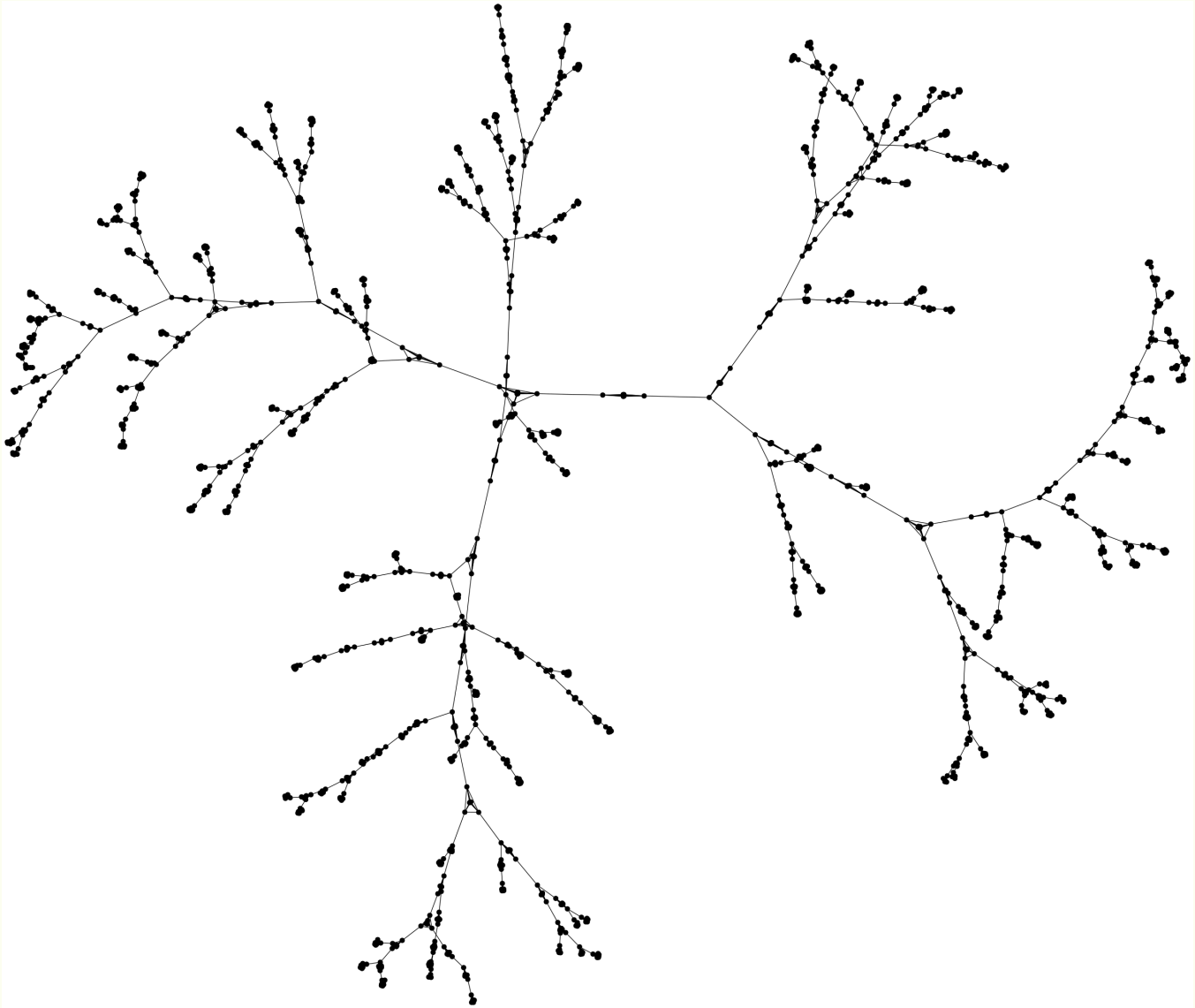
- Iteriamo i cluster in ordine casuale:
  - proviamo a rimuovere il cluster assegnando i nodi al suo interno al rispettivo cluster più conveniente
  - teniamo la modifica solo se migliora la soluzione precedente
- Ripetiamo fino alla convergenza (o raggiunta del numero massimo di iterazioni scelto)

[1] KaPoCE: A Heuristic Cluster Editing Algorithm

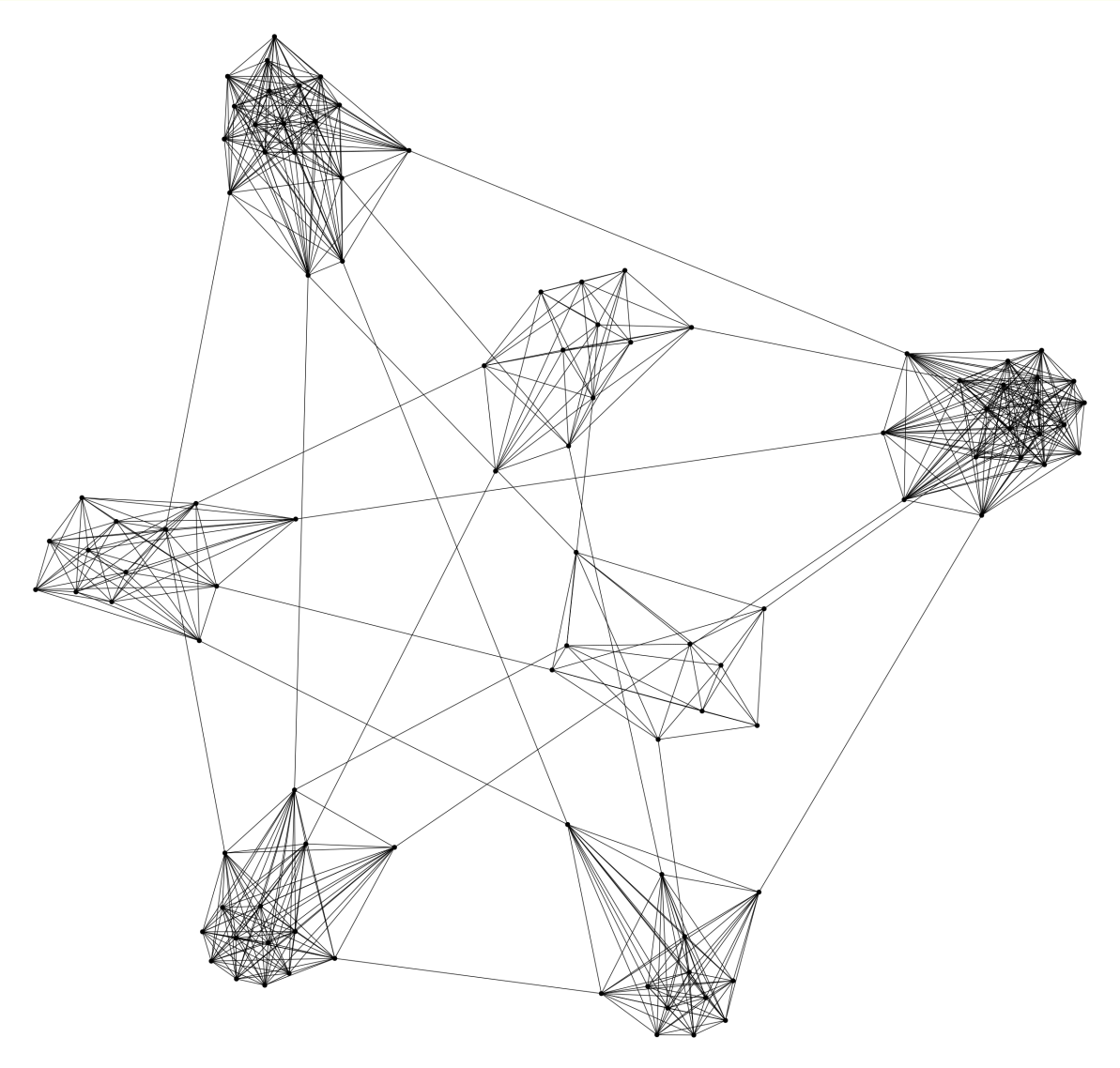
Testcase - Anello di cricche



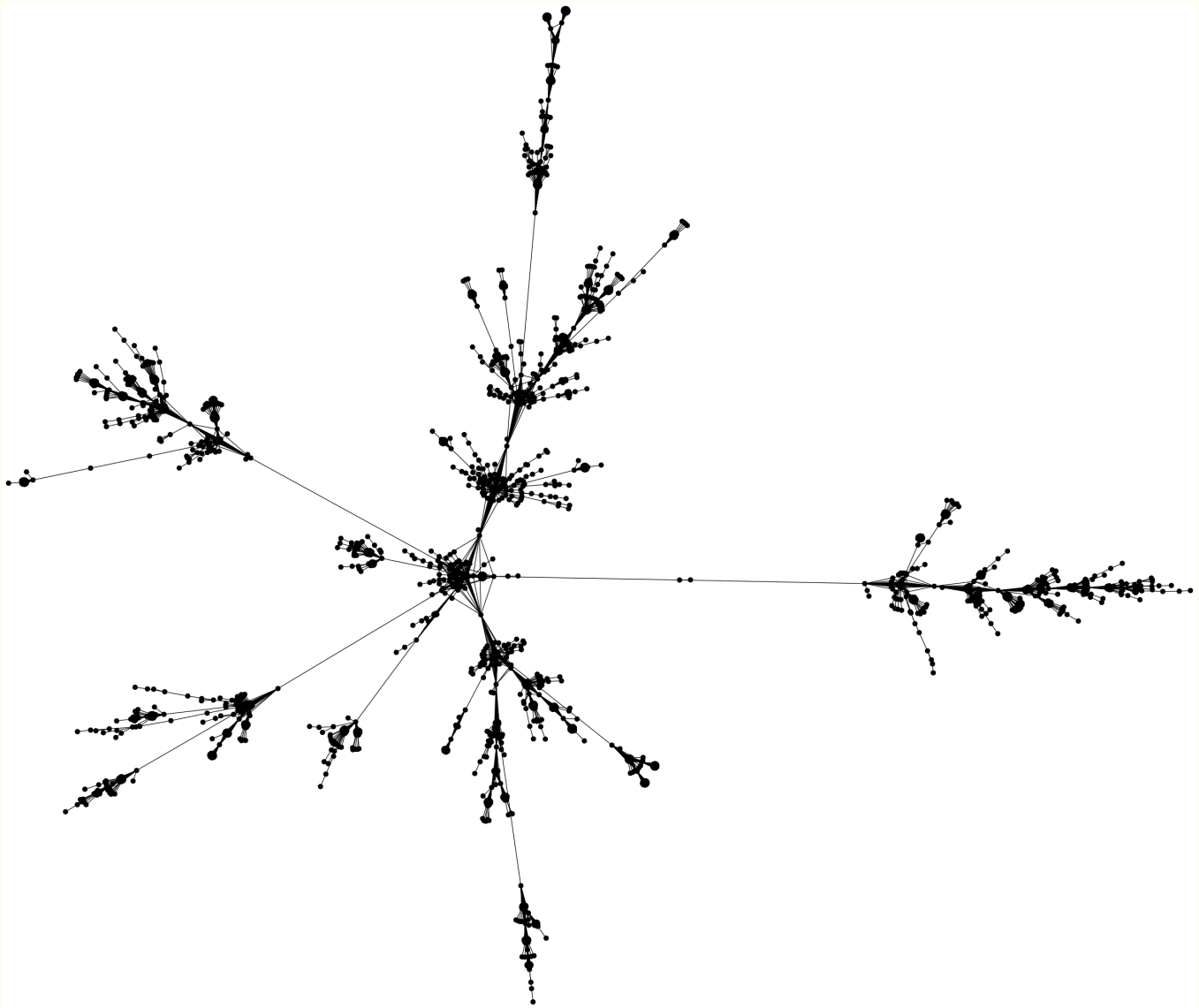
Testcase - Albero di cricche



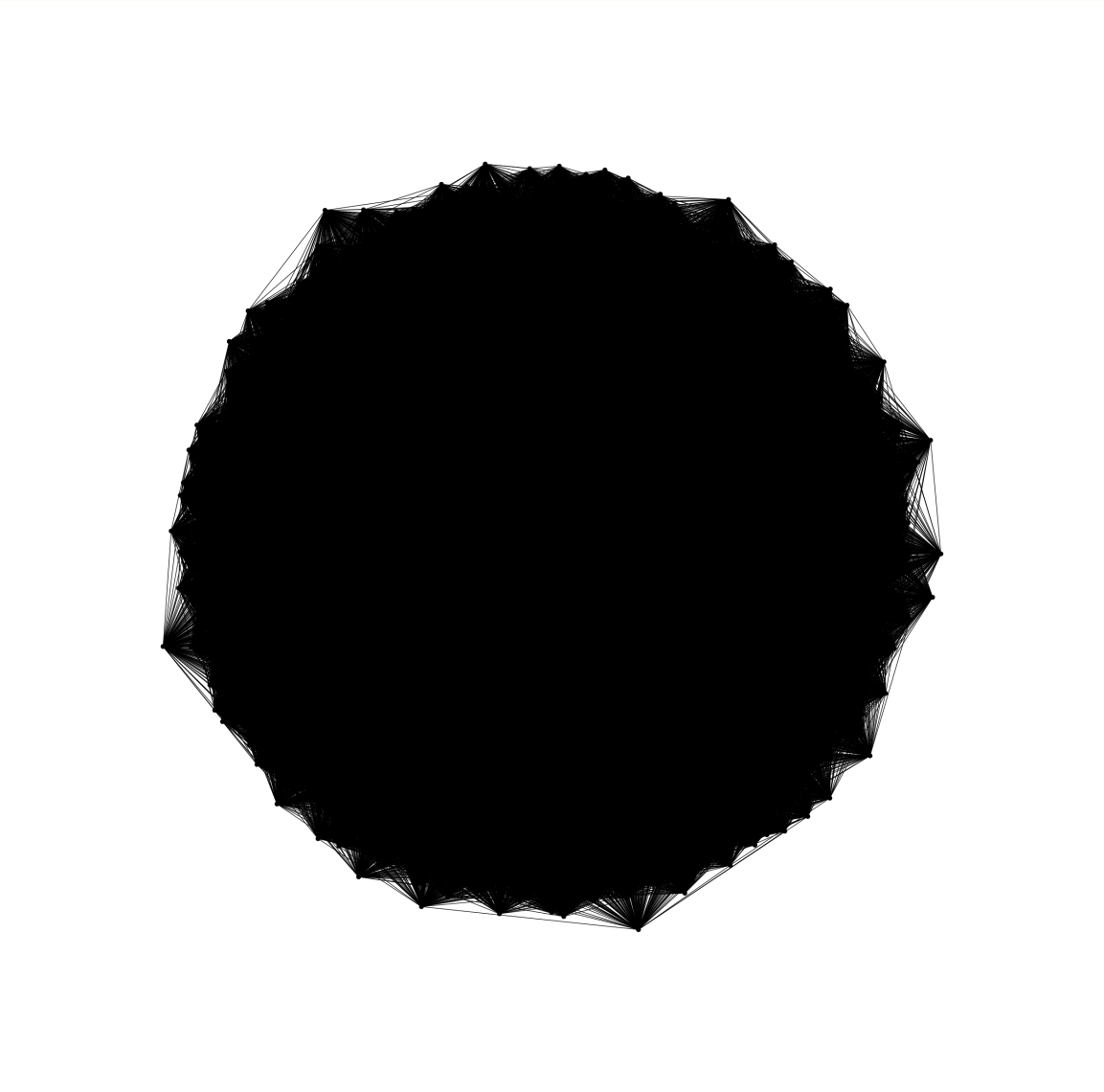
Testcase - Quasi cricche connesse



Testcase - Cactus



Testcase - Blackhole



And the winner is...