Relazione di Algoritmi Avanzati

Laboratorio 3 - Minimum Cut

Nicola Carlesso - matricola 1237782
Federico Brian - matricola 1243422

A.A. 2019/2020

Indice

Elenco delle figure

Elenco delle tabelle

1 Introduzione

Il presente documento descrive le scelte architetturali ed implementative del terzo elaborato di laboratorio del corso di Algoritmi Avanzati. Di seguito, verrà offerta una panoramica sul lavoro svolto dagli studenti Nicola Carlesso e Federico Brian, riguardante le prestazioni dell'algoritmo di Karger per il problema del *Minimum Cut*[1], rispetto a quattro parametri:

* Il tempo impiegato dalla procedura di **Full Contraction**;
* Il tempo impiegato dall'algoritmo completo per ripetere la contrazione un numero sufficientemente alto di volte;
* Il *discovery time*, ossia il momento in cui l'algoritmo trova per la prima volta il taglio di costo minimo;
* L'errore nella soluzione trovata rispetto al risultato ottimo.

Il problema del minimum cut è definito come segue.

1.1 Definizioni preliminari

Per definire il problema del mincut, abbiamo bisogno di dare prima delle definizioni topiche:

Def. 1.1. ***(Multiinsieme).*** *Collezione di oggetti con ripetizioni.*

* $S = \{\{v : v \in S\}\}$
* $\forall\ v \in S\ m(v) \in \mathbb{N} \setminus \{0\}$ *con* $m(v)$ = *molteplicità di* v*, cioè quante copie di* v *ci sono in* S*.*

Def. 1.2. ***(Multigrafo non orientato).*** *multigrafo* $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ *tale che:*

* $\mathcal{V} \subseteq \mathbb{N}, \mathcal{V}$ *finito;*
* $\mathcal{E}$ *è un multiinsieme a elementi del tipo* $\{u, v\} : u \neq v$ *(no self loops).*

Remark: un grafo semplice grafo $G = (V, E)$ è anche un multigrafo. Il viceversa non è vero.

Def. 1.3. ***(Cammino).*** *In un multigrafo, un cammino è una sequenza di nodi dove* $\forall\ u, v \in \mathcal{V}$ $\Rightarrow \exists\ (u, v) \in \mathcal{E}$.

Cioè esiste almeno un lato che connette ogni coppia di nodi in $\mathcal{V}$.

Def. 1.4. ***(Connettività).*** *Un multigrafo è connesso se* $\forall\ u, v \in \mathcal{V} \Rightarrow \exists$ *un cammino che li connette.*

Def. 1.5. ***(Taglio).*** *Dato un multigrafo* $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ *connesso, un taglio* $\mathcal{C} \subseteq \mathcal{E}$ *è un multiinsieme di lati tale che* $\mathcal{G}' = (\mathcal{V}, \mathcal{E} - \mathcal{C})$ *non è connesso.*
Equivalentemente, si può dire che $\mathcal{G}'$ *ha almeno due componenti connesse.*

[1] d'ora in poi mincut

1.2 Problema del Minimum Cut

"Dato un multigrafo $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ non orientato, un taglio $\mathcal{C} \subseteq \mathcal{E}$ è il multiinsieme di lati di cardinalità minima tale che $\mathcal{G}' = (\mathcal{V}, \mathcal{E} \setminus \mathcal{C})$ non è connesso."

2 Scelta del linguaggio di programmazione

Per lo svolgimento di questo *assignment*, come per il precedente, è stato scelto Java nella sua versione 8 come linguaggio di programmazione. La scelta è derivata, principalmente, da due fattori:

* è stato sia studiato durante il percorso di laurea triennale, sia approfondito autonomamente da entrambi;
* in Java, è possibile utilizzare riferimenti ad oggetti piuttosto che oggetti stessi. Questo ha permesso un'implementazione degli algoritmi che si potrebbe definire "accademica", perché coerente con la complessità dichiarata e semanticamente vicina allo pseudocodice visto a lezione. Sono infatti evitate complessità aggiuntive derivanti da inavvertite copie profonde.

3 Scelte implementative

Dato che l'algoritmo di Karger richiedere soprattutto di creare grafi nuovi manipolando i lati del grafo precedentemente creato, si è rivelato necessario creare un modello che potesse rimuovere ed accedere ai lati di un grafo in $\mathcal{O}(1)$. Per questo abbiamo utilizzato una matrice di adiacenza nella rappresentazione dei grafi. Siamo consapevoli che non trattandosi di grafi completi, la matrice di adiacenza non sarà quasi mai densamente popolata, però ci è sembrato un buon compromesso per mantenere una complessità costante. Abbiamo mitigato questa "problematica" adottando una matrice triangolare inferiore che però viene utilizzata come se fosse una normale matrice $n \times n$, con $n = |V|$.
Inoltre, per permettere all'algoritmo di Karger di selezionare in modo random un lato del grafo, abbiamo comunque deciso di utilizzare una lista di lati che ad ogni iterazione di `full_contraction` viene aggiornata. É stato scelto di utilizzare una lista di lati e non solo la matrice di adiacenza per la scelta del lato random perché, nel caso in cui non avessimo usato la lista di lati, pescare un lato avrebbe richiesto l'utilizzo di un ciclo `while(finché non trovi un valore non vuoto nella matrice)`, che avrebbe potuto chiedere parecchio tempo.
La lista di lati, piuttosto di utilizzare un `ArrayList`, l'abbiamo implementata usando una `LinkedList`, che permette di eliminare i gli elementi della lista in $\mathcal{O}(1)$, requisito fondamentale per mantenere la corretta complessità dell'algoritmo di `Karger`.
Come specificato nel precedente paragrafo, nell'implementazione dei tre algoritmi si è cercato di creare meno oggetti possibile usando quasi esclusivamente riferimenti.

3.1 Algoritmo di Karger

L'algoritmo di Karger si può riassumere nei tre seguenti passaggi:

* Scelgo a caso un lato;
* "contraggo" i due nodi relativi eliminando tutti i lati incidenti su entrambi;
* ripeto finché restano solo 2 nodi: restituisco i lati tra quei 2 nodi.

Ciò funziona con probabilità bassissima, che però può essere amplificato ripetendo questo processo molte volte.

```
public static int Karger(Graph G, int k){
    Integer min = Integer.MAX_VALUE;
    for(int i = 0; i < k; i++){//O(n^2 log n)
        Graph newGraph = new Graph(G);//O(m)
        int t = full_contraction(newGraph);//O(n^2)
        if(t < min)
            min = t;
    }
    return min;
}
```

Figura 1: Algoritmo di Karger.

```
private static int full_contraction(Graph G){
    while(G.getAdjacentMatrix().getEdges().size() > 1){//O(n)
        Edge randomEdge = G.getAdjacentMatrix().getRandomEdge();//O(n)
        Integer nodeA = randomEdge.getNodeA();
        Integer nodeB = randomEdge.getNodeB();
        G.contraction(nodeA, nodeB);//O(n)
    }
    return G.getNumberOfEdges();//O((n^2)/2)
}
```

Figura 2: Algoritmo di Full Contraction.

`KARGER` ripete `FULL_CONTRACTION` k volte per ridurre la probabilità di errore.

3.2 Modello

Le componenti del modello, vale a dire le classi presenti all'interno del `package` chiamato `lab2.model`, comprendono tutte le strutture dati utilizzate nella risoluzione dei tre problemi assegnati.

* `AdjacentMatrix`: matrice di adiacenza usata per rappresentare i grafi, che si presenta come una normale matrice $n \times n$ simmetrica rispetto alla sua diagonale nulla[2]. L'algoritmo `Karger`, difatti, la tratta come tale. In verità è stata implementata come matrice triangolare inferiore, contenendo `n - 1` array di lunghezza crescente: questa scelta è stata fatta per risparmiare memoria, evitando di costruire una matrice quadrata. Tale classe presenta i metodi standard `get(u,v)` e `set(u,v)`, per ottenere ed impostare il peso del lato (u,v), ed un motodo `getMinAdjacentVertexWeightIndex(v)` per ottenere il lato con peso minore che ha per estremo il vertice v. La matrice di adiacenza inoltre possiede una lista di lati, per permettere alla funzione `full_contraction` di scegliere casualmente un lato in $\mathcal{O}(n)$;

* `Edge`: classe utilizzata da `AdjacentMatrix` esclusivamente per restituire un lato del grafo in modo random;

* `Graph`: classe che contiene solo un campo `Adjacentmatrix` e che si occupa di effettuare la "contrazione" di un lato ad ogni iterazione di `full_contraction`.

[2] poiché non esistono cappi

3.3 Algoritmi

Il package `lab3.algorithm` contiene un'unica classe, `MinCut`, che permette di risolvere il problema minCut utilizzando l'algoritmo `Karger`.

* `Karger`: l'algoritmo ha una struttura identica a quella dello pseudocodice visto a lezione per l'algoritmo di Karger. Questo algoritmo chiama la funzione `full_contraction` e possiede una complessità di $\mathcal{O}(n^4 \log n)$. Molto simile a `Karger` è `Karger_discovery_time`, utilizzata per calcolare quanto tempo ci mette l'algoritmo di Karger a trovare la soluzione esatta;
* `full_contraction`: anch'essa identica allo pseudocodice visto a lezione, esegue $n - 2$ contrazioni del grafo. Come per `Karger`, è presente la funzione `full_contraction_time` che calcola il tempo di esecuzione di `full_contraction`.

3.4 Main

Il package `lab3.main` contiene la classe `Main`, responsabile dell'esecuzione degli algoritmi.
All'interno vi sono due funzioni:

* `Main`: si occupa di eseguire il calcolo dell'algoritmo di Karger, calcolare il tempo di esecuzione del *Discovery Time* e della prima iterata di `full_contraction`. Infine chiama la funzione `write_results` per calcolare l'errore relativo tra la soluzione trovata da `karger` e la soluzione corretta ed infine scrivere tutti i risultati all'interno di `mincut.txt`;
* `write_results`: si occupa di annotare in un file `.txt` i risultati e le tempistiche ottenuti dagli algoritmi sviluppati per questo laboratorio.

4 Risultati dell'algoritmo `Karger`

Questa sezione risponderà alle varie domande poste per l'assignement.

4.1 Domanda 1

Misurate i tempi di calcolo della procedura `full_contraction` sui grafi del dataset. Mostrate i risultati con un grafico che mostri la variazione dei tempi di calcolo al variare del numero di vertici nel grafo. Confrontate i tempi misurati con la complessità asintotica di `full_contraction`.

4.1.1 Svolgimento

N.	Size	Full Contraction Time (s)
1	6	1.0E-6
2	6	1.2E-6
3	6	1.7E-6
4	6	1.5E-6
5	10	3.9E-6
6	10	3.6E-6
7	10	3.7E-6
8	10	3.9E-6
9	25	2.72E-5
10	25	5.16E-5
11	25	2.3E-5
12	25	2.3E-5
13	50	1.269E-4
14	50	1.207E-4
15	50	1.136E-4
16	50	1.178E-4
17	75	3.101E-4
18	75	3.266E-4
19	75	3.47E-4
20	75	3.348E-4
21	100	6.926E-4
22	100	7.199E-4
23	100	7.257E-4
24	100	7.237E-4
25	125	0.0014224
26	125	0.0014538

Continua nella pagina seguente

Tabella 1 – *continuazione dalla pagina precedente*

N.	Size	Full Contraction Time (s)
27	125	0.0015773
28	125	0.0014295
29	150	0.0030876
30	150	0.0028798
31	150	0.0029867
32	150	0.0030172
33	175	0.0046466
34	175	0.0044992
35	175	0.0052528
36	175	0.0046107
37	200	0.0072138
38	200	0.0066759
39	200	0.0072141
40	200	0.0075121

Tabella 1: Risultati della procedura *Full Contraction* rispetto alla domanda 1

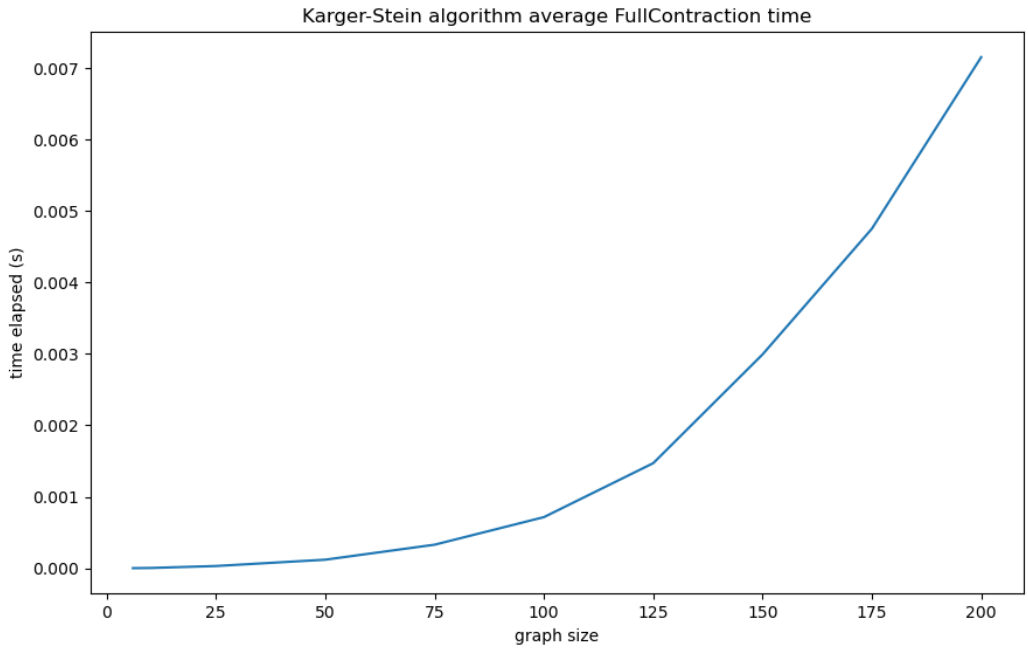

Figura 3: Analisi asintotica di `full_contraction`

`full_contraction` ha una complessità di $\mathcal{O}(n^2)$, che è ben visibile dal grafico in Figura 3.

4.2 Domanda 2

Misurate i tempi di calcolo dell'algoritmo di Karger sui grafi del dataset, usando un numero di ripetizioni che garantisca una probabilità minore o uguale a $\frac{1}{n}$ di sbagliare. Mostrate i risultati con un grafico che mostri la variazione dei tempi di calcolo al variare del numero di vertici nel grafo. Confrontate i tempi misurati con la complessità asintotica dell'algoritmo.

4.2.1 Svolgimento

Calcolo del numero di iterazioni necessarie

- Analisi di una singola iterazione di FC.
  - Sia $t = |\text{ min-cut }| = C$
  - E_i = al passo $i-$esimo di FC;
  - $Pr(\text{successo di FC}^3) \geq Pr\left(\bigcap_{i=1}^{n-2} E_1\right)$.
    - $Pr(E_1) \geq 1 - \frac{t}{t\frac{n}{2}} = 1 - \frac{2}{n}$. Quindi la probabilità di non beccare un arco del min-cut alla prima iterazione di FC è buona. Successo con alta probabilità.
    - $Pr(E_2|E_1) \geq 1 - \frac{t}{t\frac{n-1}{2}}$
    - $\vdots$
    - $Pr(E_i|E_1 \cap E_2 \cap ... \cap E_{i-1}) \geq 1 - \frac{t}{t\frac{(n-i+1)}{2}} = 1 - \frac{2}{n-i+1}$

  $\Rightarrow Pr(\text{successo di FC}) \geq Pr\left(\bigcap_{i=1}^{n-2} E_1\right) \geq \prod_{i=1}^{n-2}\left(1 - \frac{2}{n-i+1}\right) = Pr\left(\bigcap_{i=1}^{n-2} E_1\right)\frac{n-i-1}{n-i+1} =$

  $= \frac{(n-2)}{n} \cdot \frac{(n-3)}{n-1} \cdot \frac{(n-4)}{n-2} \cdot ... \cdot \frac{3}{5} \cdot \frac{2}{4} \cdot \frac{1}{3} = \frac{2}{n(n-1)} \geq \frac{2}{n^2}$

  Quindi, eseguendo FC una sola volta, la probabilità di non beccare un arco appartenente al min-cut è almeno di $\frac{2}{n^2}$.

- Dobbiamo calcolare la probabilità che le k iterazioni di FC non accumulino la taglia del min-cut.
  - $Pr(\text{le } k \text{ FC non accumulano la taglia del min-cut}) \leq \left(1 - \frac{2}{n^2}\right)^k$ (evento insuccesso)

    Vogliamo che questa probabilità sia $\leq \frac{1}{n}$. Quindi scelgo $k = \frac{n^2}{2}\log n$. Dimostriamo che è sufficiente.
    - Infatti $\left(\left(1 - \frac{2}{n^2}\right)^{\frac{n^2}{2}}\right)^{\log n} \leq e^{-\log n} = \frac{1}{n}$.

- Quindi con probabilità almeno $\frac{1}{n}$ l'algoritmo di Karger accumula la taglia del min-cut.

[3] cioè non becco nessun arco di taglio minimo C

N.	Size	Karger solution	Karger Time (s)
1	6	2	4.59E-5
2	6	1	4.97E-5
3	6	3	5.39E-5
4	6	4	5.37E-5
5	10	4	5.103E-4
6	10	3	4.918E-4
7	10	2	4.965E-4
8	10	1	4.971E-4
9	25	7	0.0306304
10	25	6	0.0957754
11	25	8	0.0318386
12	25	9	0.0320083
13	50	15	0.7410805
14	50	16	0.7322812
15	50	14	0.6886227
16	50	10	0.7194658
17	75	19	4.5581846
18	75	15	4.7667614
19	75	18	4.7321971
20	75	16	4.6194424
21	100	22	18.4590746
22	100	23	18.3235051
23	100	19	19.2442024
24	100	24	19.2161447
25	125	34	58.8190144
26	125	29	57.457255
27	125	36	70.1068629
28	125	31	58.5128556
29	150	37	164.4596518
30	150	35	159.8519787
31	150	41	164.7340277
32	150	39	162.4446075
33	175	42	354.6868445
34	175	45	355.8195941
35	175	53	376.6771329

Continua nella pagina seguente

Tabella 2 – *continuazione dalla pagina precedente*

N.	Size	Karger solution	Karger Time (s)
36	175	43	339.6508817
37	200	54	730.3265273
38	200	52	685.7037242
39	200	51	708.7966211
40	200	61	730.9270492

Tabella 2: Risultati della procedura *Karger* rispetto alla domanda 2

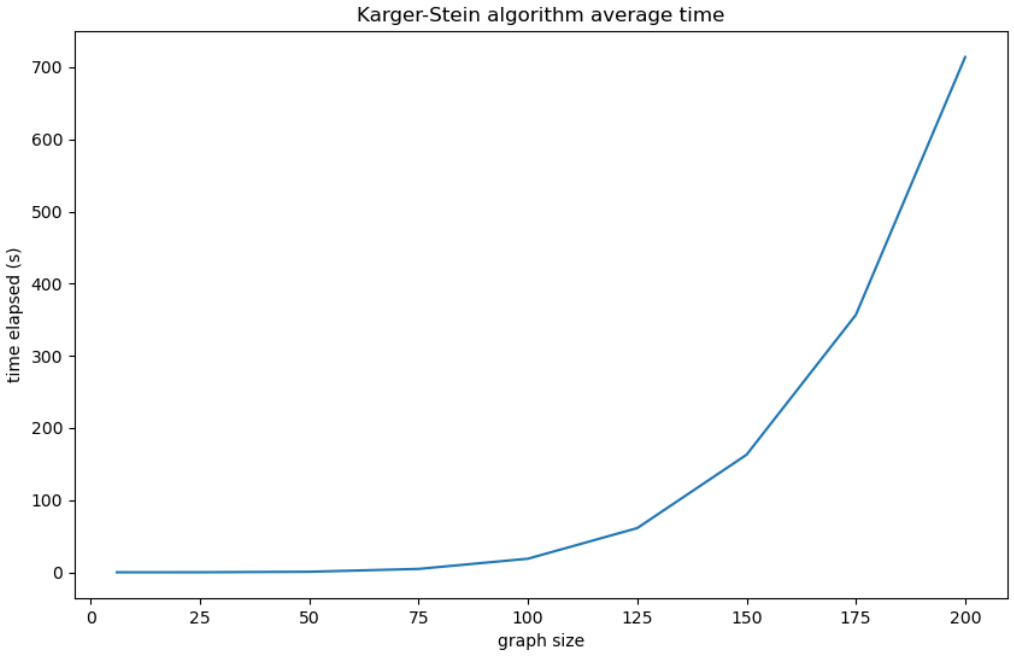

Figura 4: Analisi asintotica di `Karger`

La procedura `Karger` ha una complessità di$\mathcal{O}(n^4 \log n)$, infatti è possibile vedere in Figura 4 come rispetto a Figura 3[4] la curva inizia ad impennarsi fin da subito ma in modo più ripido.

[4]che si riferisce ad una curva asintotica di $\mathcal{O}(n^2)$

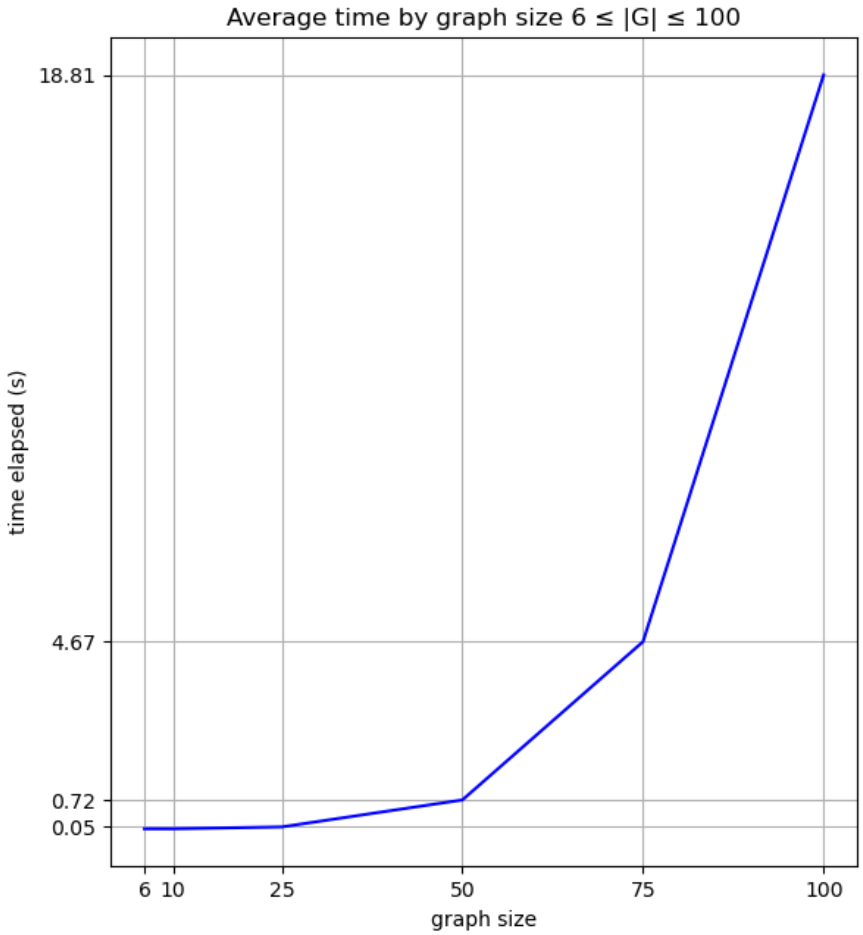

Figura 5: Analisi asintotica di `Karger` per i grafi da 6 a 100 nodi

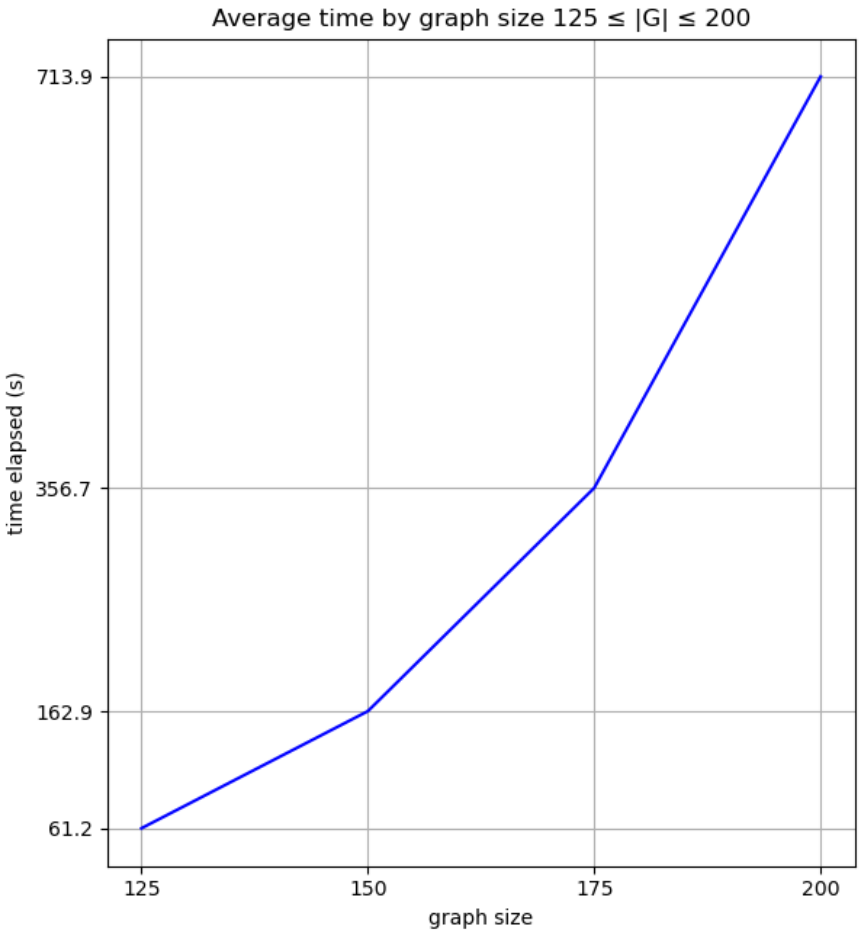

Figura 6: Analisi asintotica di `Karger` per i grafi da 125 a 200 nodi

La procedura `Karger` ha una complessità di$\mathcal{O}(n^4 \log n)$, che è possibile analizzare molto bene attraverso Figura 5 e Figura 6. Infatti è molto evidente come la curva dei grafici si impenni molto più velocemente rispetto `full_contraction` in Figura 3.

4.3 Domanda 3

Misurate il *discovery time* dell'algoritmo di Karger sui grafi del dataset. Il discovery time è il momento (in secondi) in cui l'algoritmo trova per la prima volta il taglio di costo mimimo. Confrontate il discovery time con il tempo di esecuzione complessivo per ognuno dei grafi nel dataset.

4.3.1 Svolgimento

N.	Size	Karger Tme (s)	Discovery Time (s)
1	6	4.59E-5	1.26E-5
Continua nella pagina seguente			

Tabella 3 – *continuazione dalla pagina precedente*

N.	Size	Karger Time (s)	Discovery Time (s)
2	6	4.97E-5	3.7E-6
3	6	5.39E-5	9.7E-6
4	6	5.37E-5	3.2E-6
5	10	5.103E-4	1.284E-4
6	10	4.918E-4	5.8E-6
7	10	4.965E-4	2.75E-5
8	10	4.971E-4	5.9E-6
9	25	0.0306304	7.73E-4
10	25	0.0957754	0.0115143
11	25	0.0318386	0.0077459
12	25	0.0320083	0.0040907
13	50	0.7410805	0.0138128
14	50	0.7322812	0.0478224
15	50	0.6886227	0.003738
16	50	0.7194658	0.0149178
17	75	4.5581846	0.0762589
18	75	4.7667614	0.0404856
19	75	4.7321971	0.0265994
20	75	4.6194424	0.1123946
21	100	18.4590746	0.1588621
22	100	18.3235051	0.0822596
23	100	19.2442024	0.0168746
24	100	19.2161447	0.0251183
25	125	58.8190144	0.4463958
26	125	57.457255	0.1294754
27	125	70.1068629	1.0395953
28	125	58.5128556	0.8252303
29	150	164.4596518	6.6862347
30	150	159.8519787	0.9646022
31	150	164.7340277	0.5240023
32	150	162.4446075	6.4803055
33	175	354.6868445	3.0379762
34	175	355.8195941	1.7371626
35	175	376.6771329	22.2364104
36	175	339.6508817	2.4007981

Continua nella pagina seguente

Tabella 3 – *continuazione dalla pagina precedente*

N.	Size	Karger Time (s)	Discovery Time (s)
37	200	730.3265273	7.7587212
38	200	685.7037242	0.6157363
39	200	708.7966211	3.3346295
40	200	730.9270492	12.4704872

Tabella 3: Risultati della procedura *Karger* rispetto alla domanda 3

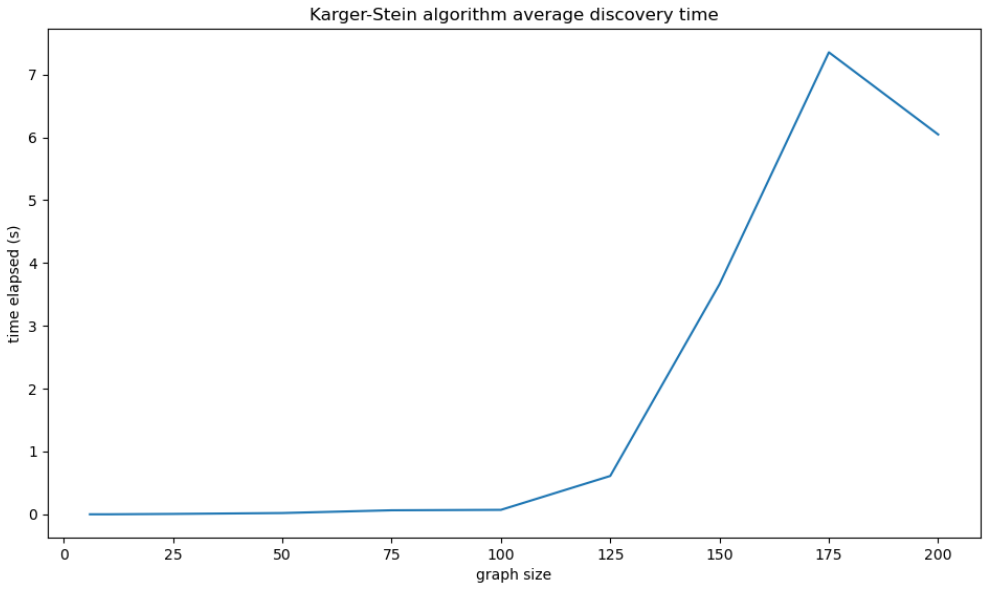

Figura 7: Analisi del `discovery_time`

Dato che l'algoritmo di Karger è un algoritmo randomizzato, non è possibile determinare quando e se otterrà la soluzione corretta, dunque è normale che in Figura 7 l'andamento non sia regolare.
É importante invece notare come il *discovery_time* sia in generale almeno 10 volte minore del tempo di esecuzione di `Kager`, questo vuol dire che la soluzione corretta viene trovata già prima delle $\frac{k}{10}$ iterazioni, con $k = n^2 \log n$. Da ciò si può dedurre che già dopo $\frac{k}{10}$ iterazioni la probabilità di trovare la soluzione corretta è già molto alta.
In particolare è interessante vedere in Figura 8, Figura 9, Figura 10 e Figura 11 la differenza tra *discovery_time* ed il tempo di esecuzione di `Karger` e di come il *discovery_time* sembri quasi lineare, rispetto a `Karger` che invece è esponenziale.

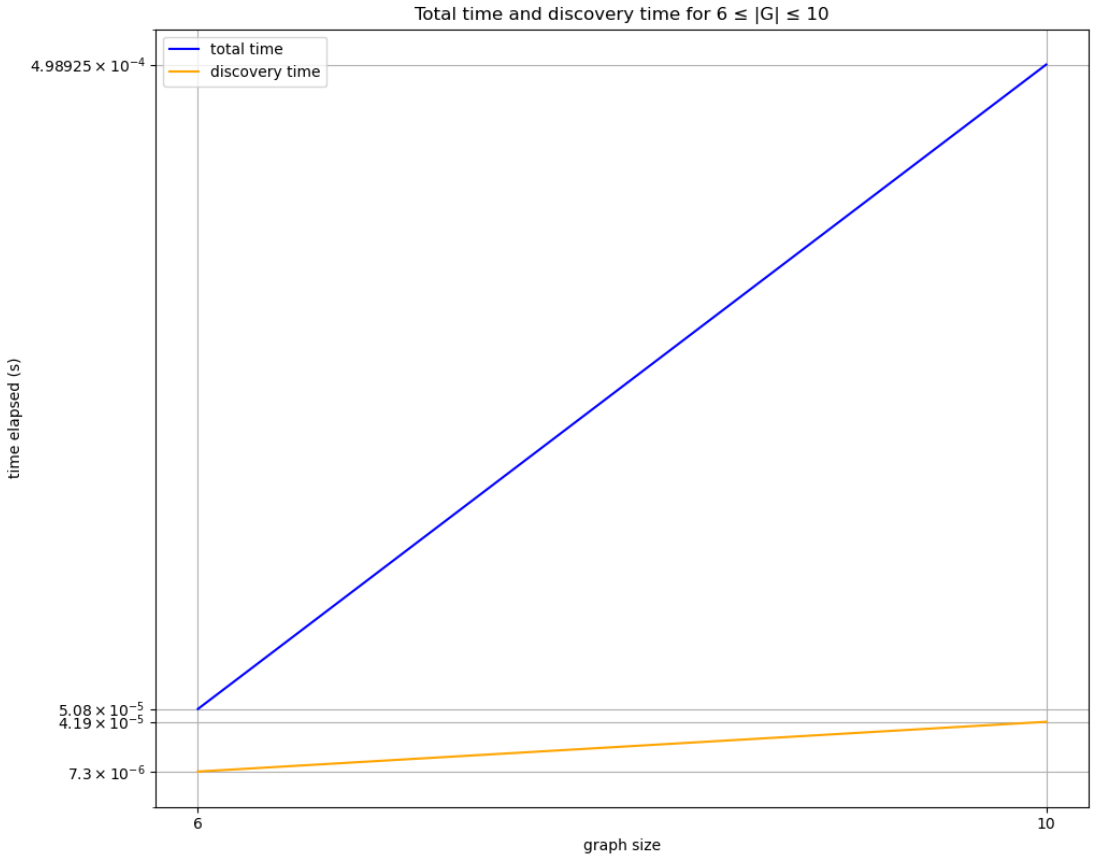

Figura 8: Analisi del `discovery_time`

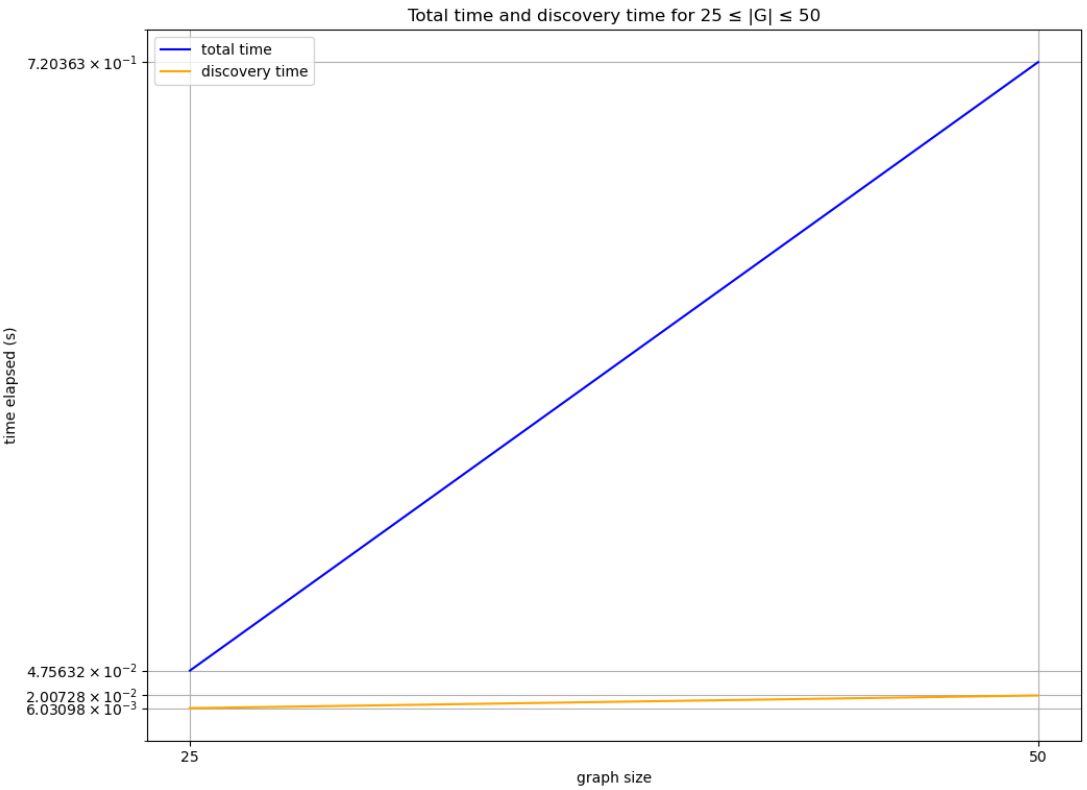

Figura 9: Analisi del `discovery_time`

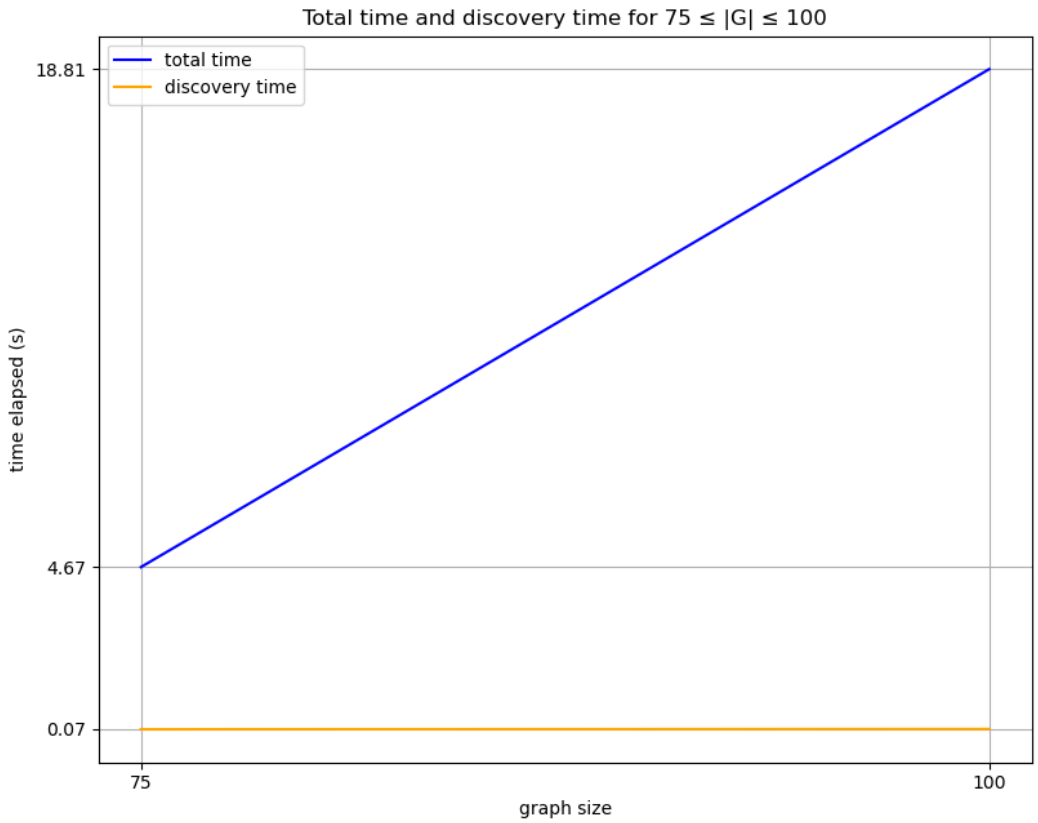

Figura 10: Analisi del `discovery_time`

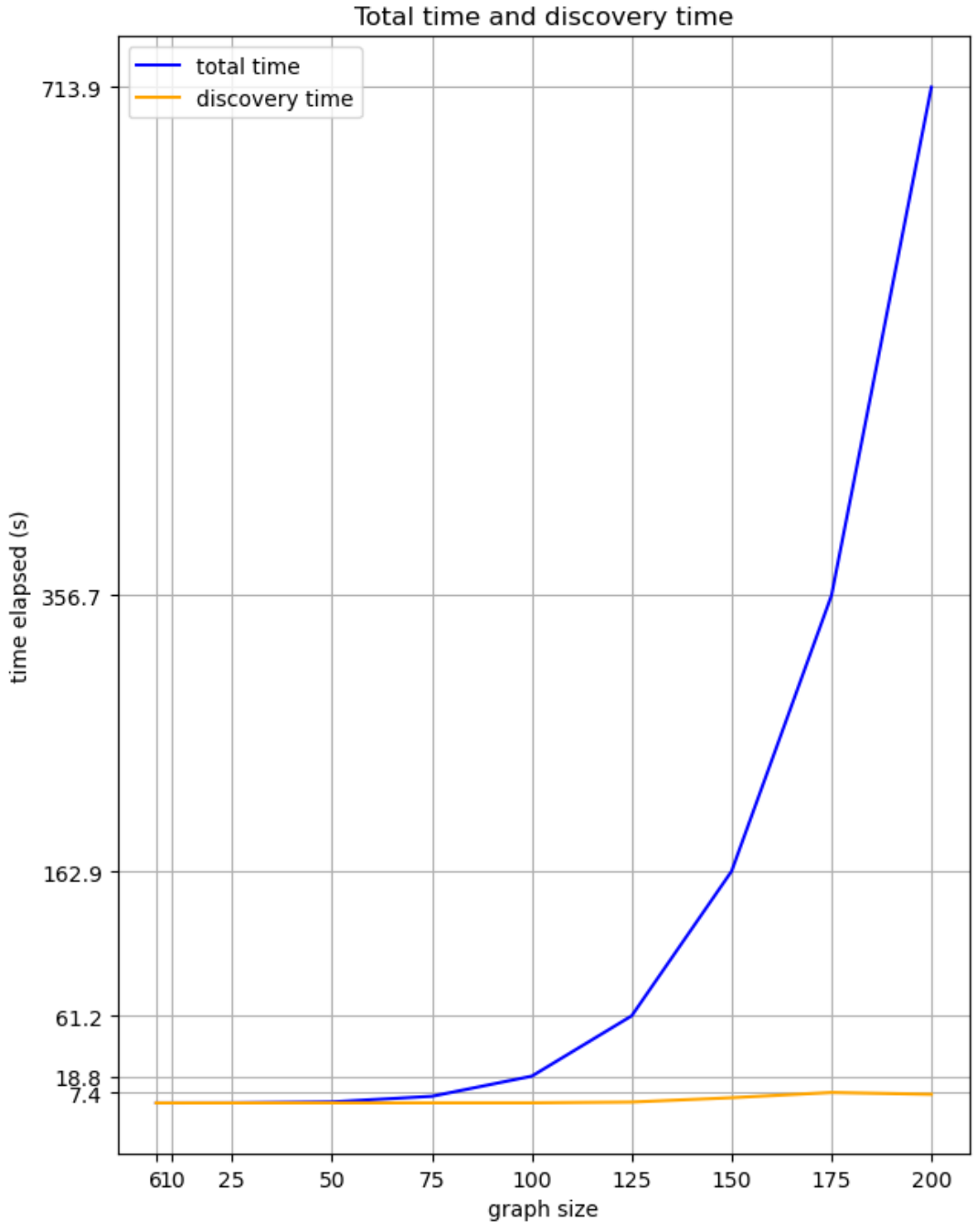

Figura 11: Analisi del `discovery_time`

4.4 Domanda 4

Per ognuno dei grafi del dataset, riportate il risultato risultato ottenuto dalla vostra implementazione, la soluzione attesa e l'errore relativo calcolato come $\frac{\text{SoluzioneTrovata} - \text{SoluzioneAttesa}}{\text{SoluzioneAttesa}}$.

4.4.1 Svolgimento

N.	Size	Karger Solution	Solution	Error (%)
1	6	2	2	0.0
2	6	1	1	0.0
3	6	3	3	0.0
4	6	4	4	0.0
5	10	4	4	0.0
6	10	3	3	0.0
7	10	2	2	0.0
8	10	1	1	0.0
9	25	7	7	0.0
10	25	6	6	0.0
11	25	8	8	0.0
12	25	9	9	0.0
13	50	15	15	0.0
14	50	16	16	0.0
15	50	14	14	0.0
16	50	10	10	0.0
17	75	19	19	0.0
18	75	15	15	0.0
19	75	18	18	0.0
20	75	16	16	0.0
21	100	22	22	0.0
22	100	23	23	0.0
23	100	19	19	0.0
24	100	24	24	0.0
25	125	34	34	0.0
26	125	29	29	0.0
27	125	36	36	0.0
28	125	31	31	0.0
29	150	37	37	0.0
30	150	35	35	0.0

Continua nella pagina seguente

Tabella 4 – *continuazione dalla pagina precedente*

N.	Size	Karger Solution	Solution	Error (%)
31	150	41	41	0.0
32	150	39	39	0.0
33	175	42	42	0.0
34	175	45	45	0.0
35	175	53	53	0.0
36	175	43	43	0.0
37	200	54	54	0.0
38	200	52	52	0.0
39	200	51	51	0.0
40	200	61	61	0.0

Tabella 4: Risultati della procedura *Karger* rispetto alla domanda 4

Abbiamo deciso di eseguire l'algoritmo di Karger senza alcun tipo di timeout, dato che coi grafi più grandi richiedeva neanche 15 minuti di tempo, e per tutti i grafi la soluzione corretta è stata trovata, avendo dunque un errore relativo sempre dello 0%.

5 Conclusioni

Nella realizzazione dell'elaborato abbiamo riscontrato qualche difficoltà nel cercare di capire quale fosse il modello per rappresentare i grafi più efficiente per eseguire l'algoritmo di Karger, in particolar modo per quanto riguardava la fase di contrazione dei lati, dato che quest'ultima doveva essere $\mathcal{O}(n)$. Abbiamo dunque deciso di creare un modello che occupasse più memoria, attraverso le matrici di adiacenza, in favore di una complessità migliore.
É stato inoltre interessante vedere come la soluzione ottima venisse trovata quasi sempre tra le prime $\frac{k}{10}$ iterazioni circa.
Ci riteniamo soddisfatti del lavoro svolto, poiché siamo riusciti ad evitare l'uso di un *timeout*: questo ci fa capire che siamo riusciti a rispettare la complessità, trovando nel 100% dei dataset forniti la soluzione aspettata in tempi relativamente brevi, all'incirca 12 minuti per i grafi più grandi da 200 nodi.