Università degli Studi di Padova

DIPARTIMENTO DI MATEMATICA "TULLIO LEVI-CIVITA"

Corso di Laurea Magistrale in Informatica

Laboratorio di Algoritmi Avanzati

Minimum Cut

Studente:
Alessio Lazzaron
Matricola 1242456

Studente:
Matteo Marchiori
Matricola 1236329

Anno Accademico 2019–2020

INDICE

1 INTRODUZIONE

La relazione descrive l'algoritmo implementato da Alessio Lazzaron e Matteo Marchiori per il terzo laboratorio del corso di algoritmi avanzati.

1.1 DESCRIZIONE DELL'ALGORITMO

L'unico algoritmo implementato per questo laboratorio è quello di Karger. In seguito poniamo lo pseudocodice e la parte interessante del codice in python per un rapido confronto, e alcune scelte fatte durante l'implementazione.

1.1.1 Algoritmo di Karger

```
Full_contraction(G=(V,E)):
    for i = 1 to |V| - 2 do:
        e = random(E)
        G' = (V, E') = G/e
        V = V'
        E = E'
    return |E|

Karger(G=(V,E), k):
    min = infinity
    for i = 1 to k do:
        t = Full_contraction(G)
        if t < min:
            min = t
    return min
```

```
def full_contraction(graph):
    nodes = graph.nodes()
    not_removed = list(nodes)
    for i in range(len(nodes)-2):
        src_index = random.randint(0, len(not_removed)-1)
        src = not_removed[src_index]
        adjacents_src = src.adjacents()
        dest = random.choice(adjacents_src)
        dest_index = dest.index()
        adjacents_dest = dest.adjacents()
        newname = graph.inc_lastnode()
        adjacents_src = [
            adjacent for adjacent in adjacents_src if adjacent != dest
        ]
        adjacents_dest = [
            adjacent for adjacent in adjacents_dest if adjacent != src
        ]
        newadjacents = adjacents_src + adjacents_dest
        newnode = Node(newname, dest_index)
        newnode.set_adjacents(newadjacents)
        for node in newadjacents:
            adjacents = node.adjacents()
            for i, adjacent in enumerate(adjacents):
                if adjacent is src or adjacent is dest:
                    adjacents[i] = newnode
        nodes[src_index] = None
        nodes[dest_index] = newnode
        del not_removed[src_index]

    mincut = 0
    for node in not_removed:
        if node is not None:
            for adjacent in node.adjacents():
                if adjacent is not None:
                    mincut += 1
            break
    return mincut
```

```
def karger(graph, k, ottimo):
    mincut = float("inf")
    original = copy.deepcopy(graph)
    time_fc = 0
    time_discovery = 0
    found = False
    for _ in range(k):
        fc_start = time.time()
        fc = full_contraction(graph)
        if fc == ottimo and not found:
            time_discovery = time.time()
            found = True
        elif fc < ottimo:
            time_discovery = time.time()
            ottimo = fc
        time_fc += time.time() - fc_start
        if fc < mincut:
            mincut = fc
        graph = copy.deepcopy(original)
    time_fc = time_fc / k
    return mincut, time_fc, time_discovery
```

Nell'algoritmo *full_contraction* la contrazione di due nodi avviene creando un nuovo nodo nel grafo che unisce due nodi presi randomicamente: il primo dai nodi non ancora eliminati mentre il secondo tra i possibili nodi adiacenti al primo, in modo da avere almeno un arco esistente tra i due. Inoltre il nuovo nodo avrà come nodi adiacenti i nodi adiacenti dei nodi scelti e verrà memorizzato al posto del secondo nodo, sovrascrivendolo, mentre il primo verrà impostato a None così eliminandolo. Alla fine la procedura restituisce il taglio trovato.

La funzione chiamata *Karger*, invece, fa eseguire la procedura *full_contraction* k volte, dove k viene determinato con la seguente formula:

$$\frac{n^2}{2} \cdot ln(n)$$

Inoltre prima di ogni nuova chiamata alla funzione *full_contraction* il grafo viene impostato alla versione fornita in input e calcolato il tempo di esecuzione. Tuttavia nella stima dei tempi dell'algoritmo di Karger è presente uno scarto temporale che dipende dalla dimensione del grafo e dal parametro k.

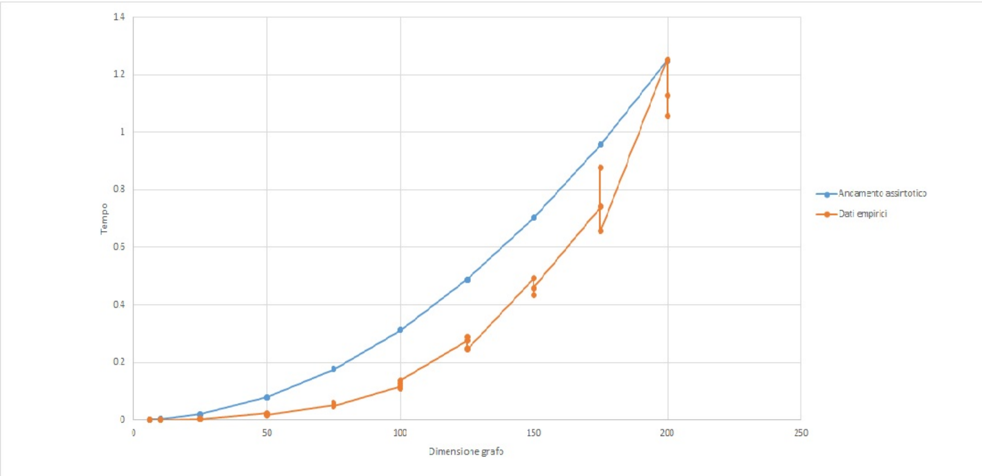
Per tutti gli algoritmi implementati abbiamo cercato di usare, per quanto possibile, la definizione di oggetti e la definizione di metodi previsti da python, in modo da mantenere per quanto possibile il codice comprensibile.

2 RISULTATI

In questa sezione mostriamo i risultati ottenuti dall'algoritmo, ovvero i tempi medi impiegati per full contraction e karger, il discovery time, la soluzione trovata, la soluzione attesa e l'errore relativo.

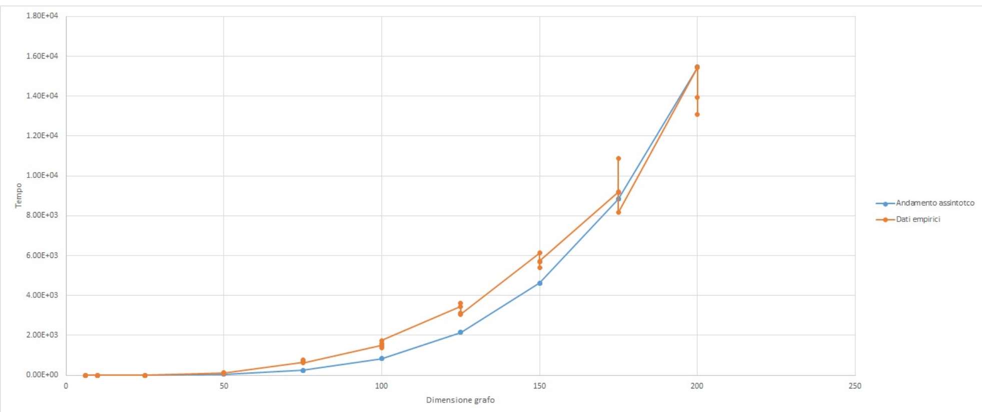
2.1 GRAFICI

2.1.1 Full contraction



Il presente grafico illustra i tempi di calcolo della procedura Full_Contraction sui grafi del dataset al variare del numero di vertici. Confrontiamo con la complessità asintotica stimata di $O(n^2)$.

2.1.2 Karger



Il presente grafico illustra i tempi di calcolo dell'algoritmo di Karger sui grafi del dataset al variare del numero di vertici. La probabilità di errore è uguale a $\frac{1}{n}$ di sbagliare per grafi inferiori a 75 nodi. Tale scelta è dovuta a motivi di tempo, come illustrato in sezione 3 a pagina 8. Confrontiamo con la complessità asintotica stimata di $O(n^4 log(n))$.

2.2 TABELLA

Istanza	Tempo di una Full contraction (s)	Discovery time (s)	Tempo totale (s)	Soluzione trovata	Soluzione ottimo	Errore relativo
input_01_6	0.00006420	0.00131726	0.00717187	2	2	0.0
input_02_6	0.00006785	0.00022292	0.03581977	1	1	0.0
input_03_6	0.00007611	0.00034499	0.00778651	3	3	0.0
input_04_6	0.00007507	0.00027013	0.00872898	4	4	0.0
input_05_10	0.00019939	0.00230646	0.05261183	4	4	0.0
input_06_10	0.00016412	0.00085616	0.04731894	3	3	0.0
input_07_10	0.00017446	0.00273633	0.04886866	2	2	0.0
input_08_10	0.00017658	0.00046015	0.04921460	1	1	0.0
input_09_25	0.00198467	0,00161695	2.71520424	7	7	0.0
input_10_25	0.00181535	0.36311722	2.52611518	6	6	0.0
input_11_25	0.00202685	0.01288033	2.87397146	8	8	0.0
input_12_25	0.00285040	0.59262013	3.88847303	9	9	0.0
input_13_50	0.02205332	1.16963172	119.23173952	15	15	0.0
input_14_50	0.02155554	0.60323572	115.70873451	16	16	0.0
input_15_50	0.01671049	2.41417027	91.38596511	14	14	0.0
input_16_50	0.01808762	1.16370130	98.28982854	10	10	0.0

Istanza	Tempo di una Full contraction (s)	Discovery time (s)	Tempo totale (s)	Soluzione trovata	Soluzione ottimo	Errore relativo
input_17_75	0.05015093	7.60165477	651.79445624	19	19	0.0
input_18_75	0.05971390	2.65774322	769.72895575	15	15	0.0
input_19_75	0.05304535	0.42174196	687.63417149	18	18	0.0
input_20_75	0.04822129	0.92826390	628.02836180	16	16	0.0
input_21_100	0.11767139	0.41041040	1496.49857569	22	22	0.0
input_22_100	0.10784917	24.04785800	1376.93518615	23	23	0.0
input_23_100	0.12643225	2.35784650	1606.15633321	19	19	0.0
input_24_100	0.13665407	32.93737745	1735.68474460	24	24	0.0
input_25_125	0.27526138	10.41306686	3446.43969345	34	34	0.0
input_26_125	0.24957255	8.42978859	3130.82415271	29	29	0.0
input_27_125	0.28772694	69.54419684	3600.48817873	36	36	0.0
input_28_125	0.24376734	12.67581010	3059.99670696	31	31	0.0
input_29_150	0.49271837	35.33633661	6129.14617682	37	37	0.0
input_30_150	0.45532591	83.78166389	5669.51948738	35	35	0.0
input_31_150	0.43285431	32.14746523	5394.68515444	41	41	0.0
input_32_150	0.45989296	58.75906992	5726.16600871	39	39	0.0
input_33_175	0.74070684	66.79247236	9177.23712540	42	42	0.0
input_34_175	0.74288779	49.13439870	9205.83131051	45	45	0.0
input_35_175	0.87808876	45.06704497	10859.42819190	53	53	0.0
input_36_175	0.65710520	6.00958300	8155.21835065	43	43	0.0
input_37_200	1.25286142	20.91466975	15461.59397507	54	54	0.0
input_38_200	1.05687901	163.0276653	13067.47582197	52	52	0.0

Istanza	Tempo di una Full contraction (s)	Discovery time (s)	Tempo totale (s)	Soluzione trovata	Soluzione ottimo	Errore relativo
input_39_200	1.12854942	13.06262207	13932.64299393	51	51	0.0
input_40_200	1.25152820	93.24242926	15435.27277613	61	61	0.0

Tabella 1: Risultati

3 ORIGINALITÀ

In questa sezione vengono presentate le originalità che abbiamo inserito nella nostra implementazione dell'algoritmo.

È possibile avere vari gradi di precisione, nel calcolo della soluzione, cambiando il parametro d passato da linea di comando. Più è alt il valore di questo parametro più è bassa la probabilità di insuccesso dell'algoritmo di Karger. Di default il parametro è impostato a 1 per avere una probabilità di insuccesso pari a $\frac{1}{n}$, dove n è il numero dei nodi del grafo.

Nell'implementazione sviluppata abbiamo impostato k a un valore inferiore quando la taglia del grafo supera i 75 nodi, ovvero a una costante pari a: $\frac{75^2}{2} \cdot ln(75)$, questo per evitare tempistiche improponibili per l'esecuzione dell'algoritmo.

4 CONCLUSIONI

4.1 RISULTATI OTTENUTI

Dai risultati ottenuti è visibile come i tempi trovati rispecchino la complessità dell'algoritmo stimata.

Essendo il discovery time molto basso, specie nei grafi di dimensioni maggiori, abbiamo preferito riportarlo solamente in tabella (dal grafico non era esposto chiaramente). Si spiega con il fatto che la probabilità di errore decresce con il numero di nodi presenti nel grafo, secondo la stima di k.

L'errore calcolato è sempre pari a 0, infatti la probabilità di errore si è dimostrata essere così bassa per ogni grafo da trovare sempre il risultato aspettato. Non è detto che questo avvenga sempre.

Per avere una stima più corretta dei tempi medi impiegati dall'algoritmo di Karger sarebbe necessario togliere il tempo impiegato per la copia del grafo, che potrebbe influire negativamente nelle istanze di dimensione maggiore.