Dipartimento di Informatica
Corso di Laurea Magistrale in Informatica

Mid Term:
Analysing the Kademlia DHT

Studente:
Luca Corbucci

Peer to Peer Systems and Blockchains
Anno Accademico 2018/2019

Indice

1 Introduzione

In questo Mid Term l'obiettivo è simulare il funzionamento della costruzione delle Routing Table dei nodi che entrano a far parte della rete di Kademlia. Una volta prodotta la simulazione è stato generato ed analizzato un grafo che rappresenta le connessioni tra i nodi entrati nella rete.

2 Contenuto del pacchetto del progetto

I dati relativi al Midterm sono distribuiti in 4 cartelle:

- Nella cartella "src" è presente tutto il codice che è stato prodotto per la simulazione
- Nella cartella "Esperimenti" sono presenti dei risultati e dei grafici relativi ad analisi effettuate sul grafo che non ho ritenuto troppo significative e che non ho inserito nella relazione.
- Nella cartella "Documentazione" è presente tutta la documentazione relativa al codice del midterm. Nel paragrafo 3.2 verrà descritta in modo più accurato.
- La cartella "Relazione" contiene il report finale.

3 Codice

Il Mid Term è stato sviluppato in Python. Tutti i test sono stati effettuati utilizzando la versione 2.7.15 di Python.

3.1 Esecuzione del codice

La simulazione e l'analisi del grafo che viene prodotto può essere avviata da terminale. Per svolgere la simulazione è necessario eseguire il file *__init__.py* con i seguenti parametri:

- Il primo parametro indica quanti nodi vogliamo inserire nella rete durante la simulazione
- Il secondo parametro indica la lunghezza dell'identificatore di ogni nodo
- Il terzo parametro ci permette di scegliere la lunghezza delle K-Bucket List delle routing table
- Al quarto parametro possiamo passare solamente valore 0, 1 o 2 e serve per indicare il tipo di gestione delle K-Bucket List nel caso in cui dovessero riempirsi. Scrivendo 1 manterremo i nodi più vecchi all'interno della bucket list scartando i più recenti (successivamente questa modalità verrà spesso indicata con "nodi vecchi"). Scrivendo 0 invece avremo il comportamento opposto (questa modalità verrà poi indicata con "nodi nuovi"). Scrivendo 2 nel caso di KBucket List piena verrà deciso casualmente se inserire il nuovo nodo o mantenere il vecchio. Questa terza opzione è stata pensata per simulare in modo più accurato il funzionamento di Kademlia che effettua un ping per capire se il nodo è ancora vivo o no.
- Anche al quinto parametro può essere assegnato il valore 0 o 1. Questo parametro viene utilizzato per indicare se vogliamo effettuare un'analisi sul grafo completo (1) o se invece vogliamo eseguire delle analisi "temporali" (0) analizzando quindi parte dei dati che sono stati inseriti in modo da capire in che modo si evolve la rete.

- L'ultimo parametro è necessario solamente se abbiamo scelto di svolgere un'analisi temporale, permette infatti di scegliere in quanti blocchi dividere i dati che vengono inseriti e quindi quante analisi svolgere.

3.2 Documentazione

Tutto il codice sorgente è stato commentato cercando di rispettare il più possibile le convenzioni di Docstring [1]. Questo ha permesso di generare una documentazione più leggibile e facilmente consultabile mediante l'utilizzo di pdoc [2]. La documentazione, presente all'interno della cartella docs, è in formato html, per ogni classe che è stata definita e per ogni metodo è presente una descrizione dei parametri in input, del tipo di output e del comportamento atteso. Aprendo il file index.html è possibile navigare tra le varie classi e tra le varie descrizioni.

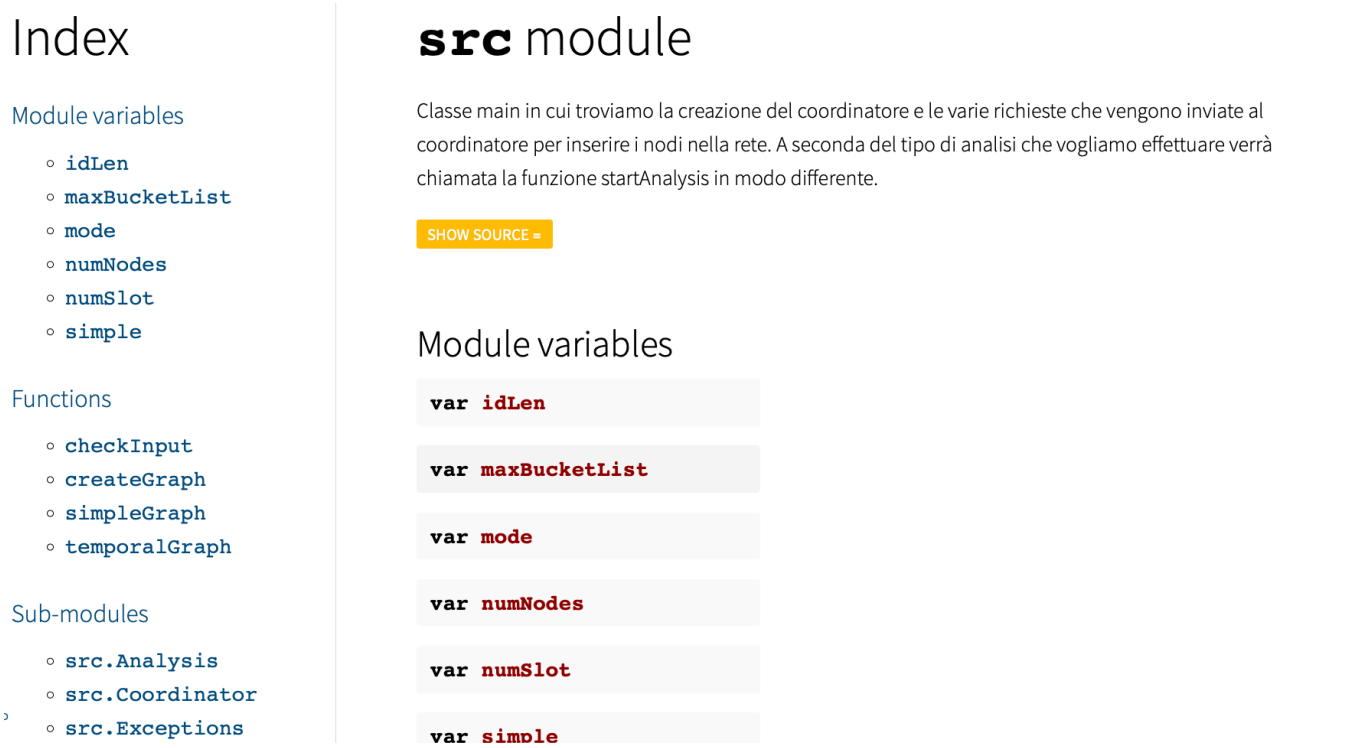


Figura 1: Documentazione generata con pdoc

3.3 Scelte implementative

3.4 File __init__.py

Il file *__init__.py* è il primo che viene eseguito, al suo interno si effettuano per prima cosa dei check sui parametri che sono stati passati. Si verifica che il numero di parametri sia corretto, che siano valori maggiori di 0 e che la lunghezza degli identificatori sia sufficiente per rappresentare il numero di nodi che abbiamo scelto. Una volta superata questa prima fase viene generato il coordinatore che è utilizzato per svolgere tutte le operazioni nella "rete". A seconda delle scelte effettuate verrà chiamata la funzione *simpleGraph*() o *temporalGraph*(). La prima inserisce tutti i nodi all'interno della rete per poi eseguire l'analisi. La seconda invece divide i dati da inserire in vari blocchi e per ogni blocco di nodi svolge l'analisi sul grafo "parziale" che viene generato.

Sempre all'interno del file *__init__.py* troviamo la funzione *createGraph*() che crea il grafo su cui poi verrà fatta l'analisi.

3.4.1 File Coordinator.py

La parte più importante del progetto è sicuramente la classe *Coordinator* che viene utilizzata per la gestione della rete Kademlia. Il coordinatore crea un primo nodo che viene aggiunto all'interno della rete. A tutti i nodi che vengono inseriti il coordinatore assegna un id univoco generato casualmente e di lunghezza fissata. Il nodo che viene generato nel momento in cui viene creato il coordinatore sarà il "nodo bootstrap" del primo nodo che chiederà l'accesso alla rete.

L'inserimento degli altri nodi all'interno della rete viene effettuato mediante l'esecuzione del metodo join che svolge le seguenti operazioni:

- Per prima cosa il nodo che vuole entrare nella rete invia una richiesta findNode al nodo bootstrap. In questo modo il bootstrap viene a conoscenza dell'esistenza del nodo che vuole entrare nella rete e inserisce il suo ID nella sua routing table. Allo stesso modo anche il joiningNode inserirà nella sua routing table l'id del nodo bootstrap.
- Per ogni bucket della routing table del joiningNode viene generato un ID random (appartenente a quel bucket). Questo ID viene inviato al bootstrap node tramite la funzione *Lookup*. Il bootstrap node per ogni ID che riceve, restituisce al richiedente al più K nodi che hanno un identificatore vicino ad ID. Per prima cosa viene controllato il bucket dove il bootstrap node inserirebbe il nodo con identificatore ID e poi nel caso in cui non si dovesse arrivare ad avere K nodi verranno controllati i bucket vicini salendo di 1 e scendendo di 1 ad ogni iterazione. È anche possibile che il bootstrap node restituisca meno di K nodi, questo accade quando il bootstrap node conosce meno di K nodi.
- Una volta che il joining Node ha ricevuto per ogni bucket list al più K nodi, utilizza questa informazione per cercare di aumentare i dati contenuti all'interno della sua routing table. Ognuno dei nodi restituiti dal bootstrap node viene contattato con un findNode in modo da farsi conoscere. Successivamente viene inviata una lookup con un ID che appartiene al bucket dove si trova quel nodo e si inseriscono i nodi che si ottengono all'interno della routing table, come già successo con il bootstrap node.

Il calcolo della distanza tra gli identificatori è stato definito nella funzione *computeDistance* del coordinatore. Per la distanza è stato effettuato lo xor come indicato nel paper di Kademlia. Effettuando delle prove però il calcolo dello xor (a seconda della dimensione dell'identificatore) richiedeva un tempo piuttosto elevato. Dato che ho utilizzato il calcolo della distanza solamente per capire in quale bucket inserire l'identificatore di un nodo, ho fatto in modo che il calcolo dello xor si interrompesse al primo 1 trovato nella soluzione. Trovare il primo 1 mi permette di capire in quale bucket inserire l'identificatore.

3.4.2 File KBucketList.py

Nel file *KBucketList.py* viene definita la KBucket List che poi verrà utilizzata all'interno della Routing Table. La KBucket list è stata implementata utilizzando un array che deve avere una dimensione massima fissata. La procedura di inserimento di un nuovo id all'interno della KBucket list controlla che l'array non sia già pieno ed eventualmente inserisce nell'array la nuova informazione. Nel caso in cui l'array dovesse già essere pieno abbiamo tre possibilità:

- Se abbiamo scelto di voler mantenere i nodi più vecchi, l'informazione riguardante il nuovo nodo che entra nella rete viene scartata.
- Se abbiamo scelto di voler inserire i nodi più recenti invece viene eliminato il primo elemento dell'array e si inserisce in fondo l'identificatore del nuovo nodo.
- Nella modalità random viene scelto casualmente se il nuovo nodo deve essere inserito nella KBucket List o no. Se non lo inseriamo, il nodo più vecchio della lista viene estratto e poi inserito in fondo. Se invece inseriamo il nuovo rimuoviamo il più vecchio e aggiungiamo il nuovo in fondo.

3.4.3 File RoutingTable.py

La routing table di ogni nodo della rete è stata implementata utilizzando un array di KBucketList. La lunghezza della routing table dipende dal parametro che mi indica la dimensione degli identificatori dei nodi che entrano nella rete.

3.4.4 File Map.py

La classe map viene utilizzata solamente per eseguire e memorizzare il mapping tra gli identificatori dei nodi della rete e un intero, questo mapping viene utilizzato successivamente per la creazione del grafo della rete. Quindi ad esempio potremmo avere un mapping del tipo 100101:5 001100:6. Poi all'interno del file usato per generare il grafo avremo delle coppie 5 6 per indicare che i nodi 5 e 6 sono connessi. Maggiori informazioni riguardo alla creazione del grafo sono presenti nella sezione Studio del Grafo.

3.4.5 File Analysis.py

Il file *Analysis.py* contiene i vari metodi che vengono utilizzati per eseguire le analisi sul grafo. Nella funzione *startAnalysis*() viene fatta una distinzione tra le analisi che devono essere svolte solamente sul grafo completo e quelle che invece devono essere svolte anche sui grafi parziali. Le varie funzioni presenti in questo file producono dei risultati che vengono memorizzati in un file di testo. Per alcune delle analisi invece si producono dei grafici che vengono salvati nella cartella ./results.

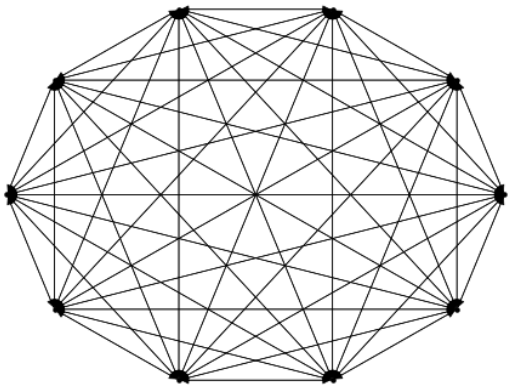
4 Studio del grafo

4.1 Generazione del grafo

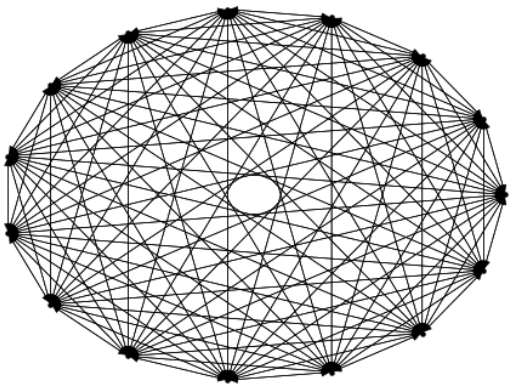
Il grafo è stato generato utilizzando la libreria NetworkX [3], quest'ultima è stata sfruttata anche per svolgere le analisi. Per generare il grafo è stato creato un file di testo in cui ogni riga rispetta il formato "NodoDiOrigine NodoDestinatario". Questo file viene creato dal coordinatore e poi viene letto nel momento in cui si va a creare il grafo che, in questo caso, è orientato.

4.2 Analisi effettuate

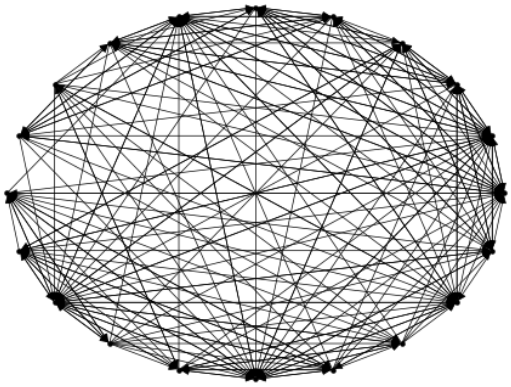
Come già indicato in precedenza, la parte relativa all'analisi del grafo è stata definita nel file *Analysis.py*. Le analisi che vengono eseguite dipendono dal tipo di grafo che viene utilizzato.



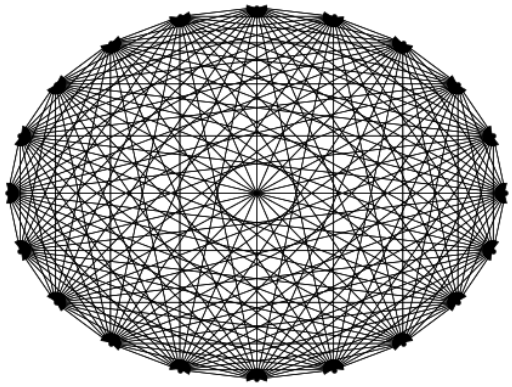
(a) Grafo con 10 nodi e KBucket List di lunghezza 20



(b) Grafo con 15 nodi e KBucket List di lunghezza 20



(c) Grafo con 20 nodi e KBucket List di lunghezza 5



(d) Grafo con 20 nodi e KBucket List di lunghezza 20

Figura 2: Rappresentazione grafica dei grafi della rete Kademlia

Se stiamo eseguendo l'analisi temporale, su ognuno degli snapshot vengono effettuate le seguenti analisi:

- Calcolo del numero di archi nel grafo
- Calcolo del numero di nodi nel grafo
- Calcolo del numero di componenti connesse
- Calcolo del numero di componenti fortemente connesse
- Diametro del grafo
- Lunghezza media dei cammini minimi nel grafo definita come:

$$a = \sum_{s,t \in V} \frac{d(s,t)}{n(n-1)} \tag{1}$$

Dove V è l'insieme dei nodi del grafo, d(s,t) è il cammino minimo tra s e t e n è il numero di nodi del grafo.

L'esecuzione di questo tipo di analisi ha permesso di capire l'evoluzione del grafo con l'aumentare dei nodi inseriti all'interno della rete. Queste stesse analisi appena descritte vengono effettuate anche sul grafo completo, a cui vanno però aggiunge anche le seguenti:

- Distribuzione del grado entrante dei nodi
- Distribuzione del grado uscente dei nodi
- Centralità armonica di ogni nodo u del grafo, definita come la somma del reciproco dei cammini minimi tra tutti gli altri nodi e u.
- Clustering Coefficient

5 Risultati dell'analisi del grafo

5.1 Analisi sul grafo completo (K-Bucket list con vecchi nodi)

Una prima analisi sul grafo completa è stata effettuata prendendo in considerazione 10.000 nodi e delle KBucketList di dimensione 20. Le KBucket List in questa prima analisi mantengono al loro interno i nodi vecchi scartando i nuovi nel caso in cui dovesse riempirsi tutta la lista. Questo primo grafo che è stato prodotto contiene al suo interno 657565 archi e presenta un'unica componente connessa. Il diametro del grafo ha una lunghezza pari a 3 e le componenti fortemente connesse sono 1200. La lunghezza media del cammino minimo tra i nodi del grafo è 1.98. Il clustering coefficient calcolato per questo grafo è pari a 0.6278.

È stata poi studiata la distribuzione del grado entrante ed uscente dei nodi del grafo. In particolare per quanto riguarda il grado entrante abbiamo una distribuzione di tipo power law come è possibile vedere nella Figura 5

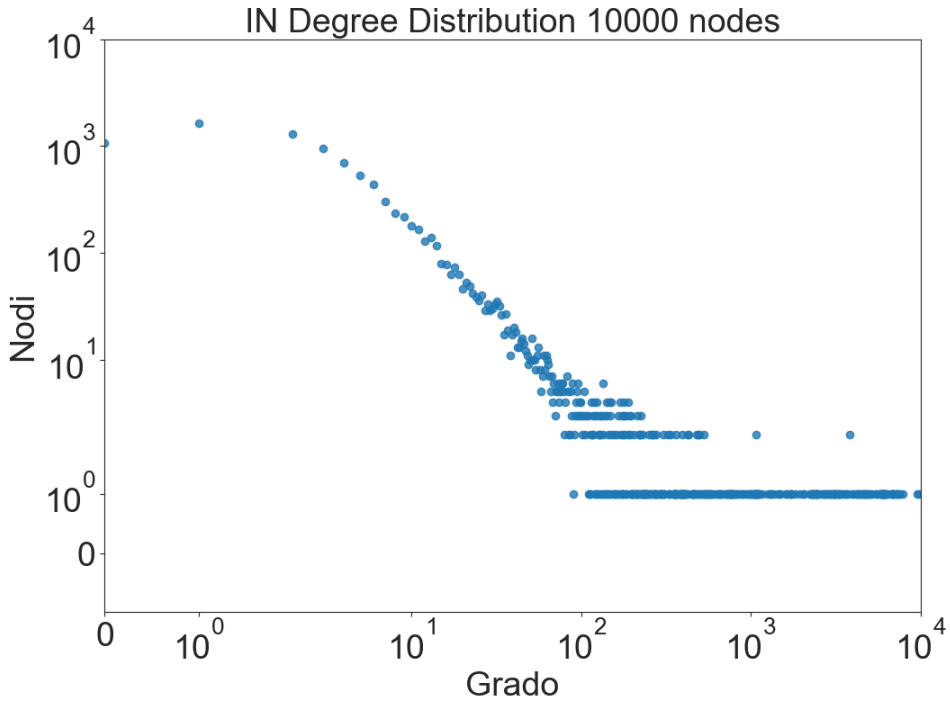
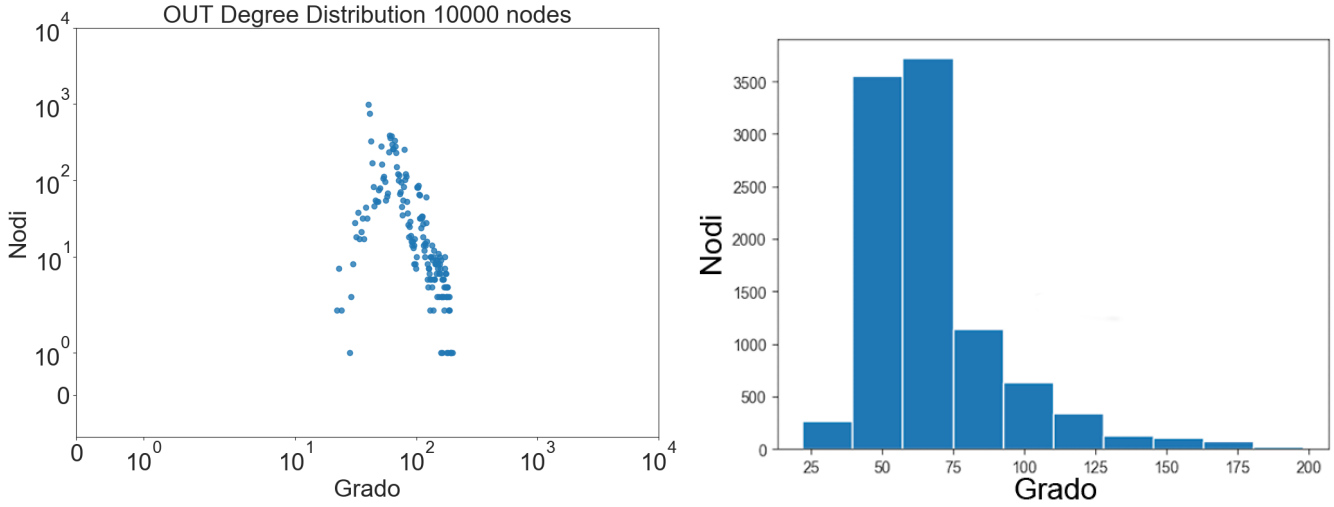


Figura 3: Distribuzione grado entrante: 10.000 nodi, KBucketList di dimensione 20 e nodi vecchi

Per il grado uscente abbiamo un comportamento differente, non di tipo power law. Per cercare di rappresentare meglio questi dati, è stato prodotto un istogramma (Figura 6b) e il grafico della distribuzione in scala logaritmica (Figura 6a).



(a) Distribuzione del grado uscente

(b) Istogramma

Figura 4: Grado uscente. 10000 Nodi, KbucketList di dimensione 20 e nodi vecchi

Sempre in questo grafo è stata calcolata anche la centralità armonica, l'analisi ci ha restituito un dizionario del tipo *{Nodo: Valore Centralità Armonica}*. Non ho trovato molto interessanti questi risultati perchè trattandosi di una simulazione non è possibile trovare una corrispondenza "reale" per i nodi più centrali nella rete. Nella cartella Esperimenti sono però presenti dei grafici che avevo prodotto relativi alla centralità e che non ho inserito in questa relazione. Sempre nella cartella Esperimenti sono presenti i grafici e e i risultati relativi al calcolo della centralità in base ad altre due metriche ovvero "Out Degree" e "In Degree".

5.2 Analisi sul grafo completo (KBucket List con nuovi nodi)

Un secondo esperimento è stato svolto considerando sempre il grafo completo con 10.000 nodi e delle bucket list lunghe al più 20 modificando però il modo in cui vengono gestiti gli inserimenti dei nuovi nodi all'interno delle Bucket List. In questo caso infatti nel caso in cui la KBucketList dovesse risultare piena, il primo nodo della lista fa posto al nuovo nodo che quindi si posiziona in fondo. Le analisi effettuate sono le stesse mostrate nel precedente paragrafo. Abbiamo 649218 archi con una sola componente connessa e un diametro che anche in questo caso ha una lunghezza pari a 3. Il dato più interessante rispetto al precedente esperimento è quello relativo al numero delle componenti fortemente connesse, in questo caso sono 42, quindi un valore molto più basso. Varia anche il Clustering coefficient medio che scende ad un valore pari a 0.2716, quindi molto più basso rispetto al primo esperimento effettuato. Rispetto al precedente esperimento non varia molto la lunghezza media dei cammini minimi che in questo caso è 2.021.

Per quanto riguarda la distribuzione del grado entrante si verifica un comportamento simile al caso precedente con una power law.

Anche per quel che riguarda la distribuzione del grado uscente abbiamo un comportamento simile al primo esperimento:

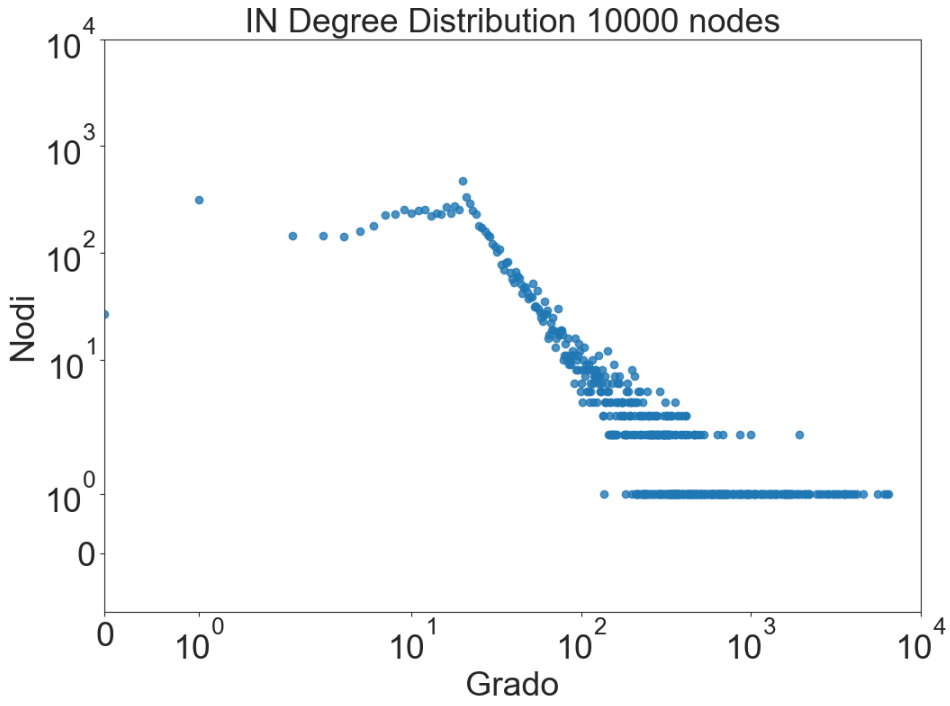
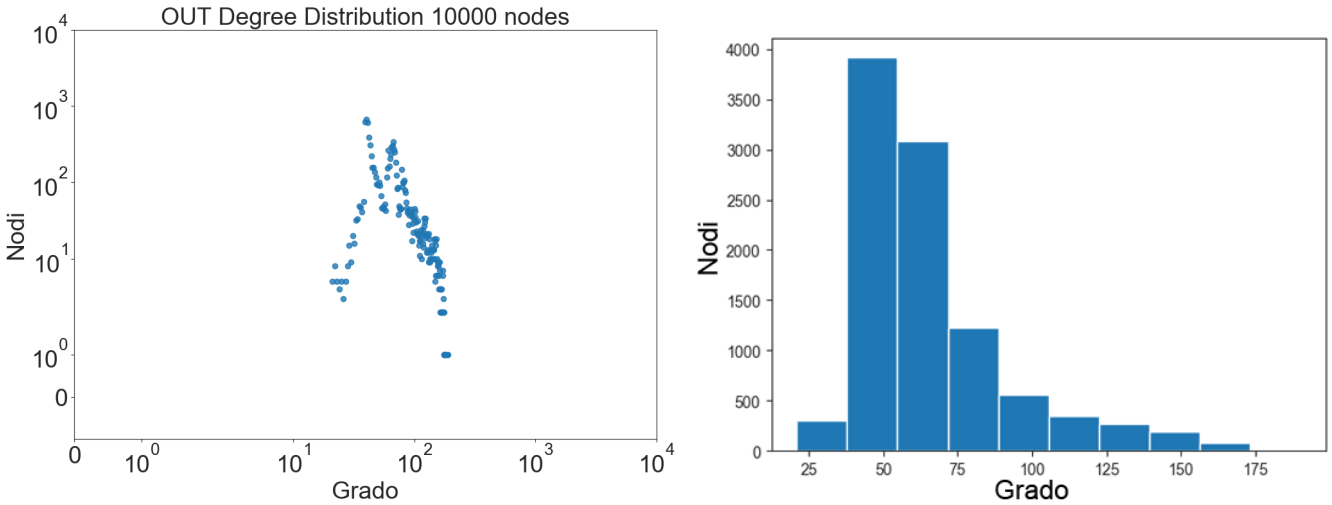


Figura 5: Distribuzione grado entrante: 10.000 nodi, KBucketList di dimensione 20 e nodi nuovi



(a) Distribuzione del grado uscente

(b) Istogramma

Figura 6: Grado uscente. 10000 Nodi, KbucketList di dimensione 20 e nodi nuovi

Anche in questo grafo è stata calcolata la centralità armonica, il risultato è presente nalla cartella Esperimenti.

5.3 Analisi sul grafo completo (KBucket list random)

Per questa terza analisi sul grafo con 10.000 è stata considerata una gestione Random della KBucket List (nei paragrafi precedente è stato indicato il funzionamento). Le KBucket list hanno dimensione 20 e le analisi effettuate sono le stesse viste nelle precedenti sezioni. Nel grafo abbiamo 694625 archi con un diametro di lunghezza 3 e un'unica componente connessa. Il numero di componenti

fortemente connesse è 255. La lunghezza media dei cammini minimi è 1.9929 mentre il valore medio del clustering coefficient è 0.3352.

Nella Figura 7 abbiamo la distribuzione del grado entrante che, anche in questo caso ha un comportamento di tipo power law.

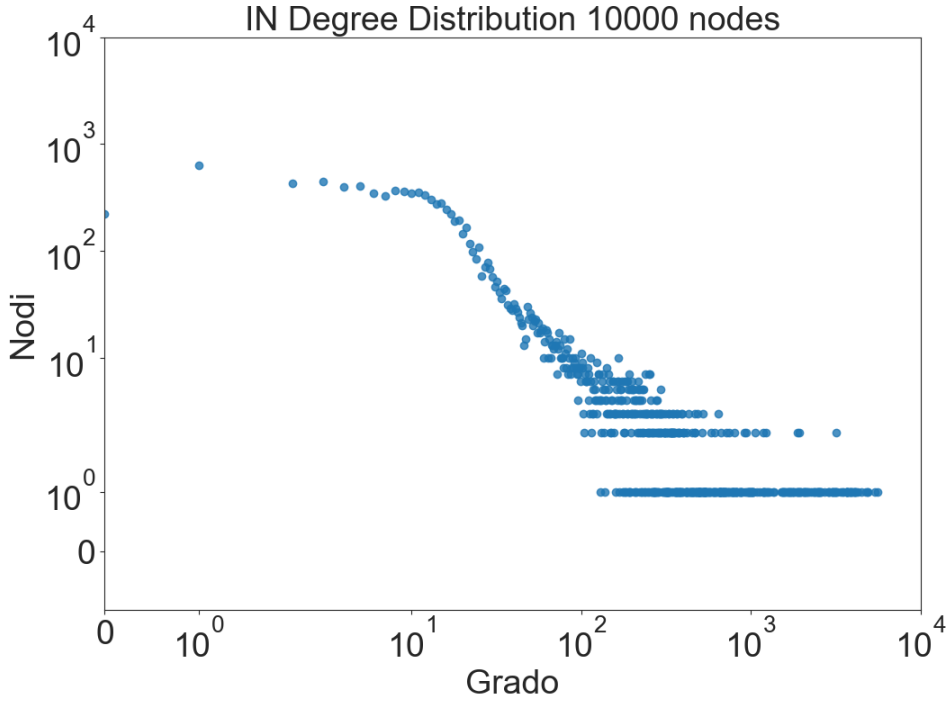
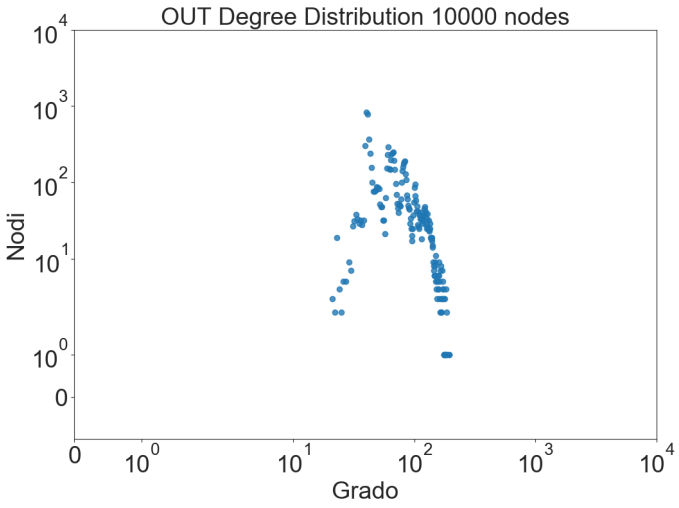
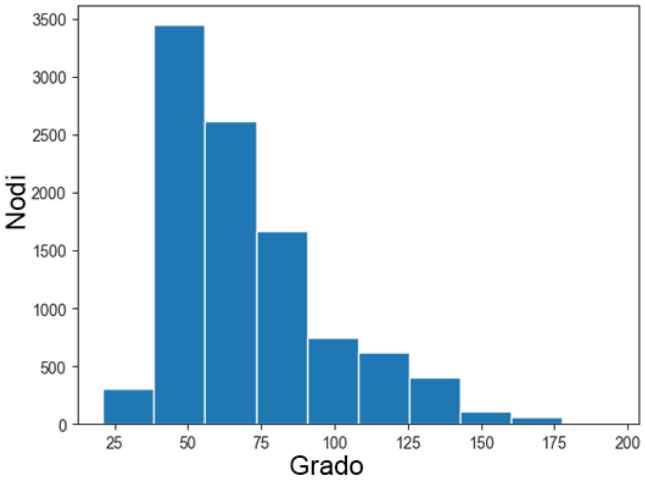


Figura 7: Distribuzione grado entrante: 10.000 nodi, KBucketList di dimensione 20. Gestione random della KBucket List

Nelle figure 8a e 8b abbiamo invece l'analisi del grado uscente che comunque non si differenzia troppo rispetto ai risultati mostrati nei paragrafi precedenti.



(a) Distribuzione del grado uscente



(b) Istogramma

Figura 8: Grado uscente. 10000 Nodi, KbucketList di dimensione 20. Gestione Random della KBucket List

5.4 Analisi sugli snapshot

È stata effettuata un'analisi anche su dati parziali suddividendo l'inserimento dei nodi all'interno della rete in vari blocchi. Dopo l'inserimento di un blocco di dati sono state effettuate le analisi sul grafo parziale prodotto. Per questo test sono stati inseriti all'interno della rete 10.000 nodi con una KBucket list di dimensione 20. L'esperimento è stato svolto due volte modificando il modo in cui vengono gestiti gli inserimenti nella KBucketList in caso di lista già piena. Nel primo caso è stato considerata una gestione della KBucket list in cui si preferiva mantenere i nodi vecchi già presenti mentre nel secondo caso sono stati inseriti i nodi nuovi eliminando i vecchi.

Nella figura 9 è possibile vedere la crescita del numero di archi presenti all'interno del grafo di step in step.

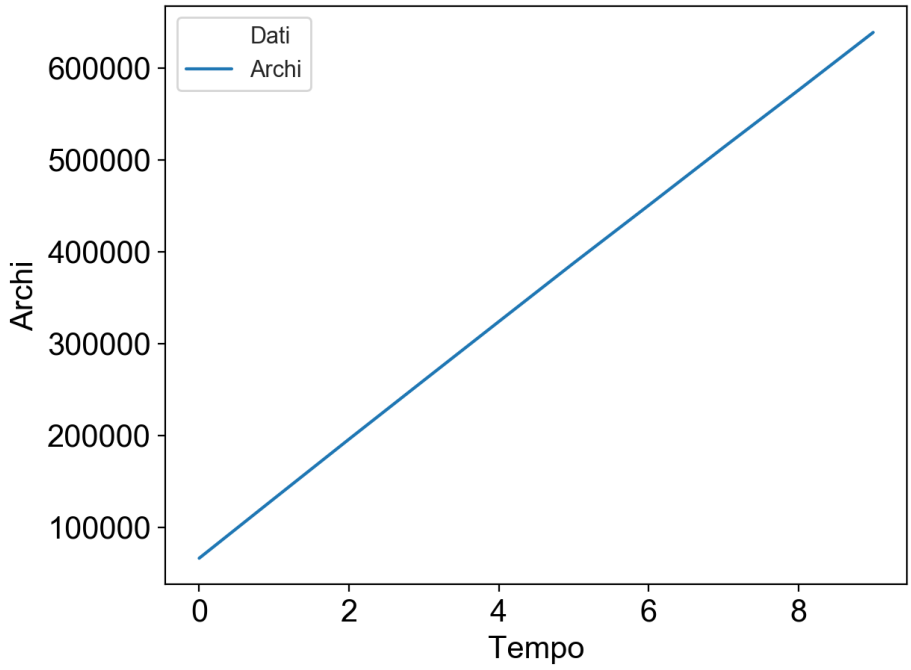
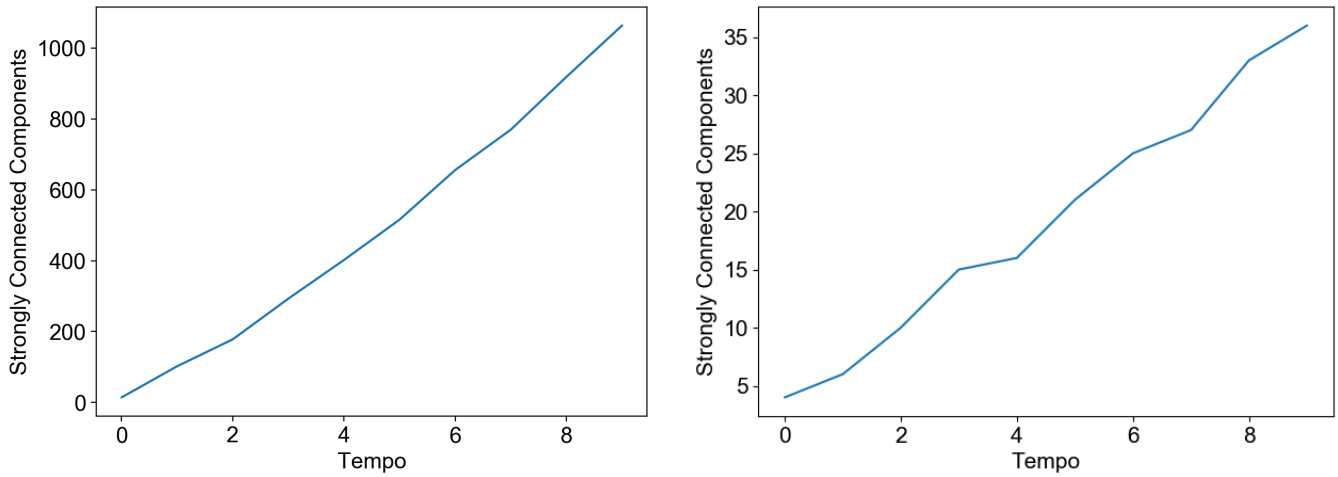


Figura 9: Crescita del numero di archi nel grafo

Nelle figure 10a e 10b possiamo vedere come varia il numero delle componenti connesse a mano a mano che vengono inseriti nuovi nodi nella rete. Una differenza che possiamo notare (e che è stata già sottolineata in precedenza) è che il numero di componenti connesse della simulazione in cui si gestisce la KBucket List scartando i nodi più vecchi risulta decisamente più basso rispetto alla simulazione in cui invece vengono scartati i più recenti.

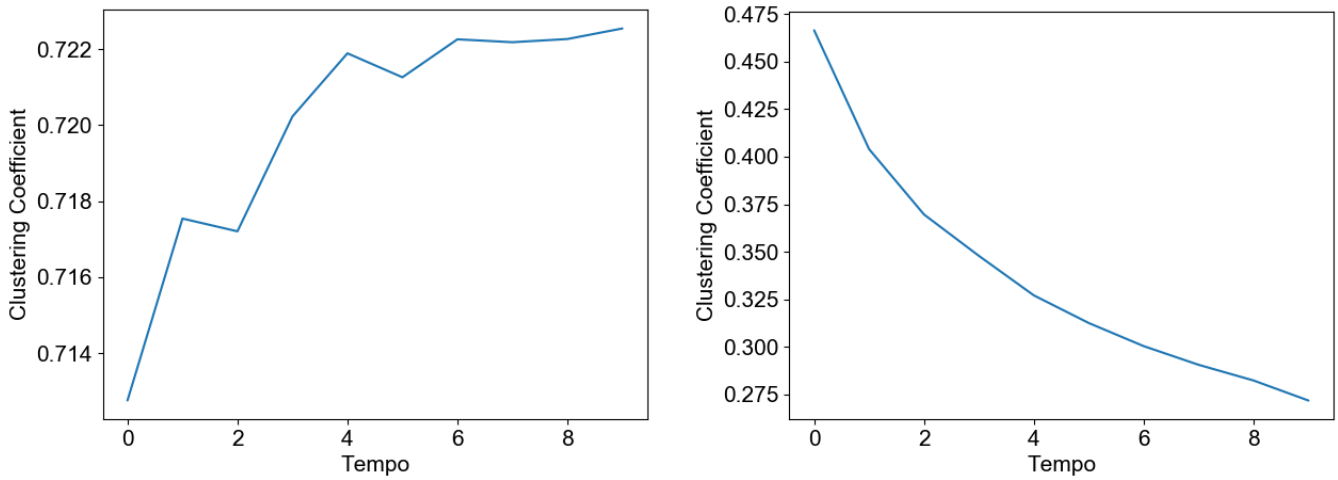


(a) Analisi su 10 snapshot con 10.000 nodi complessivi e gestione della KBucket List con nodi vecchi (in caso di riempimento)

(b) Analisi su 10 snapshot con 10.000 nodi complessivi e gestione della KBucket List con nodi nuovi (in caso di riempimento)

Figura 10: Crescita del numero di Strongly Connected Components

Anche la variazione del clustering coefficient è decisamente differente a seconda della gestione della KBucket List. Se gestiamo la KBucket list scartando i nodi nuovi nel caso in cui dovessimo avere già la lista piena avremo un valore medio del clustering coefficient che tende a salire nel momento in cui aumentano i nodi presenti all'interno del grafo. Questo comportamento lo possiamo vedere in figura 11a. Un comportamento del tutto differente lo abbiamo nella figura 11b, in questo caso la KBucket list viene gestita preferendo l'inserimento dei nodi nuovi e il valore medio del clustering cofficient tende a diminuire con l'aumentare del numero dei nodi presenti all'interno del grafo.



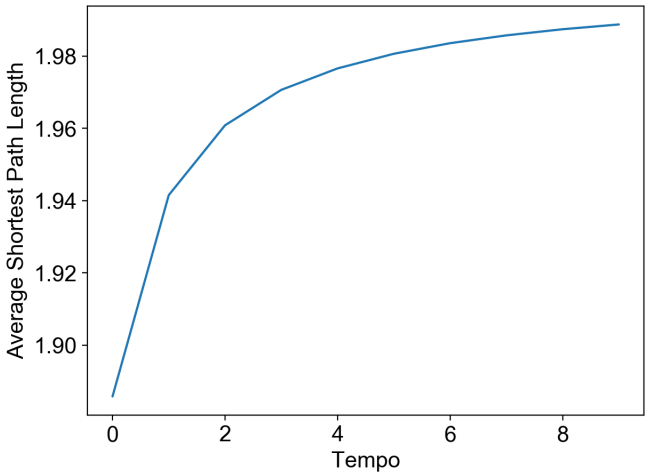
(a) Analisi su 10 snapshot con 10.000 nodi complessivi e gestione della KBucket List con nodi vecchi (in caso di riempimento)

(b) Analisi su 10 snapshot con 10.000 nodi complessivi e gestione della KBucket List con nodi nuovi (in caso di riempimento)

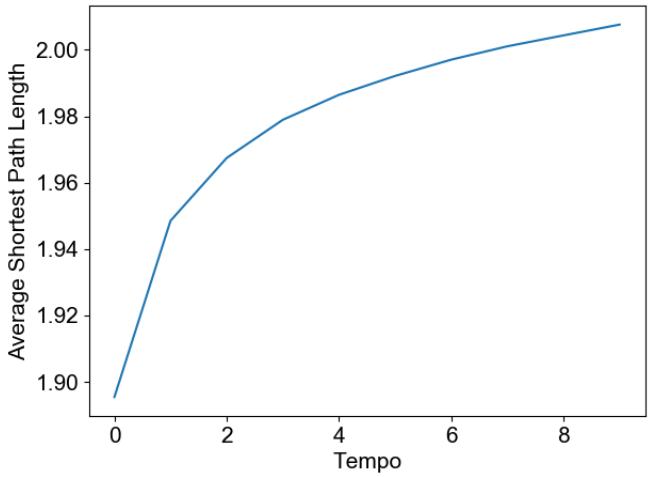
Figura 11: Variazione del Clustering Coefficient

Un'ultima analisi effettuata sugli snapshot è la variazione della lunghezza dello shortest path, in questo caso abbiamo un comportamento praticamente identico in entrambi i casi, come possiamo vedere nelle figure 12a e 12b.

Shortest Path Length-TemperaleNodiNuovi.png



(a) Analisi su 10 snapshot con 10.000 nodi complessivi e gestione della KBucket List con nodi vecchi (in caso di riempimento)



(b) Analisi su 10 snapshot con 10.000 nodi complessivi e gestione della KBucket List con nodi nuovi (in caso di riempimento)

Figura 12: Variazione dello shortest path length

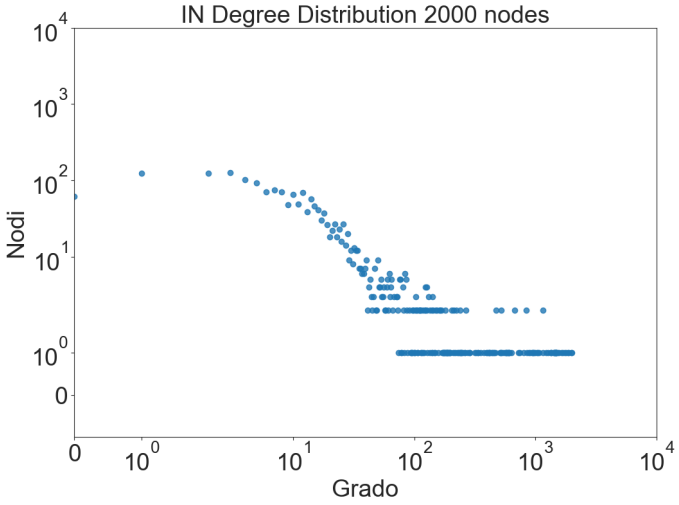
5.5 Confronto tra grafi

Nella Tabella 1 viene mostrato come cambiano i risultati dell'analisi al variare del numero di nodi presenti all'interno del grafo. In questo primo esperimento è stato considerato un grafo con delle Kbucket List di dimensione 20, nel caso in cui durante un inserimento una lista dovesse risultare piena, verranno mantenuti i nodi già presenti e scartato il nuovo.

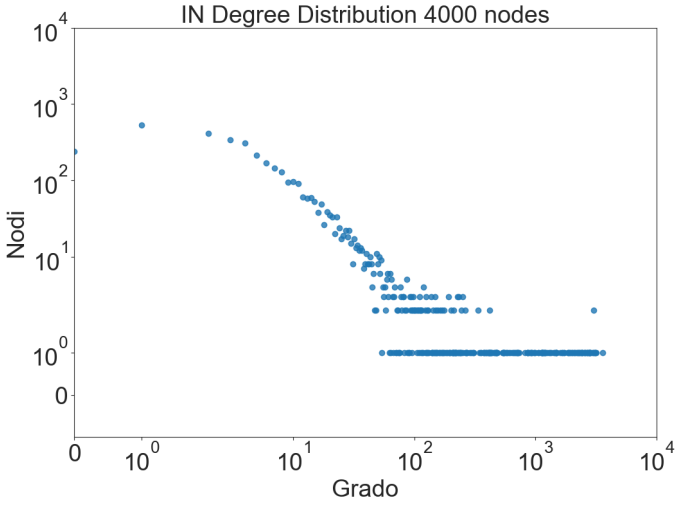
Nodi	Diametro	Archi	Connected Components	SCC	AVG Clustering Coefficient	AVG shortest Path Length
2000	2	123686	1	68	0.7019	1.9455
4000	3	262800	1	257	0.6883	1.9695
6000	3	416631	1	692	0.7051	1.9780
8000	3	482849	1	616	0.6677	1.9866
10000	3	657565	1	1200	0.6278	1.9874

Tabella 1: Risultati delle analisi al variare del numero di nodi presenti nel grafo. KBucket list con "nodi vecchi"

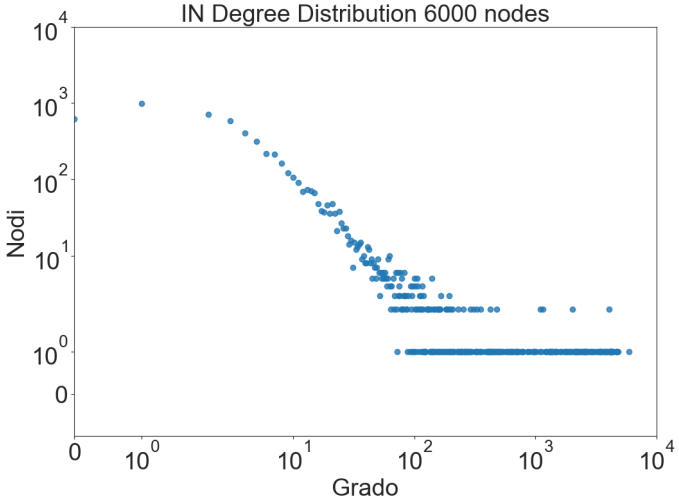
Dalla tabella 1 possiamo notare come il diametro del grafo rimanga sempre di dimensione molto bassa senza variare molto da un test all'altro. Nella figura 13 vengono confrontate le distribuzioni del grado entrante mentre nella Figura 14 le distribuzioni del grado uscente.



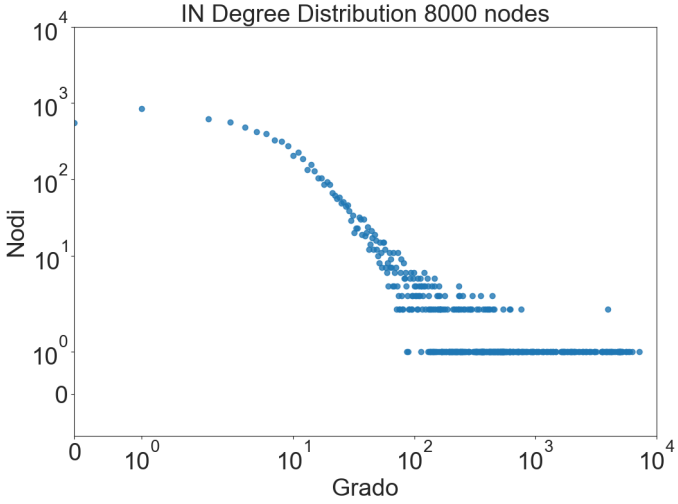
(a) Grafo con 2000 nodi



(b) Grafo con 4000 nodi

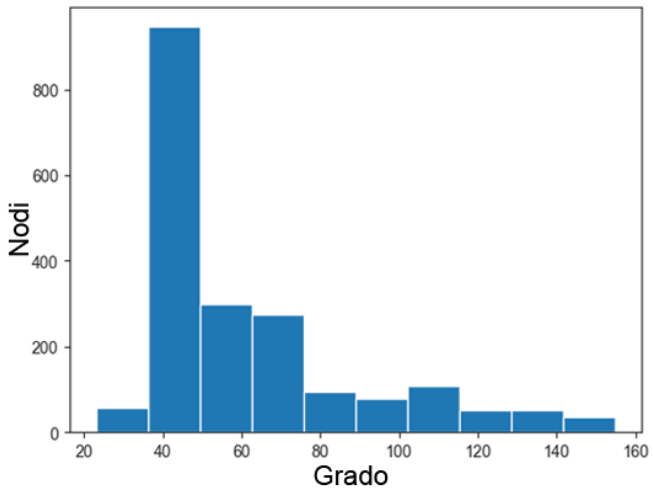


(c) Grafo con 6000 nodi

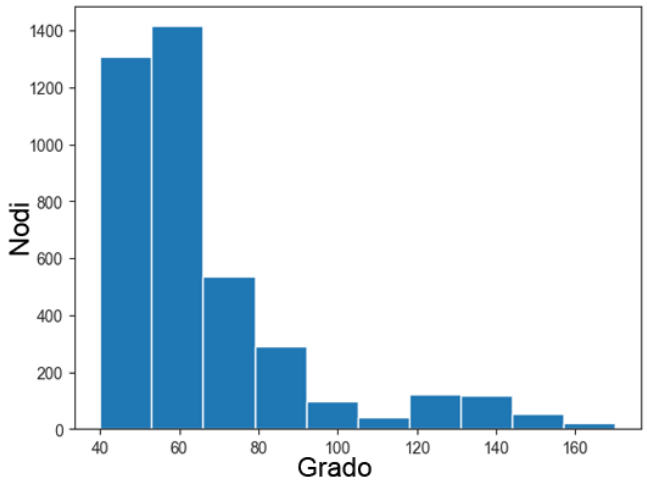


(d) Grafo con 8000 nodi

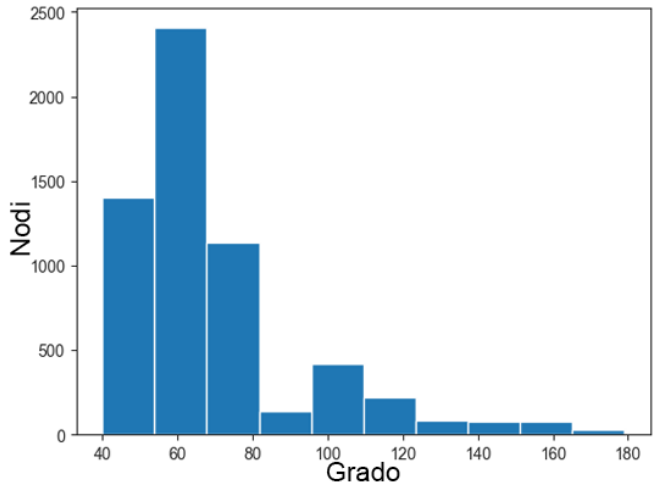
Figura 13: Distribuzione del grado entrante al variare del numero di nodi nella rete



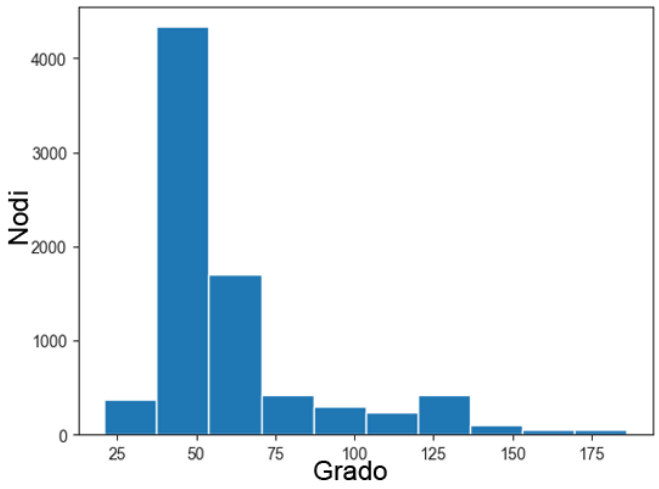
(a) Grafo con 2000 nodi



(b) Grafo con 4000 nodi



(c) Grafo con 6000 nodi



(d) Grafo con 8000 nodi

Figura 14: Distribuzione del grado uscente al variare del numero di nodi nella rete

Nella tabella 2 abbiamo ripetuto l'esperimento precedente modificando però la gestione delle KBucket List, in questo caso quando una lista è piena si scarta il nodo più vecchio per inserire il nuovo. Lo stesso esperimento è stato ripetuto anche con la gestione "random" della KBucket List e il risultato viene mostrato nella Tabella 3

Nodi	Diametro	Archi	Connected Components	SCC	AVG Clustering Coefficient	AVG shortest Path Length
2000	3	131829	1	6	0.4328	1.9436
4000	3	256775	1	23	0.3829	1.9742
6000	3	385421	1	22	0.2964	1.9856
8000	3	500095	1	22	0.2756	2.0039
10000	3	649218	1	42	0.2716	2.0216

Tabella 2: Risultati delle analisi al variare del numero di nodi presenti nel grafo. KBucket list con "nodi nuovi"

Nodi	Diametro	Archi	Connected Components	SCC	AVG Clustering Coefficient	AVG shortest Path Length
2000	3	140438	1	47	0.4847	1.9370
4000	3	276180	1	91	0.4181	1.9688
6000	3	376427	1	101	0.3198	1.9839
8000	3	575341	1	226	0.3641	1.9845
10000	3	694625	1	255	0.3352	1.9929

Tabella 3: Risultati delle analisi al variare del numero di nodi presenti nel grafo. KBucket list con "nodi random"

Lo stesso tipo di analisi è stata svolta anche modificando la dimensione della KBucket List. Anche in questo caso sono stati testati tutti e tre i metodi per la gestione delle KBucket List. Nella tabella 4 abbiamo la gestione con i nodi "nuovi", nella 5 con i nodi "vecchi" e nella Tabella 6 quella "random". In tutte le prove sono stati inseriti solamente 5000 nodi all'interno della rete.

KBucket List	Diametro	Archi	Connected Components	SCC	AVG Clustering Coefficient	AVG shortest Path Length
5	4	75550	1	22	0.3549	2.3313
10	4	161564	1	30	0.3098	2.0637
15	3	239083	1	19	0.2950	1.9964
20	3	344710	1	24	0.3525	1.9772

Tabella 4: Risultati delle analisi al variare della dimensione della KBucket List. Gestione della KBucket list con "nodi nuovi"

KBucket List	Diametro	Archi	Connected Components	SCC	AVG Clustering Coefficient	AVG shortest Path Length
5	4	74805	1	1013	0.7093	2.0592
10	3	158605	1	731	0.6640	1.9884
15	3	224982	1	499	0.6766	1.9835
20	3	337371	1	453	0.7124	1.9748

Tabella 5: Risultati delle analisi al variare della dimensione della KBucket List. Gestione della KBucket list con "nodi vecchi"

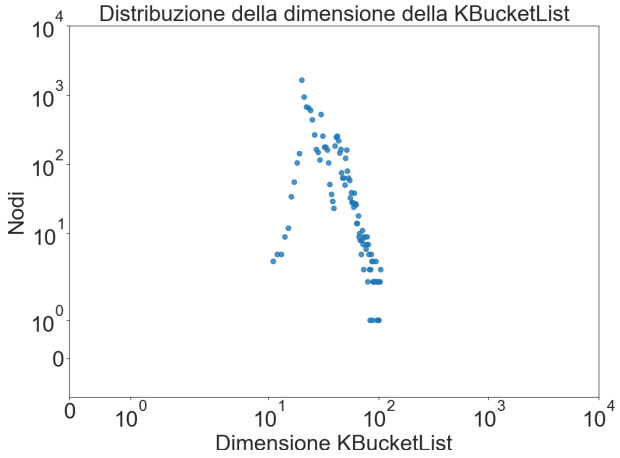
KBucket List	Diametro	Archi	Connected Components	SCC	AVG Clustering Coefficient	AVG shortest Path Length
5	4	77801	1	234	0.3495	2.2876
10	3	166388	1	173	0.3251	2.0412
15	3	256434	1	135	0.3532	1.9868
20	3	322924	1	100	0.3880	1.9777

Tabella 6: Risultati delle analisi al variare della dimensione della KBucket List. Gestione della KBucket list "random"

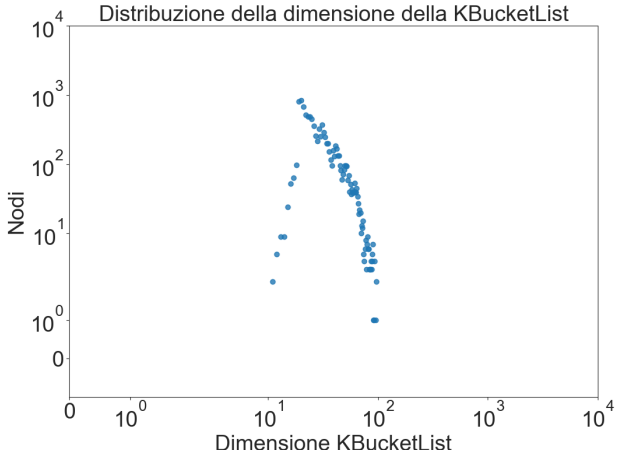
5.6 Studio delle Routing Table e sulle KBucket List

Una volta completate le analisi sul grafo è stato eseguito un ultimo test anche sulle routing table e sulle KBucket List per cercare di capire quanto queste venissero riempite dopo l'inserimento di tutti i nodi nella rete. Tutte le prove presenti in questa sezioni sono state svolte considerando l'inserimento di 10.000 nodi all'interno del grafo.

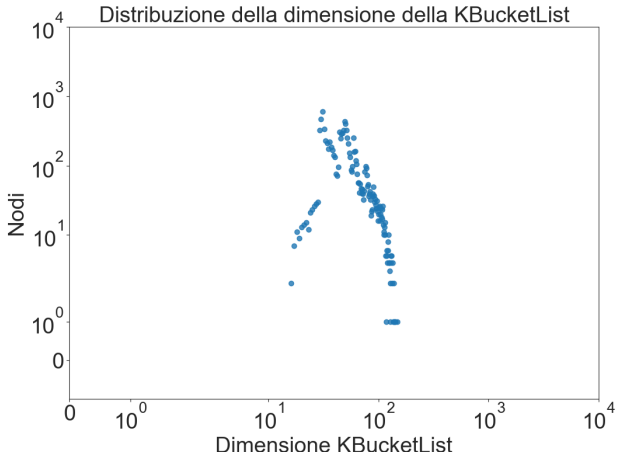
Nella figura 15 viene mostrata la distribuzione della quantità di nodi presenti all'interno delle routing table al variare della dimensione della KBucket List e del tipo di gestione della KBucket List.



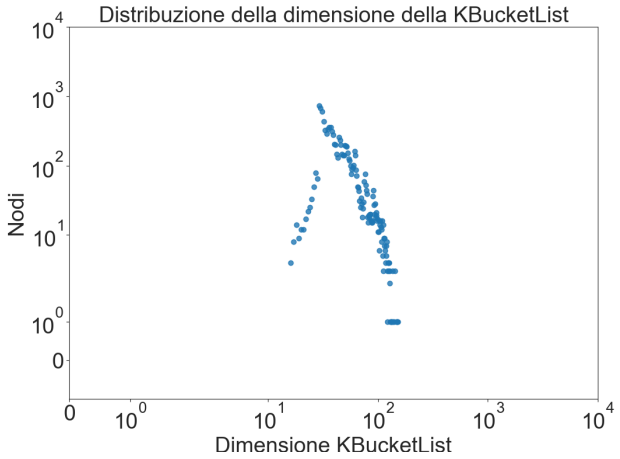
(a) Distribuzione della quantità di nodi nelle routing table. KBucketList lunga 10. KbucketList con Nodi vecchi



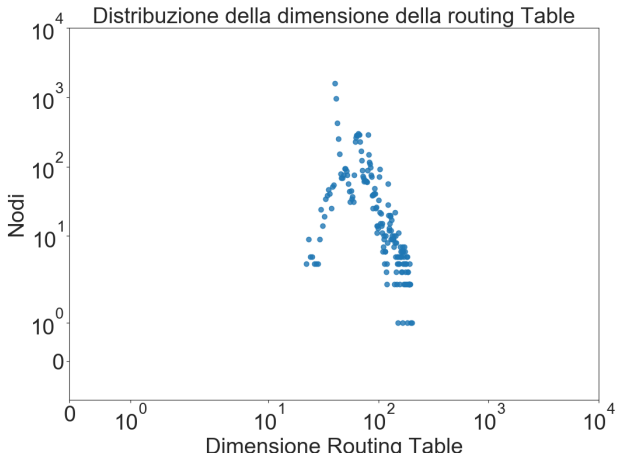
(b) Distribuzione della quantità di nodi nelle routing table. KBucketList lunga 10. KbucketList con Nodi nuovi



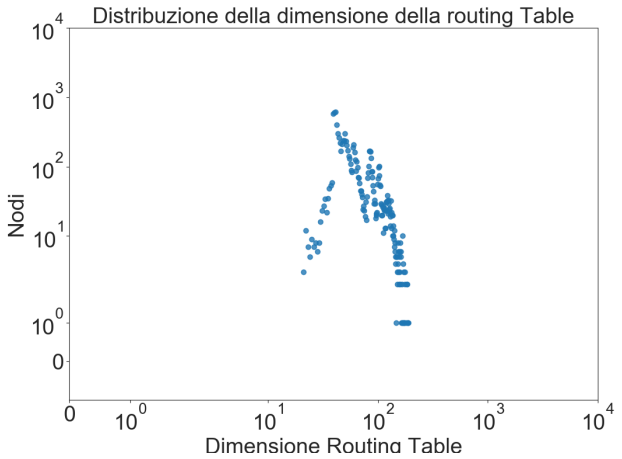
(c) Distribuzione della quantità di nodi nelle routing table. KBucketList lunga 15. KbucketList con Nodi vecchi



(d) Distribuzione della quantità di nodi nelle routing table. KBucketList lunga 15. KbucketList con Nodi nuovi



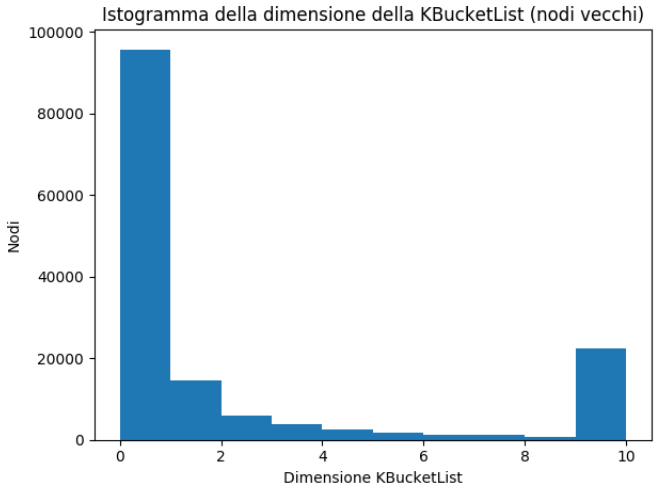
(e) Distribuzione della quantità di nodi nelle routing table. KBucketList lunga 20. KbucketList con Nodi vecchi



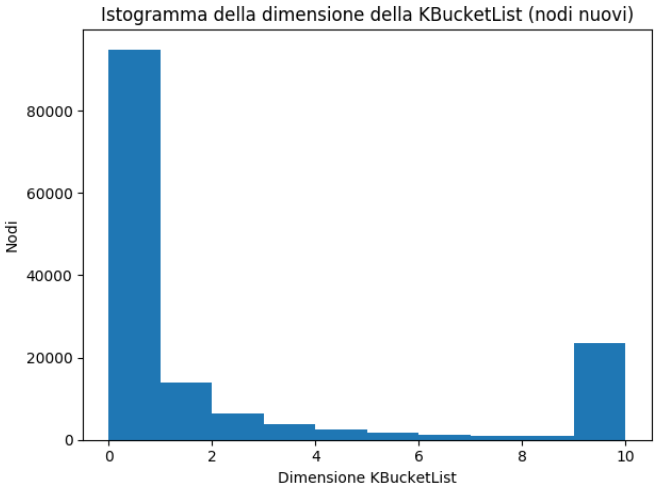
(f) Distribuzione della quantità di nodi nelle routing table. KBucketList lunga 20. KbucketList con Nodi nuovi

Figura 15: Distribuzione della quantità di nodi nelle Routing Table

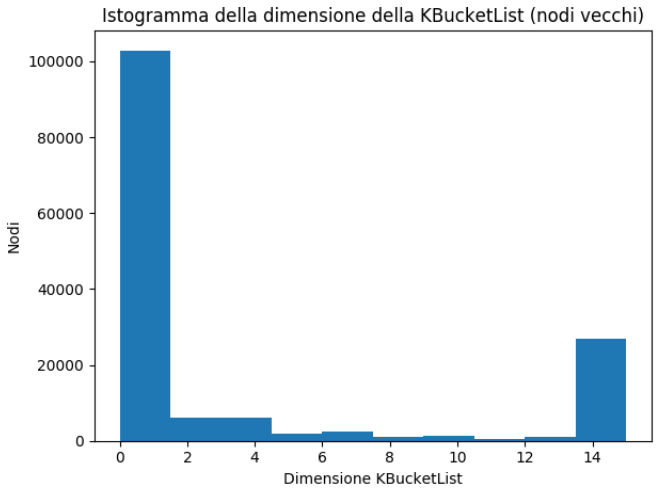
Nella figura 16 invece vediamo gli istogrammi che rappresentano la quantità di nodi presenti all'interno delle KBucket List al variare dei parametri.



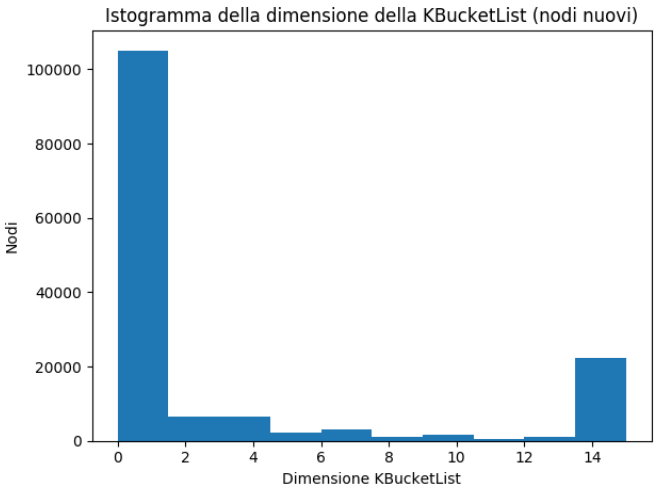
(a) Istogramma della quantità di nodi nelle KBucketList. KBucketList lunga 10. KbucketList con Nodi vecchi



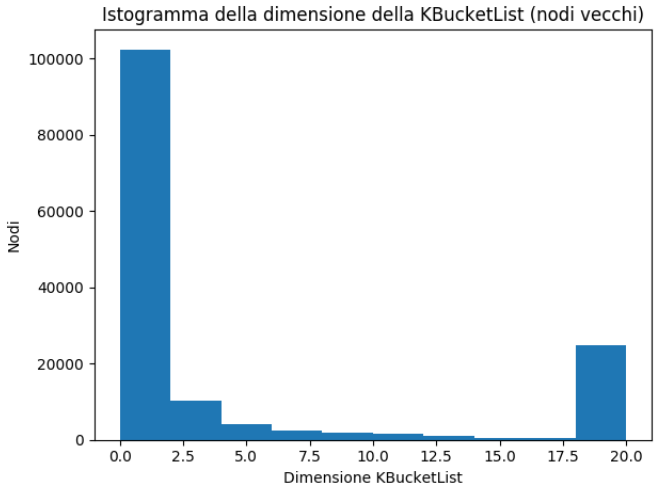
(b) Istogramma della quantità di nodi nelle KBucketList. KBucketList lunga 10. KbucketList con Nodi nuovi



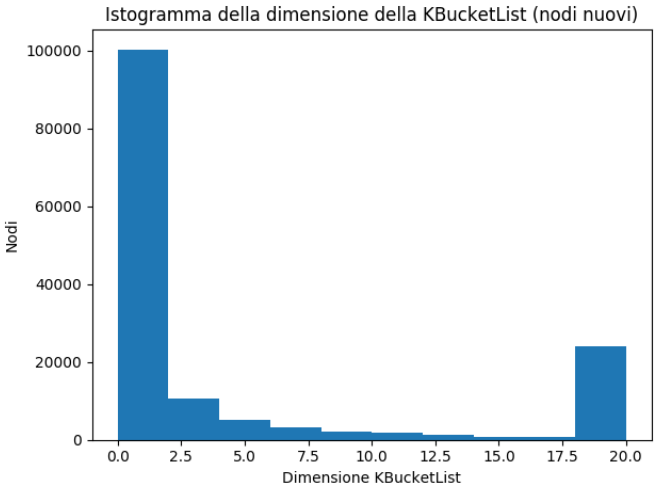
(c) Istogramma della quantità di nodi nelle KBucketList. KBucketList lunga 15. KbucketList con Nodi vecchi



(d) Istogramma della quantità di nodi nelle KBucketList. KBucketList lunga 15. KbucketList con Nodi nuovi



(e) Istogramma della quantità di nodi nelle KBucketList. KBucketList lunga 20. KbucketList con Nodi vecchi



(f) Istogramma della quantità di nodi nelle KBucketList. KBucketList lunga 20. KbucketList con Nodi nuovi

Figura 16: Distribuzione della quantità di nodi nelle KBucket List

6 Conclusioni

Tutti i grafi che sono stati analizzati hanno sempre avuto al loro interno al più 10.000 nodi, questa scelta è legata principalmente al tempo necessario per svolgere l'analisi. Per l'analisi del grafo con 10.000 nodi sono state necessarie circa 5 ore mentre per un'analisi sugli snapshot circa 10 (sul mio computer portatile). Inserendo un numero più alto di nodi non sarebbe stato possibile svolgere troppe analisi a causa del troppo tempo necessario, quindi ho preferito scegliere questo numero massimo di nodi svolgendo però più esperimenti.

Sarebbe stato interessante confrontare le analisi effettuate su questa simulazione con delle analisi svolte sulla reale rete di Kademlia in modo da comprendere se e in che modo l'implementazione proposta differisce da quella reale. Purtroppo però non ho trovato nessuno studio del grafo di Kademlia e quindi non ho potuto effettuare questo confronto.

Riferimenti bibliografici

[1] Pep 257 – docstring conventions. https://www.python.org/dev/peps/pep-0257/#id19.

[2] pdoc. https://pdoc3.github.io/pdoc/.

[3] Networkx. https://networkx.github.io.