ALMA MATER STUDIORUM - UNIVERSITA' DI BOLOGNA

SEDE DI CESENA

FACOLTA' DI SCIENZE MATEMATICHE, FISICHE E NATURALI CORSO DI LAUREA IN SCIENZE E TECNOLOGIE INFORMATICHE

ALLA RICERCA DI GRAFI DI GRANDI DIMENSIONI CON DIAMETRO E GRADO MASSIMO FISSATI.

Relazione finale in

Algoritmi e strutture dati

Relatore

Chiar.mo Prof. Luciano Margara

Presentata da

Alex Vecchiettini

Sessione III

Anno Accademico 2011/2012

Sommario

1. Introduzione

La teoria dei grafi è un campo di studio relativamente recente che trova applicazione in moltissimi settori; per questo motivo vengono utilizzati approcci diversi ai problemi che essa presenta. Un grafo è una struttura che permette di schematizzare una vasta serie di condizioni, dalla rete di nodi in comunicazione fra loro qual è internet, alle mappe dei singoli siti fino alle mappe dei navigatori satellitari.

Questa semplice struttura è composta da:

- *nodi* (o vertici) strutture semplici;
- *collegamenti* fra i vari nodi che, a seconda della tipologia di grafo possono essere detti:
    - *archi* nel caso in cui si tratti di un grafo orientato, tali collegamenti sono unidirezionali;
    - *spigoli* nel caso in cui il grafo in questione sia non orientato, in questo caso i collegamenti risultano essere bidirezionali.

È, inoltre, possibile classificare i grafi secondo il valore che ciascun collegamento, sia esso un arco o uno spigolo, assume; si possono quindi distinguere:

- i *grafi pesati* in cui ciascun collegamento ha un suo valore specifico, detto peso, che indica il costo per necessario a percorrere tale collegamento;
- i *grafi non pesati* in cui tutti i collegamenti assumono lo stesso valore, si può quindi affermare che i grafi non pesati altro non sono che un caso particolare di grafi pesati.

Tra gli innumerevoli problemi tuttora aperti in questo ambito (ricordiamo fra gli altri “Il problema del commesso viaggiatore” o la ricerca della Clique massima, problemi appartenenti alla famiglia NP-Complete) ci siamo imbattuti, nel contesto di uno scambio culturale, nel cosi detto “Degree-diameter problem”, oggetto di questa tesi.

Come si evince dal nome stesso questo problema concerne, in particolare, due proprietà di un grafo:

- il diametro, ovvero la massima distanza fra due nodi del grafo, essendo la distanza la lunghezza del cammino più breve fra i due nodi;
- il grado, in questo caso inteso come grado massimo, che indica il maggiore fra i gradi dei nodi del grafo (si ricorda che il grado di un nodo equivale al numero di archi entranti ed uscenti dallo stesso).

1.1 Tecnologie utilizzate

Per lo studio di queste strutture si è utilizzato un ambiente di calcolo particolarmente potente ed intuitivo: si tratta di Mathematica 8 sviluppato dalla Wolfram Research. Tale ambiente è dotato di un semplice linguaggio proprietario e comprende una serie di librerie e funzioni di facile utilizzo e particolarmente ottimizzate. Lo sviluppo di funzioni inerenti ai grafi è stato quasi immediato; in tale ambiente, dopo un primissimo e necessario momento di comprensione del linguaggio, è stato possibile dedicarsi interamente alla creazione, osservazione e valutazione di quanto pensato e scoperto senza perdersi in dettagli implementativi.

2. Il Problema

Il "Degree-diameter problem" consta nel trovare, costruire o ottenere grafi di grandi dimensioni sottostando ai vincoli di grado massimo e diametro. Al variare dei valori di queste due proprietà si avranno, pertanto, diversi grafi possibili e all'aumentare di questi corrisponde, chiaramente, un aumento della dimensione massima dei grafi ottenibili.

2.1 Il limite di Moore

Esiste un limite superiore alla dimensione del grafo; tale vincolo è denominato "limite di Moore" (o Moore bound) e determina, al variare dei valori massimi di grado e diametro, il numero di nodi che è possibile ottenere. Come detto, però, è un limite *superiore* e non garantisce, quindi, l'esistenza di grafi di tali dimensioni. Il limite di Moore è cosi definito:

$$1 + d\sum_{i=0}^{k-1}(d-1)^i$$

Dove *d* rappresenta il grado e *k* il diametro, dalla precedente si ottiene:

$$1 + d + d(d-1) + \cdots + d(d-1)^{k-1} = \frac{d(d-1)^k - 2}{d-2}$$

La comprensione di questo vincolo è abbastanza immediata; si può infatti supporre di partire da un nodo (rappresentato dall'unita che si aggiunge alla sommatoria) che avrà un massimo di *d* nodi adiacenti, ciascuno dei quali avrà a sua volta *d-1* nodi adiacenti (essendo il *d*-esimo nodo adiacente il nodo di "provenienza"), perciò al passo successivo vi saranno *d(d-1)* nodi ad avere *d-1* nodi adiacenti ed avremo quindi $d(d-1)^2$ nuovi nodi. È possibile protrarre tale procedimento per *k-1* iterazioni, delineando cosi la struttura di un albero e, alla (*k-1)*-esima iterazione, le foglie avranno una distanza *k* dalla radice. È evidente, però, che tale vincolo è un limite superiore e, per quanto concerne il "Degree-diameter problem", diversi grafi ottimi risultano ampiamente al di sotto di tale limite!

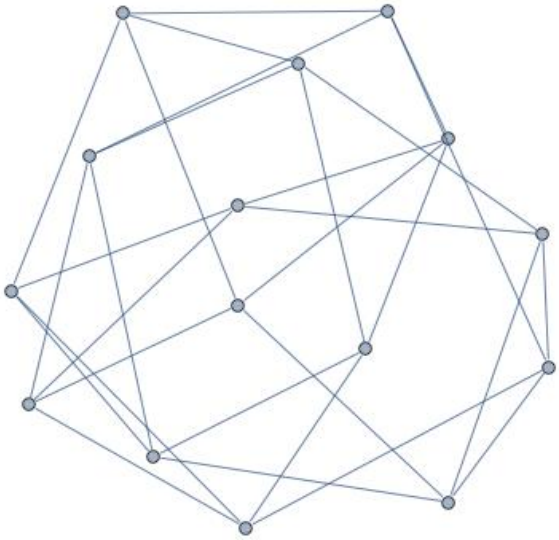


Figura 1 K3*C5
Questo Grafo è ottimo per d=4 e k=2
Moore bound = 17 Dimensione = 15

2.2 Tabella dei record

Al fine di registrare e condividere i risultati inerenti a questo problema è stata approntata una tabella (raggiungibile al sito http://www-mat.upc.es/grup_de_grafs/table_g.html/) che ha per righe i valori del grado massimo mentre, nelle colonne, si hanno i valori del diametro.

d \ k	2	3	4	5	6	7	8	9	10
3	10	20	38	70	132	196	336	600	1 250
4	15	41	98	364	740	1 320	3 243	7 575	17 703
5	24	72	212	624	2 772	5 516	17 030	53 352	164 720
6	32	111	390	1 404	7 917	19 282	75 157	331 387	1 212 117
7	50	168	672	2 756	11 988	52 768	233 700	1 124 990	5 311 572
8	57	253	1 100	5 060	39 672	130 017	714 010	4 039 704	17 823 532
9	74	585	1 550	8 268	75 893	270 192	1 485 498	10 423 212	31 466 244
10	91	650	2 223	13 140	134 690	561 957	4 019 736	17 304 400	104 058 822
11	104	715	3 200	18 700	156 864	971 028	5 941 864	62 932 488	250 108 668
12	133	786	4 680	29 470	359 772	1 900 464	10 423 212	104 058 822	600 105 100
13	162	851	6 560	39 576	531 440	2 901 404	17 823 532	180 002 472	1 050 104 118
14	183	916	8 200	56 790	816 294	6 200 460	41 894 424	450 103 771	2 050 103 984
15	187	1 215	11 712	74 298	1 417 248	8 079 298	90 001 236	900 207 542	4 149 702 144
16	198	1 600	14 640	132 496	1 771 560	14 882 658	104 518 518	1 400 103 920	7 394 669 856

Tabella 1 Tabella dei grafi record
I diversi colori dello sfondo rappresentano i diversi autori, metodi di costruzione o testi di riferimento.

Come già puntualizzato, nonostante i grafi riportati siano i migliori risultati finora ottenuti, si osserva una sensibile distanza fra l'ordine di tali grafi e i valori corrispettivi del Moore bound.

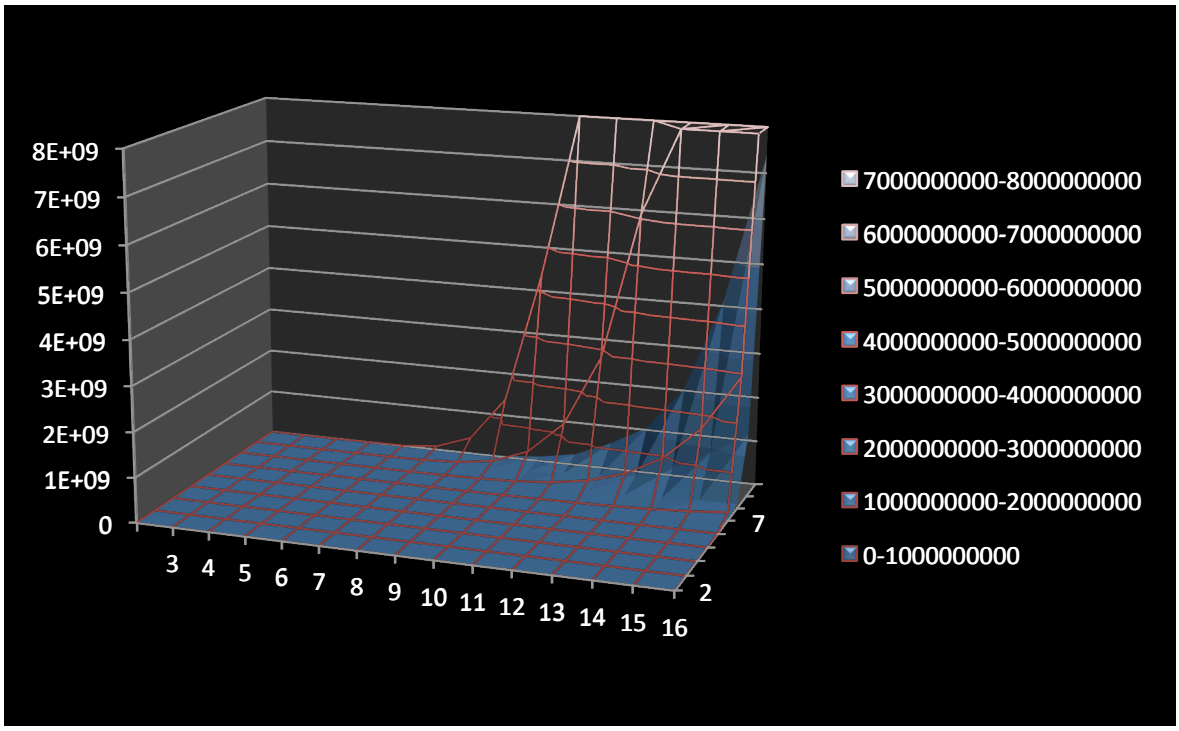


Grafico 1 Valori Grafi record / Moore Bound

L'area blu rappresenta i valori dell'ordine dei grafi record mentre il reticolato rosso indica il Moore bound, al variare di grado e diametro.

2.3 Valutazione della bontà di un grafo

Risulta evidente la necessità, al fine di comprendere la qualità dei grafi in analisi, di stabilire un criterio di valutazione, a tale scopo si utilizzerà il rapporto fra l'ordine del grafo ed il Moore bound, definendolo come:

$$\alpha(G) = |G| \frac{d(G) - 2}{d(G)(d(G) - 1)^{k(G)} - 2}$$

α(G)	2	3	4	5	6	7	8	9	10
3	1,00	0,91	0,83	0,74	0,69	0,51	0,44	0,39	0,41
4	0,88	0,77	0,61	0,75	0,51	0,30	0,25	0,19	0,15
5	0,92	0,68	0,50	0,37	0,41	0,20	0,16	0,12	0,09
6	0,86	0,59	0,42	0,30	0,34	0,16	0,13	0,11	0,08
7	1,00	0,56	0,37	0,25	0,18	0,13	0,10	0,08	0,06
8	0,88	0,55	0,34	0,23	0,25	0,12	0,09	0,08	0,05
9	0,90	0,89	0,29	0,20	0,23	0,10	0,07	0,06	0,02
10	0,90	0,71	0,27	0,18	0,20	0,09	0,07	0,04	0,02
11	0,85	0,59	0,26	0,15	0,13	0,08	0,05	0,05	0,02
12	0,92	0,49	0,27	0,15	0,17	0,08	0,04	0,04	0,02
13	0,95	0,42	0,27	0,13	0,15	0,07	0,04	0,03	0,01
14	0,93	0,36	0,25	0,13	0,14	0,08	0,04	0,04	0,01
15	0,83	0,38	0,26	0,12	0,16	0,07	0,05	0,04	0,01
16	0,77	0,41	0,25	0,15	0,14	0,08	0,04	0,03	0,01

Tabella 2 Valori $\alpha(G)$ applicati ai grafi record

Dalla precedente tabella si osserva come all' aumentare dei valori di grado e diametro la distanza fra i grafi record e il valore del vincolo di Moore aumenti, pertanto il valore derivante dal rapporto sopra descritto come $\alpha(G)$ sarà da considerare relativamente all'ordine ed al grado di tale grafo e non in senso assoluto.

2.4 Approcci di risoluzione

Al fine di poter indagare il problema nei suoi vari aspetti, si sono utilizzati e studiati differenti metodi atti a scoprire grafi che massimizzino l'ordine minimizzando il grado massimo e il diametro.

2.4.1 Composizione di grafi

A partire dall'analisi delle famiglie di grafi più comuni, proseguendo con lo studio delle operazioni di prodotto fra grafi, si è proceduto all'analisi delle composizioni fra le differenti famiglie al fine di ottenerne un grafo risultante che ne combinasse le caratteristiche interessanti ai fini del problema. Per ottenere l'osservazione empirica dei risultati si è proceduto all'implementazione delle diverse tipologie di prodotto, dalle più banali, come il prodotto cartesiano, al prodotto diretto e a sue più raffinate varianti, già definite in letteratura, che

hanno portato alla scoperta di alcuni grafi record come il già citato K3*C5 (record per diametro 2 e grado massimo 4 di ordine 15) o K3*X8 (record per diametro 2 e grado massimo 5 di ordine 24).

2.4.2 Elaborazione di euristiche

In seguito allo studio delle famiglie di grafi si è proseguita e approfondita l'analisi dei grafi random e di una classe di grafi di recente concezione, i così detti grafi "Expanders" che hanno mostrato risultati comparabili, se non migliori, di quelli ottenuti da diverse famiglie tradizionali di grafi. Lo sviluppo delle euristiche ha quindi preso due direzioni:

- la perturbazione di grafi generati casualmente al fine di minimizzarne grado massimo e diametro e, ove possibile, aumentarne l'ordine;
- la creazione di "Expanders", tramite diversi metodi studiati in documenti di letteratura inerenti a questi processi, come, ad esempio, la composizione tramite il "zig-zag product" fra grafi regolari.

3. Composizione di grafi

Come base per la composizione di grafi si sono studiate alcune delle famiglie di grafi più note considerandone i valori del grado massimo e del diametro al variare dell'ordine. Si è, quindi, proceduto all'analisi dei prodotti fra grafi e delle proprietà delle composizioni da essi ottenute in relazione ai grafi dati in input.

3.1 Studio delle famiglie

Si è scelto, per motivi di chiarezza espositiva, di rappresentare i valori di grado massimo e diametro in funzione dell'ordine del grafo, come funzioni continue, anche se, ovviamente, si tratta di funzioni discrete.

S'intenderà, con grafo x-regolare un grafo con distribuzione uniforme degli archi sui nodi, ovvero, con nodi aventi tutti lo stesso grado pari ad x.

3.1.1 Grafi completi

I grafi completi risultano di semplice analisi, sono definiti come grafi completamente connessi, cioè, grafi in cui ogni nodo è collegato a tutti gli altri nodi del grafo, risulta per tanto evidente che un grafo completo di ordine N avrà un grado massimo *d* (uguale per tutti i nodi) di N-1 ed un diametro *k* pari a 1. Questi grafi sono, dunque, (N-1)-regolari e hanno come insieme di spigoli l'insieme di tutti gli spigoli possibili, questo insieme ha cardinalità pari a $\frac{N(N-1)}{2}$, cioè $\binom{N}{2}$, l'insieme di tutte le coppie di nodi. Si è soliti indicare questi grafi con KN dove N rappresenta l'ordine del grafo in questione.

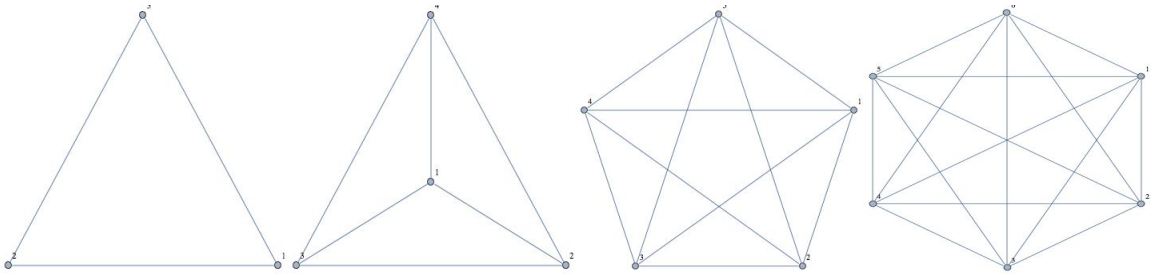


Figura 2 Grafi completi

Da sinistra verso destra K3, K4, K5 e K6

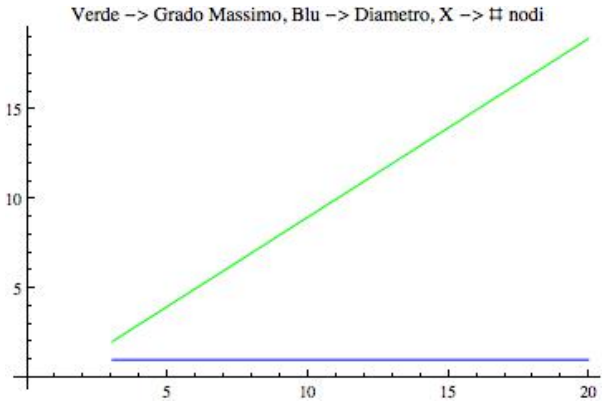


Grafico 2 Grado-Diametro grafi completi

Il grafico indica i valori di grado massimo e diametro per un campione di grafi completi di ordine compreso in [3, 20]

3.1.2 Grafi ciclici

Un'altra classe di immediata interpretazione è quella dei grafi ciclici, questi sono definiti con CN in cui N indica l'ordine del grafo e sono costituiti da N nodi (su cui per semplicità di spiegazione si definisce un ordine) collegati unicamente al nodo che li precede ed al nodo che li segue ed in cui l'ultimo nodo è collegato al primo. Questa classe gode dunque della proprietà di essere 2-regolare e, pertanto con grado massimo pari a 2. Il diametro, invece, risulta pari a $\left\lfloor \frac{N}{2} \right\rfloor$.

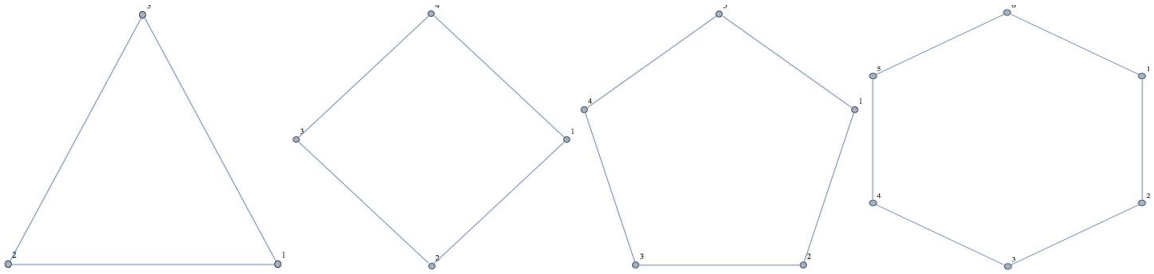


Figura 2 Grafi ciclici

Da sinistra verso destra C3, C4, C5 e C6

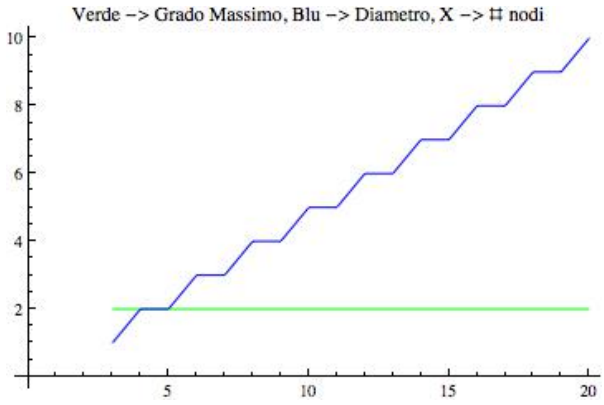


Grafico 3 Grado-Diametro grafi ciclici

Il grafico indica i valori di grado massimo e diametro per un campione di grafi ciclici di ordine compreso in [3, 20]

3.1.3 Grafi stella

I grafi stella presentano una struttura particolare, nella quale un unico nodo centrale è collegato a tutti gli altri nodi esterni, risulta quindi, per N maggiori di 2, un grafo con diametro 2 e grado massimo N-1. Appare chiaro che questa tipologia di grafi può essere migliorata ai fini del problema in quanto il grado massimo pari ad N-1 è presente nel solo nodo centrale mentre tutti i nodi esterni hanno grado 1. I grafi appartenenti a questa classe sono solitamente denotati come SN.

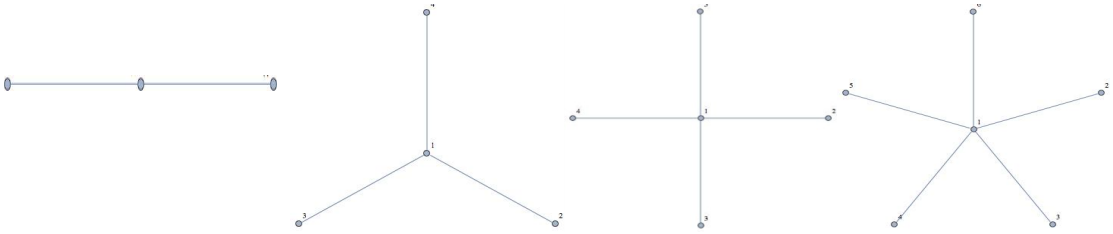


Figura 3 Grafi stella

Da sinistra verso destra S3, S4, S5 e S6

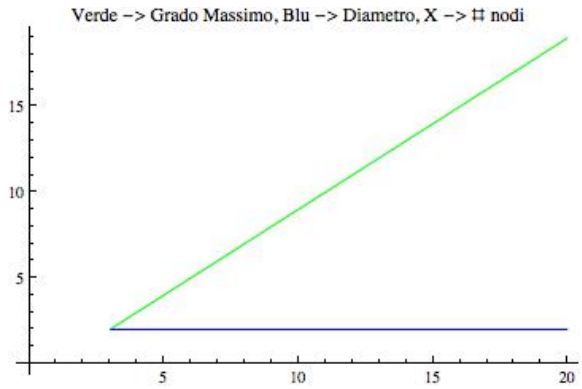


Grafico 4 Grado-Diametro grafi stella

Il grafico indica i valori di grado massimo e diametro per un campione di grafi stella di ordine compreso in [3, 20]

3.1.4 Grafi ruota

Nascono dall'unione degli insiemi dei nodi e degli spigoli di un grafo ciclico (C(N-1)) e di un grafo stella (S(N)), hanno pertanto una serie di nodi disposti esternamente e collegati fra loro da spigoli con il nodo precedente ed il successivo, possiedono, inoltre, un nodo centrale collegato a tutti i nodi disposti sulla circonferenza. Da tale disposizione si evince un grado massimo *d* ed un diametro *k* pari a quelli di un grafo stella, ovvero, rispettivamente N-1 e 2. Si osserva che questa tipologia di grafo non sfrutta le possibilità di connessione dei nodi esterni del grafo stella, da cui è derivato, in direzione di un miglioramento del grafo stesso nelle caratteristiche interessanti per il problema.

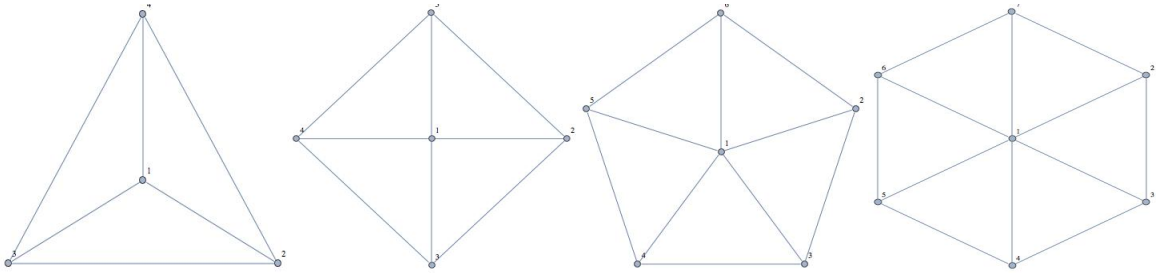


Figura 4 Grafi ruota

Da sinistra verso destra W4, W5, W6 e W7

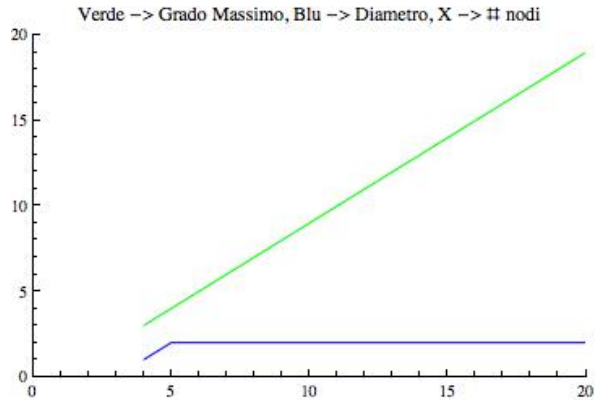


Grafico 5 Grado-Diametro grafi ruota

Il grafico indica i valori di grado massimo e diametro per un campione di grafi stella di ordine compreso in [4, 20

3.1.5 Grafi farfalla

Una definizione matematica per questa tipologia di grafi è la seguente: dati *n* e *b* due valori interi, si può definire un "butterfly graph" come un grafo avente $(n+1)b^n$ nodi, i quali, sono coppie $\{w, i\}$ con *w* stringa in base *b* di lunghezza *n,* e *i* un intero compreso in [*0, n*]. Per ogni *i*, vi è uno spigolo da $\{w, i\}$ a $\{w', i+1\}$ se *w* e w' sono congruenti in tutti i loro valori o, al più, differiscono per l'(*i +1*)-esimo valore. Questo caso è più interessante poiché, a differenza delle famiglie viste fin ora, non vi è una proporzionalità diretta fra diametro o grado massimo ed ordine del grafo ma, bensì, fatta eccezione per il caso *n=1* (per il quale *d=2 e k=b*), si ha *k=2n e d=2b* mentre l'ordine cresce, come già detto, esponenzialmente con *n* secondo $(n+1)b^n$. Si osserva, comunque, che definendo *G(n,b)* come un grafo farfalla con *n* e *b,* rispettivamente, lunghezza della stringa e base, si ha:

$$\alpha\big(G(n,b)\big) = \frac{(n+1)b^n * (2b-2)}{2b(2b-1)^{2n}-2} \approx \frac{nb^n * (2b)}{(2b)^{(2n)+1}} \approx \frac{nb^n}{(2b)^{2n}} \approx \frac{n}{2^{2n} * b^n}$$

il quale, purtroppo, è lontano dall'essere un buon risultato.

Un'ulteriore caratteristica di questa famiglia di grafi è che risultano bipartiti, ovvero, è possibile dividere i nodi dell'arco in due insiemi avendo tutti gli spigoli che collegano nodi appartenenti a due insiemi diversi.

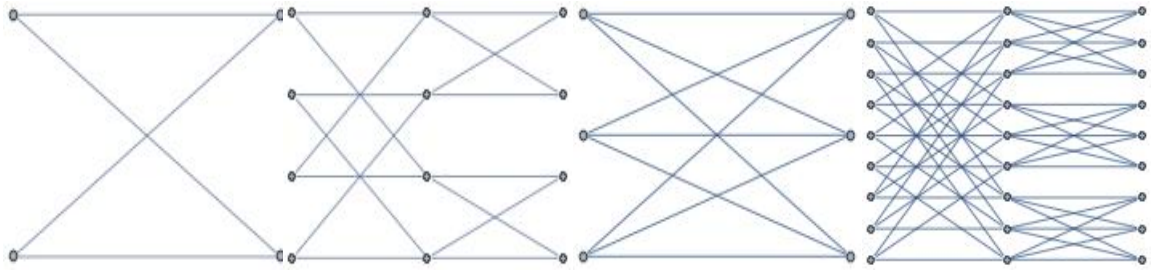


Figura 5 Grafi farfalla

Da sinistra verso destra grafi farfalla con (n, b): (1, 2), (2, 2), (1, 3) e (2, 3)

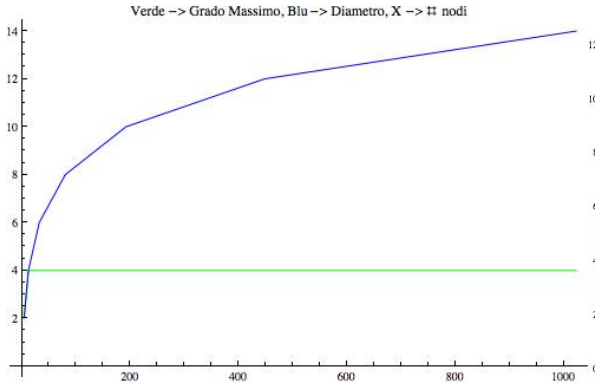


Grafico 6 Grado-Diametro grafi farfalla con base 2

Il grafico indica i valori di grado massimo e diametro per un campione di grafi farfalla con base 2 ed n compreso in [1, 7], si può notare che l'ordine dei grafi è compreso in [4, 1024].

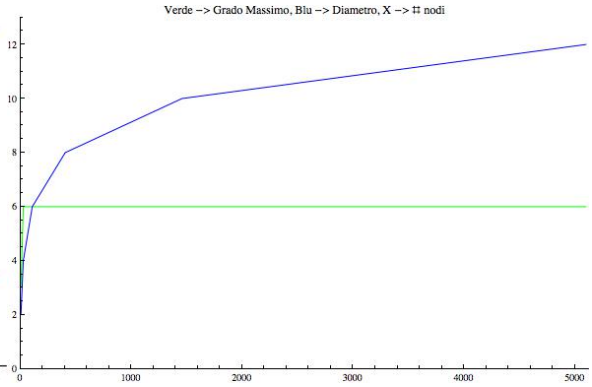


Grafico 7 Grado-Diametro grafi farfalla con base 3

Il grafico indica i valori di grado massimo e diametro per un campione di grafi farfalla con base 3 ed n compreso in [1, 6], si può notare che l'ordine dei grafi è compreso in [6, 5013].

3.1.6 Grafi di DeBruijn

Date una lunghezza *n* ed una base *b* il grafo di DeBruijn *G(b,n)* possiede b^n nodi, i quali sono etichettati con ogni possibile sequenza di lunghezza *n* dei *b* valori possibili; si ha quindi che due nodi *u* e *v* sono collegati da un arco in direzione (*v, u*) se e solo se *v* è ottenibile da *u* tramite uno spostamento a sinistra delle cifre e con l'inserimento di una nuova cifra al primo posto a destra. Al fine di conformare i grafi di DeBruijn al problema saranno apportate alcune modifiche a tali grafi, eliminando eventuali auto-archi e rendendo con spigoli indiretti gli archi diretti, cosi facendo si perderanno tre proprietà dei grafi di DeBruijn e, pertanto i grafi utilizzati non saranno più m-regolari, non saranno Euleriani (ovvero non tutti i nodi avranno grado pari e, pertanto, non vi saranno cicli Euleriani) e non avranno la proprietà per cui il "line graph" di *G(b,n)*

coincide con *G(b,n+1)*; rimarranno, invece, grafi Hamiltoniani (conterranno, cioè, cicli Hamiltoniani: percorsi ciclici che passano per ciascun nodo un'unica volta).

L'analisi di questi tipi di grafi ha evidenziato un elevato costo computazionale nella creazione degli stessi, infatti, i pochi dati presenti nei grafici sotto riportati sono dovuti ad un eccessiva quantità di tempo richiesta dal calcolatore nell'effettuare le operazioni richieste.

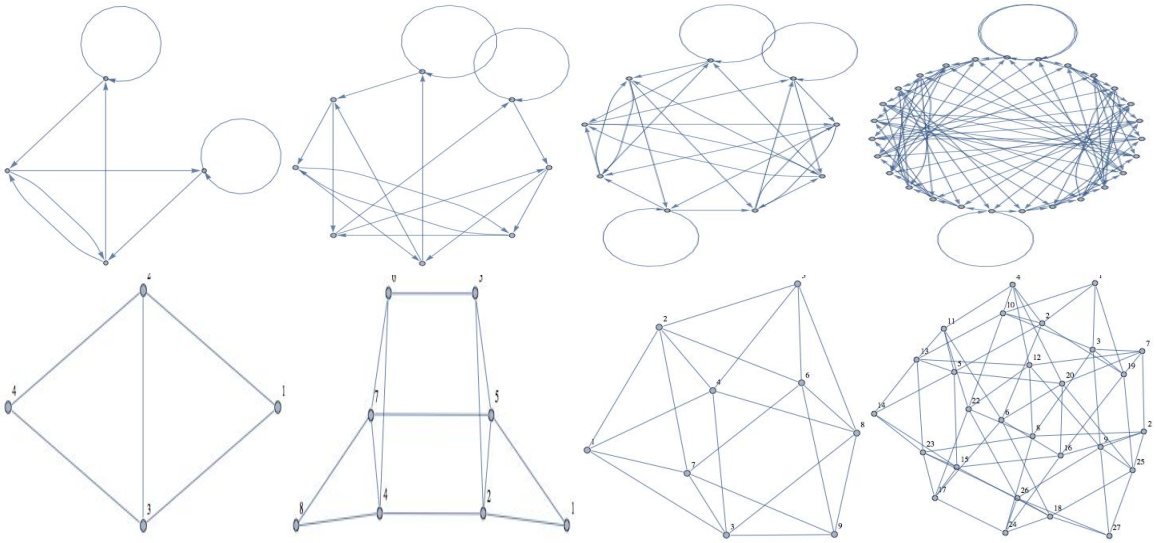


Figura 6 Grafi di DeBruijn

Da sinistra verso destra la prima riga riporta i grafi di DeBruijn con i valori (b, n): (2, 2), (2,3), (3, 2) e (3, 3). Da sinistra verso destra la seconda riga riporta i grafi di DeBruijn adattati con i valori (b, n): (2, 2), (2,3), (3, 2) e (3, 3).

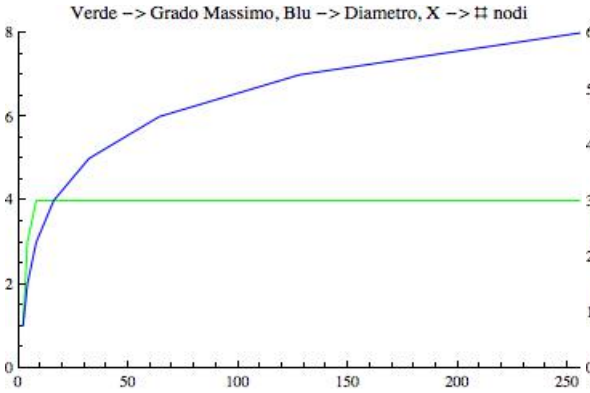


Grafico 7 Grado-Diametro Grafi DeBruijn modificati con *b=2*

Il grafico mostra i valori di grado e diametro al variare di n compreso in [1, 8] e con un ordine compreso in [2, 256]

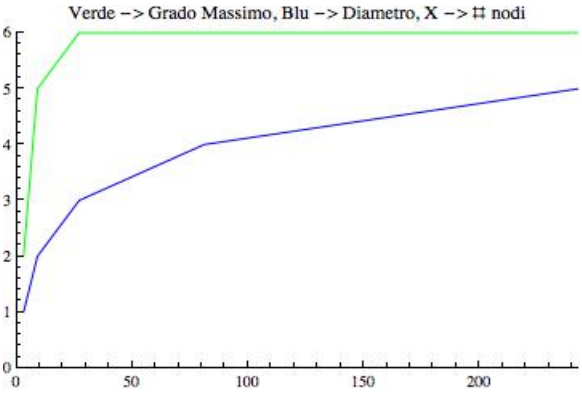


Grafico 8 Grado-Diametro Grafi DeBruijn modificati con *b=3*

Il grafico mostra i valori di grado e diametro al variare di n compreso in [1, 5] e con un ordine compreso in [3, 243]

3.2 Studio delle operazioni di prodotto fra grafi

Tutte le diverse operazioni di prodotto, se applicate sugli stessi grafi *A* e *B* con rispettivamente *n1* ed *n2* nodi producono il medesimo insieme di nodi, dato dal prodotto cartesiano degli insiemi di nodi di partenza, avranno, quindi, sempre ordine *n1 x n2* ; queste operazioni differiscono, invece, nelle condizioni alle quali uno spigolo apparterrà all'insieme degli spigoli del grafo risultante.

Dato un grafo *G* si indicherà con |*G*| l'ordine di tale grafo, con *V(G)* il suo insieme di nodi e con *E(G)* l'insieme dei suoi spigoli.

3.2.1 Prodotto cartesiano

Come già visto il prodotto cartesiano fra due grafi *A* □ *B* da come insieme risultante degli spigoli il prodotto cartesiano degli insiemi *V(A)* e *V(B)*, pertanto si potrà definire ogni vertice come una coppia ordinata (a, b) in cui $a \in V(A)$ e $b \in V(B)$; passando, invece, alla definizione dell'insieme di spigoli che apparterranno ad *A* □ *B*, si avrà che due vertici (a, b) e (a^I, b^I) saranno adiacenti se e solo se:

- $a = a^I$ e b è adiacente a b^I in B, o
- $b = b^I$ e a è adiacente a a^I in A.

Quest'operazione è commutativa e associativa, ovvero A □ B è isomorfo a B □ A e, pertanto, anche (A □ B) □ C risulta isomorfo ad A □ (B □ C). Se i due grafi in input sono bipartiti allora il prodotto cartesiano manterrà tale proprietà nel grafo risultante.

Il prodotto cartesiano *A* □ *B* crea un grafo di ordine |*A*| *x* |*B*| con (|*E(A)*| x |*B*| + |*E(B)*| x |*A*|) spigoli. Il grado massimo di tale prodotto sarà, quindi, pari a *d(A)* + *d(B)*, mentre l'analisi del diametro del risultante appare meno intuitiva ma, empiricamente, si verifica che *k(A* □ *B)=k(A) + k(B).*

```
In[3]:= CartesianProductGraph[A_, B_] :=
        Module[{i, j, flagEdgeA, flagEdgeB, g, N},
         N = Round[(VertexCount[B])^(1/10)] + 1;
         g := Graph[{-1 ↔ -2}, VertexLabels → "Name"];
         For[i = 1, i ≤ VertexCount[A], i++,
          For[j = 1, j ≤ VertexCount[B], j++,
                g = VertexAdd[g, (((VertexList[A][[i]]) * (10^N)) + (VertexList[B][[j]]))];
           ];
         ];
         g = EdgeDelete[g, {-1 ↔ -2}];
         g = VertexDelete[g, {-1, -2}];
         For[i = 1, i ≤ VertexCount[g], i++,
          For[j = i, j ≤ VertexCount[g], j++,
            If[((Quotient[VertexList[g][[j]], 10^N]) ==
                 (Quotient[VertexList[g][[i]], 10^N])) && (Intersection[EdgeList[B],
                  {(Mod[VertexList[g][[j]], 10^N]) ↔ (Mod[(VertexList[g][[i]]), 10^N]),
                   (Mod[(VertexList[g][[i]]), 10^N]) ↔ (Mod[VertexList[g][[j]], 10^N])}] ≠ {}),
                 flagEdgeA = True,
                 flagEdgeA = False];
            If[((Mod[VertexList[g][[j]], 10^N]) ==
                 (Mod[(VertexList[g][[i]]), 10^N])) && (Intersection[EdgeList[A],
                  {(Quotient[VertexList[g][[j]], 10^N]) ↔ (Quotient[VertexList[g][[i]], 10^N]),
                   (Quotient[VertexList[g][[i]], 10^N]) ↔ (Quotient[VertexList[g][[j]], 10^N])}] ≠ {}),
                 flagEdgeB = True,
                 flagEdgeB = False];
            If[ ((flagEdgeA  || flagEdgeB) &&
                (Intersection[EdgeList[g],
                   {(VertexList[g][[i]]) ↔ (VertexList[g][[j]]),
                    (VertexList[g][[j]]) ↔ (VertexList[g][[i]])}] == {})),
                 g = EdgeAdd[g, {(VertexList[g][[i]]) ↔ (VertexList[g][[j]])}],
                 g = g];
           ];
         ];
         Return[g];
        ];
```

Codice 1 Implementazione del prodotto cartesiano fra grafi

Queste osservazioni inducono a pensare che questa tipologia di prodotto sia poco performante ai fini del Degree-Diameter problem, in ogni caso, si è proceduto all'implementazione di quest'operazione come da codice 1.

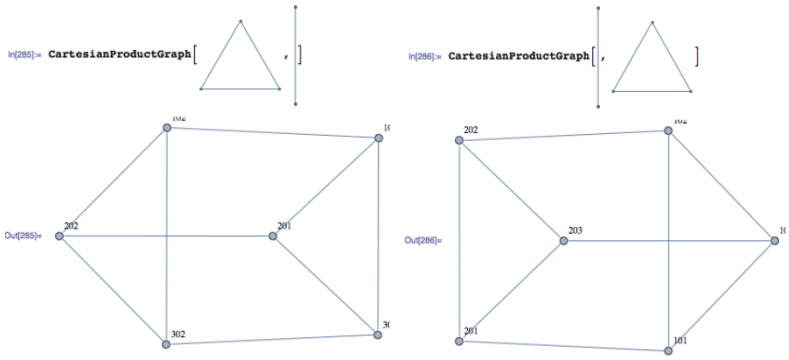


Figura 7 K3 □ K2 e K2 □ K3

3.2.2 Prodotto lessicografico

In quest'operazione l'insieme degli spigoli appartenenti a $A \cdot B$ è dato da tutte le coppie di nodi (a, b), (a^I, b^I) in cui a è adiacente ad a^I in *A* oppure $a = a^I$ e b è adiacente a b^I in *B.* Il prodotto lessicografico non gode né della proprietà associativa né della proprietà commutativa.

Il grado massimo del grafo prodotto $A \cdot B$ sarà dato da $d(A)|B| + d(B)$ in quanto il nodo con grado massimo di *A* originerà nodi aventi:

- spigoli verso tutti i $|B|$ nodi che avranno, in a^I, uno spigolo a cui a è adiacente in *A*; vi saranno $d(A)$ diversi a^I, da cui $d(A)|B|$;
- $d(B)$ spigoli verranno, nei nodi di grado massimo di $A \cdot B$, dalle coppie (a, b), (a, b^I) in cui b è il nodo di grado massimo di *B,* essendo $a = a^I$.

I risultati ottenuti da J. C´Aceres, C. Hernando, M. Mora, I.M. Pelayo, e M.L. Puertas (presenti in Boundary-type sets and product operators in graphs), invece, assicurano che $k(A \cdot B)=2$ se *A* è un grafo completo, altrimenti si ha $k(A \cdot B)=k(A)$. Secondo quanto asserito, dunque, per ottenere buoni risultati con questa tipologia di prodotto il grafo *A* dovrebbe possedere già ottime caratteristiche ma il grado massimo sarebbe comunque inficiato dal numero di nodi di *B*.

```
In[2]:= LexicographicProductGraph[A_, B_] :=
        Module[{i, j, flagEdgeA, flagEdgeB, g, N},
         N = Round[(VertexCount[B])^(1/10)] + 1;
         g := Graph[{-1 ↔ -2}, VertexLabels → "Name"];
         For[i = 1, i ≤ VertexCount[A], i++,
          For[j = 1, j ≤ VertexCount[B], j++,
                g = VertexAdd[g, (((VertexList[A][[i]]) * (10^N)) + (VertexList[B][[j]]))];
           ];
         ];
         g = EdgeDelete[g, {-1 ↔ -2}];
         g = VertexDelete[g, {-1, -2}];
         For[i = 1, i ≤ VertexCount[g], i++,
          For[j = i, j ≤ VertexCount[g], j++,
            If[((Quotient[VertexList[g][[j]], 10^N]) ==
                 (Quotient[VertexList[g][[i]], 10^N])) && (Intersection[EdgeList[B],
                  {(Mod[VertexList[g][[j]], 10^N]) ↔ (Mod[(VertexList[g][[i]]), 10^N]),
                   (Mod[(VertexList[g][[i]]), 10^N]) ↔ (Mod[VertexList[g][[j]], 10^N])}] ≠ {}),
                 flagEdgeA = True,
                 flagEdgeA = False];
            If[(Intersection[EdgeList[A],
                 {(Quotient[VertexList[g][[j]], 10^N]) ↔ (Quotient[VertexList[g][[i]], 10^N]),
                  (Quotient[VertexList[g][[i]], 10^N]) ↔ (Quotient[VertexList[g][[j]], 10^N])}] ≠ {}),
                 flagEdgeB = True,
                 flagEdgeB = False];
            If[ ((flagEdgeA || flagEdgeB) &&
                (Intersection[EdgeList[g],
                   {(VertexList[g][[i]]) ↔ (VertexList[g][[j]]),
                    (VertexList[g][[j]]) ↔ (VertexList[g][[i]])}] == {})),
                 g = EdgeAdd[g, {(VertexList[g][[i]]) ↔ (VertexList[g][[j]])}],
                 g = g];
           ];
         ];
         Return[g];
        ];
```

Codice 2 Implementazione del prodotto lessicografico fra grafi

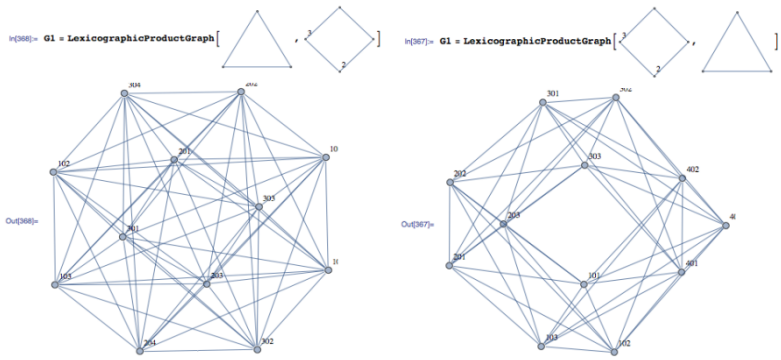


Figura 8 K3 · C4 e C4 · K3

3.2.3 Prodotto forte

L'operazione di prodotto forte $A \boxtimes B$ non gode delle proprietà di associatività o commutatività e l'insieme degli spigoli $E(A \boxtimes B)$, dati i vertici (a, b), (a^I, b^I) con $a, a^I \in A$ e $b, b^I \in B$ è così definito:

- se b è adiacente a b^I allora $(a, b), (a^I, b^I) \in E(A \boxtimes B)$;
- se $b = b^I$ e a è adiacente a a^I allora $(a, b), (a^I, b^I) \in E(A \boxtimes B)$;

- se $b = b^I$ e $a = a^I$ allora (a, b), $(a^I, b^I) \in E(A \boxtimes B)$.

Si osserva però che l'ultima condizione sopra elencata produce unicamente auto-archi e pertanto si ridefinisce questo prodotto eliminando tale condizione, si avrà, quindi, la seguente implementazione:

```
StrongProductGraphB[A_, B_] :=
  Module[{i, j, flagEdgeA, flagEdgeB, flagEdgeC, g, N},
   N = Round[(VertexCount[B])^(1/10)] + 1;
   g := Graph[{-1 ↔ -2}, VertexLabels → "Name"];
   For[i = 1, i ≤ VertexCount[A], i++,
    For[j = 1, j ≤ VertexCount[B], j++,
          g = VertexAdd[g, (((VertexList[A][[i]]) * (10^N)) + (VertexList[B][[j]]))];
     ];
   ];
   g = EdgeDelete[g, {-1 ↔ -2}];
   g = VertexDelete[g, {-1, -2}];
   For[i = 1, i ≤ VertexCount[g], i++,
    For[j = i, j ≤ VertexCount[g], j++,
      If[((Mod[VertexList[g][[j]], 10^N]) ==
           (Mod[VertexList[g][[i]], 10^N])) && ((Intersection[EdgeList[B],
             {(Quotient[VertexList[g][[j]], 10^N]) ↔ (Quotient[(VertexList[g][[i]]), 10^N]),
              (Quotient[(VertexList[g][[i]]), 10^N]) ↔ (Quotient[VertexList[g][[j]], 10^N])}] ≠ {})),
          flagEdgeA = True,
          flagEdgeA = False];
      (*If[((Mod[VertexList[g][[j]],10^N])==
           (Mod[VertexList[g][[i]],10^N]))&&((Quotient[VertexList[g][[j]],10^N])==
           (Quotient[VertexList[g][[i]],10^N])),
            flagEdgeC=True,
            flagEdgeC=False];*)
      If[(Intersection[EdgeList[B],
           {(Mod[VertexList[g][[j]], 10^N]) ↔ (Mod[(VertexList[g][[i]]), 10^N]),
            (Mod[(VertexList[g][[i]]), 10^N]) ↔ (Mod[VertexList[g][[j]], 10^N])}] ≠ {}),
          flagEdgeB = True,
          flagEdgeB = False];
      If[ ((flagEdgeA || flagEdgeB(*||flagEdgeC*)) &&
          (Intersection[EdgeList[g],
             {(VertexList[g][[i]]) ↔ (VertexList[g][[j]]),
              (VertexList[g][[j]]) ↔ (VertexList[g][[i]])}] == {})),
         g = EdgeAdd[g, {(VertexList[g][[i]]) ↔ (VertexList[g][[j]])}],
         g = g];
     ];
    ];
    Return[g];
   ];
```

Codice 3 Implementazione del prodotto forte fra grafi

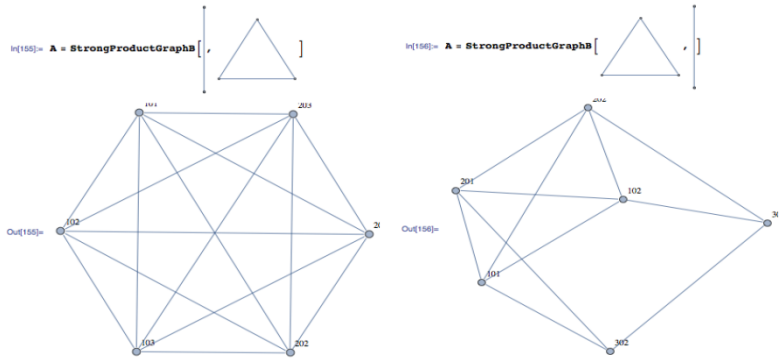


Figura 9 K2 ⊠ K3 e K3 ⊠ K2

Con questa modifica si ottengono i seguenti risultati per grado e diametro:

- *d (A ⊠ B) = d(B)(|A| + 1);*
- *k (A ⊠ B) = k(B)* se *B* non è un grafo completo, è *2* se *B* è un grafo completo.

3.2.4 Prodotto diretto

A X B è il prodotto diretto di *A* e *B* (anche detto prodotto tensore) ed ha come spigoli tutte le coppie di vertici (a, b), (a^I, b^I) tali per cui a, a^I sono adiacenti in *A* e b, b^I sono adiacenti in *B.* Si ricorda che l'insieme dei nodi è dato dal prodotto cartesiano degli insiemi *V(A)* e *V(B).* Si deduce che quest'operazione gode della proprietà commutativa; il grado massimo del grafo prodotto si ottiene come prodotto dei gradi massimi dei grafi *A* e *B*. Non è invece possibile stabilire a priori il valore del diametro in quanto dipende dalla forma dei grafi di partenza e non dai loro valori di ordine, grado e diametro.

```
In[218]:= DirectProductGraph[A_, B_] :=
 Module[{i, j, flagEdgeA, flagEdgeB, g, N},
  N = Round[(VertexCount[B])^(1/10)] + 1;
  g := Graph[{-1 ↔ -2}, VertexLabels → "Name"];
  For[i = 1, i ≤ VertexCount[A], i++,
   For[j = 1, j ≤ VertexCount[B], j++,
         g = VertexAdd[g, (((VertexList[A][[i]]) * (10^N)) + (VertexList[B][[j]]))];
    ];
  ];
  g = EdgeDelete[g, {-1 ↔ -2}];
  g = VertexDelete[g, {-1, -2}];
  For[i = 1, i ≤ VertexCount[g], i++,
   For[j = i, j ≤ VertexCount[g], j++,
     If[(Intersection[EdgeList[B],
         {(Mod[VertexList[g][[j]], 10^N]) ↔ (Mod[(VertexList[g][[i]]), 10^N]),
          (Mod[(VertexList[g][[i]]), 10^N]) ↔ (Mod[VertexList[g][[j]], 10^N])}] ≠ {}),
         flagEdgeA = True,
         flagEdgeA = False];
     If[(Intersection[EdgeList[A],
         {(Quotient[VertexList[g][[j]], 10^N]) ↔ (Quotient[VertexList[g][[i]], 10^N]),
          (Quotient[VertexList[g][[i]], 10^N]) ↔ (Quotient[VertexList[g][[j]], 10^N])}] ≠ {}),
         flagEdgeB = True,
         flagEdgeB = False];
     If[ ((flagEdgeA && flagEdgeB) &&
        (Intersection[EdgeList[g],
           {(VertexList[g][[i]]) ↔ (VertexList[g][[j]]),
            (VertexList[g][[j]]) ↔ (VertexList[g][[i]])}] == {})),
         g = EdgeAdd[g, {(VertexList[g][[i]]) ↔ (VertexList[g][[j]])}],
         g = g];
    ];
   ];
   Return[g];
 ];
```

Codice 4 Implementazione prodotto diretto fra grafi

3.2.5 Prodotto *

Dalle ricerche effettuate in letteratura, ed in particolare, in uno scritto congiunto di C. Bermond, C. Delorme, e G. Farhi (1981), si rileva l'esistenza di un particolare prodotto fra grafi appositamente creato per lo scopo di formare grafi di grandi dimensioni con grado e diametro contenuti; tale prodotto risulta più sofisticato dei precedenti ma assicura un grado massimo risultante pari a *d(A) + d(B)* e un diametro *k(A * B)* ≤ *k(A) + k(B)*. Questo prodotto ha come insieme dei vertici il prodotto cartesiano di *V(A)* e *V(B)*. La determinazione degli spigoli risulta, invece, più complessa: al fine di determinare gli spigoli si crea un insieme *U(A)* composto dagli archi ottenuti da un orientamento scelto arbitrariamente per gli spigoli di *A*, per ciascun arco in *U*, poi, per ogni arco (a, a^I), si definisce una funzione $f_{(a,a^I)}$ che mappi completamente da *V(B)* a *V(B)*. Date le suddette premesse una coppia di vertici (a, b), (a^I, b^I) risulta collegata in *A * B* se:

- $a = a^I$ e b è adiacente a b^I in *B*; o
- $(a, a^I) \in U(A)$ e $b^I = f_{(a,a^I)}(b)$.

La scelta delle $f_{(a,a^I)}$ influenza il valore del diametro di *(A * B)* ma chiaramente non ha conseguenze sul grado massimo di tale prodotto.

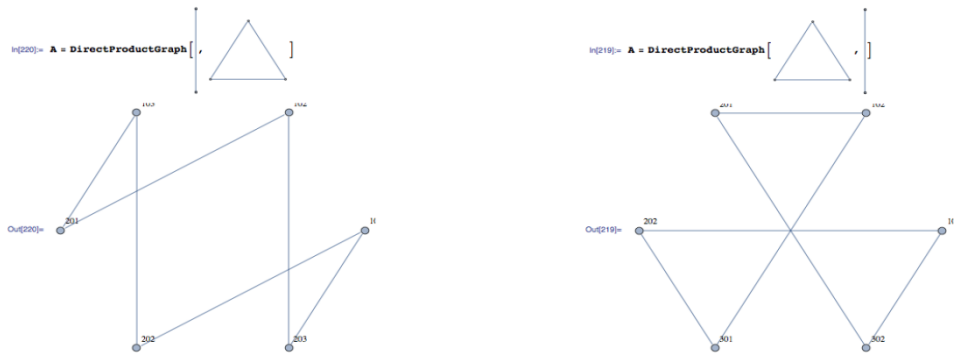


Figura 10 K2 x K3 e K3 x K2

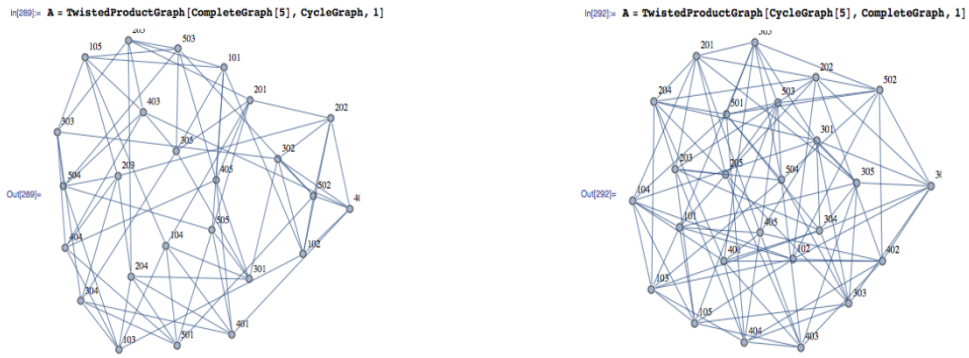


Figura 11 K5 * C5 e C5 * K5

Nell'implementazione effettuata si è utilizzata come funzione di mapping $f_{(a,a^I)}(b) = \big((2a+1)b\%(2a^2+2a+1)\big)+1$, pertanto, B avrà un ordine di $(2a^2+2a+1)$ affinchè il prodotto lavori al meglio. Si è considerato ogni spigolo di A (a,a^I) come arco orientato dal nodo con valore di etichetta minore al nodo collegato con etichetta maggiore.

```
In[271]:= TwistedProductGraph[A_, BB_, a_] :=
 Module[{i, j, O, x, flagEdgeA, flagEdgeB, g, N, B},
  O = 2 (a^2) + 2 a + 1;
  B = BB[O];
  N = Round[(VertexCount[B])^(1 / 10)] + 1;

  g := Graph[{-1 ↔ -2}, VertexLabels → "Name"];
  For[i = 1, i ≤ VertexCount[A], i++,
   For[j = 1, j ≤ VertexCount[B], j++,
         g = VertexAdd[g, (((VertexList[A][[i]]) * (10^N)) + (VertexList[B][[j]]))];
    ];
  ];
  g = EdgeDelete[g, {-1 ↔ -2}];
  g = VertexDelete[g, {-1, -2}];
  For[i = 1, i ≤ VertexCount[g], i++,
   For[j = 1, j ≤ VertexCount[g], j++,
     If[((Quotient[VertexList[g][[j]], 10^N]) == (Quotient[VertexList[g][[i]], 10^N]) &&
       (Intersection[EdgeList[B],
          {(Mod[VertexList[g][[j]], 10^N]) ↔ (Mod[(VertexList[g][[i]]), 10^N]),
           (Mod[(VertexList[g][[i]]), 10^N]) ↔ (Mod[VertexList[g][[j]], 10^N])}] ≠ {})),
        flagEdgeA = True,
        flagEdgeA = False];
     If[((Quotient[VertexList[g][[j]], 10^N]) < (Quotient[VertexList[g][[i]], 10^N]) &&
       (Mod[VertexList[g][[j]], 10^N]) == (Mod[((2 * a) + 1) * (Mod[VertexList[g][[i]], 10^N]), O] + 1)),
        flagEdgeB = True,
        flagEdgeB = False];
     If[ ((flagEdgeA || flagEdgeB) &&
       (Intersection[EdgeList[g],
          {(VertexList[g][[i]]) ↔ (VertexList[g][[j]]),
           (VertexList[g][[j]]) ↔ (VertexList[g][[i]])}] == {})),
        g = EdgeAdd[g, {(VertexList[g][[i]]) ↔ (VertexList[g][[j]])}],
        g = g];
    ];
   ];
   Return[g];
  ];
```

Codice 5 Implementazione prodotto * fra grafi

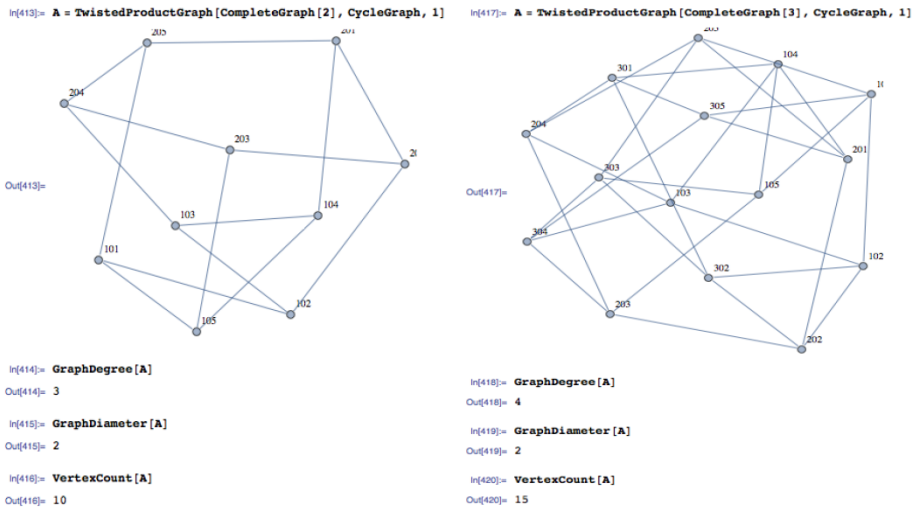


Figura 12 K2 * C5 e K3 * C5

3.3 Conclusioni

Si può facilmente osservare che, fra i prodotti analizzati, quello più performante risulta essere il prodotto *, infatti, per questa operazione grado massimo e diametro risultano dalla somma di tali valori nei grafi originari. Si sottolinea però come questa operazione risulti vantaggiosa solo per i grafi di piccole dimensioni: per il vincolo di Moore l'ordine del grafo dovrebbe aumentare, come ordine di grandezza, in funzione di d^k, invece, le famiglie fin qui analizzate crescono mediamente in ordine lineare o con il grado o con il diametro, pertanto il prodotto di due famiglie di questo tipo darà luogo ad un grafo che sarà di ordine $\theta(d * k)$. Si conclude quindi che il prodotto sopra visto può dare ottimi risultati ma solo nella direzione di grafi di piccole dimensioni. Si riportano di seguito alcune composizioni effettuate con una lieve modifica dell'algoritmo di prodotto sopra riportato, modifiche utili ad utilizzare tale operazione anche su grafi a farfalla (che risultano dall'analisi svolta i più performanti in merito a questo problema).

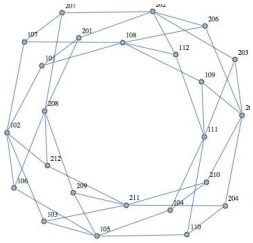


Figura 13
K2*ButterflyGraph[2,2]
d = 5, k = 4, Ordine = 24
$\alpha = 0.06$

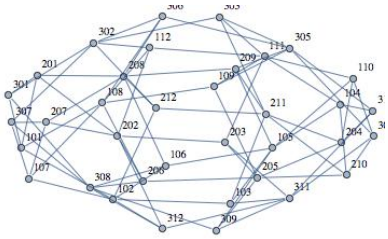


Figura 15
K3*ButterflyGraph[2,2]
d = 6, k = 4, Ordine = 36
$\alpha = 0.04$

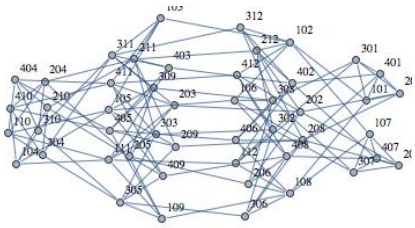


Figura 16
C4*ButterflyGraph[2,2]
d = 7, k = 4, Ordine = 48
$\alpha = 0.03$

4. Elaborazione di euristiche

4.1 Perturbazione di Grafi

Non essendo emersa nell'analisi delle famiglie di grafi precedentemente fatta una famiglia che, più delle altre, presenti significative qualità si procederà alla perturbazione ed al miglioramento di grafi ottenuti da una generazione pseudo-casuale. Si osserva sperimentalmente che dati *n* nodi ed $\frac{n(n-1)}{4}$ archi il diametro rimane $o(5)$ mentre il grado massimo risulta $o(\frac{n}{2})$. Come già dimostrato (Béla Bollobás 2001), infatti, la descrizione di un grafo pseudo-casuale tramite il numero di archi (*G(n, M)* dove *n* rappresenta il numero di nodi e *M* rappresenta il numero di archi) è assimilabile, per proprietà come il diametro e il grado, ad una definizione data come numero di nodi e probabilità p di esistenza di ciascuno spigolo (*G(n, p)*); a condizione che:

$$0 < p = \frac{M}{N} < 1$$

con $N = n(n-1)/2$.

Per evidenziare ciò si è creato un banale algoritmo per la generazione di grafi pseudo-casuali tramite la definizione di probabilità di esistenza per un arco.

Si sono quindi interpolati i dati ottenuti dalle due diverse generazioni in un intervallo di n [1, 150] e con *p=1/2* e *M=pN,* il risultato è il seguente:

```
In[448]:= RandomGraphP[N_, P_] :=
           Module[{G},
            G = Graph[{-1 ↔ -2}];
            For[i = 1, i < N, i++,
             For[j = i + 1, j ≤ N, j++,
               If[Random[] ≤ P,
                 G = EdgeAdd[G, {i ↔ j}]
                ];
              ];
            ];
            G = EdgeDelete[G, {-1 ↔ -2}];
            G = VertexDelete[G, {-1, -2}];
            Return[G];
           ];
```

Codice 6 Implementazione generazione grafo *G(n,p)*

Nelle prossime operazioni, dunque, si utilizzerà una generazione *G(n,M)* per motivi di efficienza degli algoritmi di generazione di tali grafi.

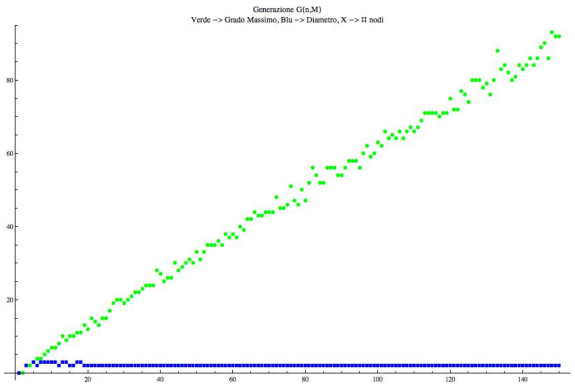


Grafico 9 Generazione di grafi Random *G(n,M)*

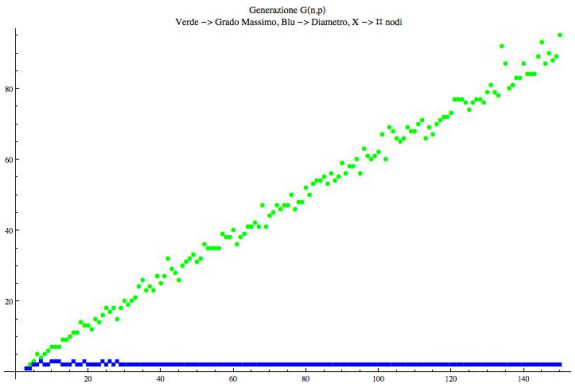


Grafico 10 Generazione di grafi Random *G(n,p)*

4.1.1 Operazioni base

Per migliorare un grafo è possibile agire in tre direzioni:

- diminuire il grado massimo;
- diminuire il diametro;
- aumentare l'ordine del grafo.

Per diminuire il grado massimo è necessario, banalmente, ridurre il numero di spigoli presenti nei nodi aventi tale grado, quest'operazione può portare, per contro, ad un aumento del diametro. La prima soluzione proposta prevede l'eliminazione degli spigoli presenti fra i nodi di grado massimo e i loro vicini di grado maggiore. Con questa soluzione si ottiene una riduzione del grado di un'unità. Applicato su grafi poco connessi può, soprattutto se reiterato, rendere il grafo sconnesso cosa che comporta per convenzione un diametro di valore infinito.

Come si può osservare questa procedura ottiene risultati migliori su grafi non fortemente connessi.

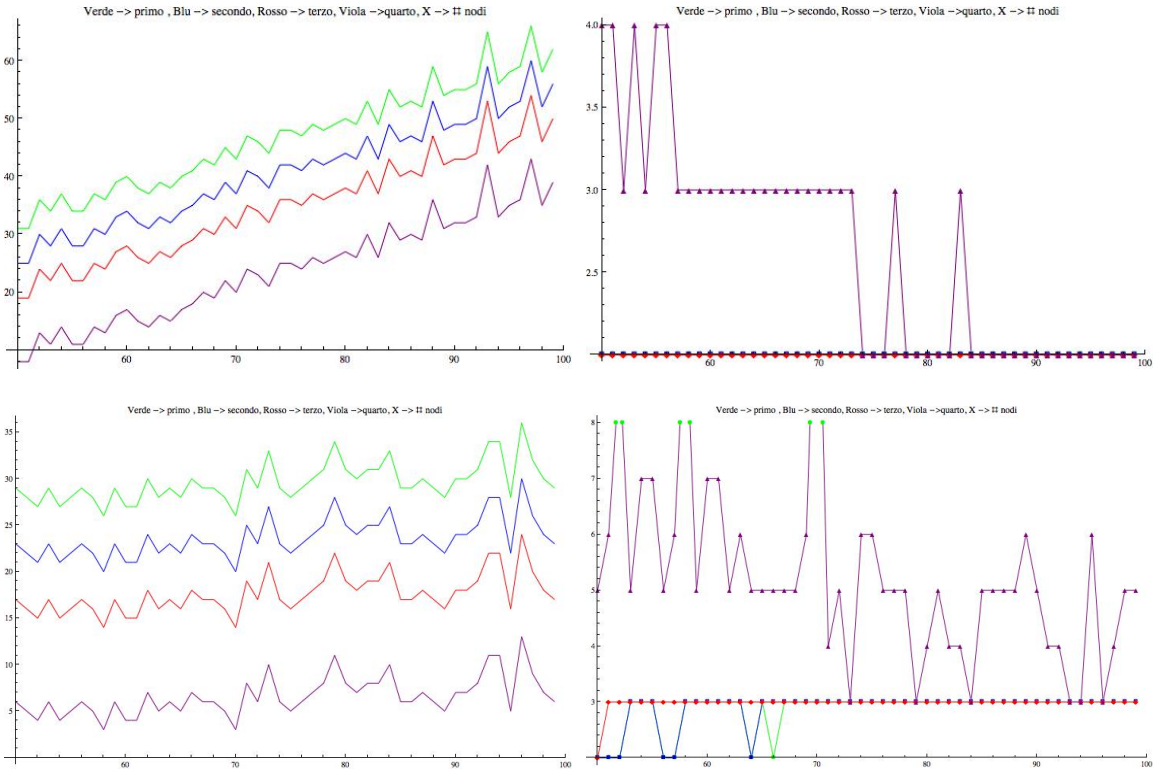


Grafico 11 Applicazione della procedura AbbassaGrado

I grafici riportati mostrano, a sinistra, l'andamento del grado e, a destra, quello del diametro, i grafi riportati sopra si riferiscono a generazioni pseudo casuali di grafi fortemente connessi (numero di archi pari a $\frac{n(n-1)}{4}$) mentre i grafici posti sotto corrispondo a grafi più debolmente connessi (numero di archi pari a $10n$); i diversi colori, invece, indicano il diverso numero d'iterazioni (rispettivamente *verde, blu, rosso* e *viola* indicano 0, 5, 10 e 20 iterazioni).In ascissa si riporta il valore dell'ordine del grafo. I valori risultanti sono stati connessi cosi da rendere più leggibile il grafico.

```
In[337]:= AbbassaGrado[A_] :=
           Module[{AA, i, j, Nodo, ListaNodiGradoMassimo, ListaAdiacenti, max, MaxP},
            AA = A;
            ListaNodiGradoMassimo = Position[VertexDegree[AA], GraphDegree[A]];
            If[GraphDegree[A] > 2,
             While[Length[ListaNodiGradoMassimo] > 0,
               Nodo = ListaNodiGradoMassimo[[1, 1]];
               ListaAdiacenti = VertexComponent[AA, Nodo, 1];
               ListaAdiacenti = Delete[ListaAdiacenti, Position[ListaAdiacenti, Nodo]];
               max = 1;
               For[j = 1, j ≤ Length[ListaAdiacenti], j++,
                If[VertexDegree[AA, ListaAdiacenti[[j]]] ≥ max,
                  max = VertexDegree[AA, ListaAdiacenti[[j]]];
                  MaxP = ListaAdiacenti[[j]];
                 ];
               ];
               AA = EdgeDelete[AA, Intersection[EdgeList[AA], {MaxP ↔ Nodo, Nodo ↔ MaxP}]];
               ListaNodiGradoMassimo = Position[VertexDegree[AA], GraphDegree[A]];
              ];
            ];
            Return[AA];
           ];
```

Codice 7 Implementazione di una prima versione di AbbassaGrado

La versione qui presentata è la seconda descritta poiché alla prova pratica ha mostrato risultati migliori

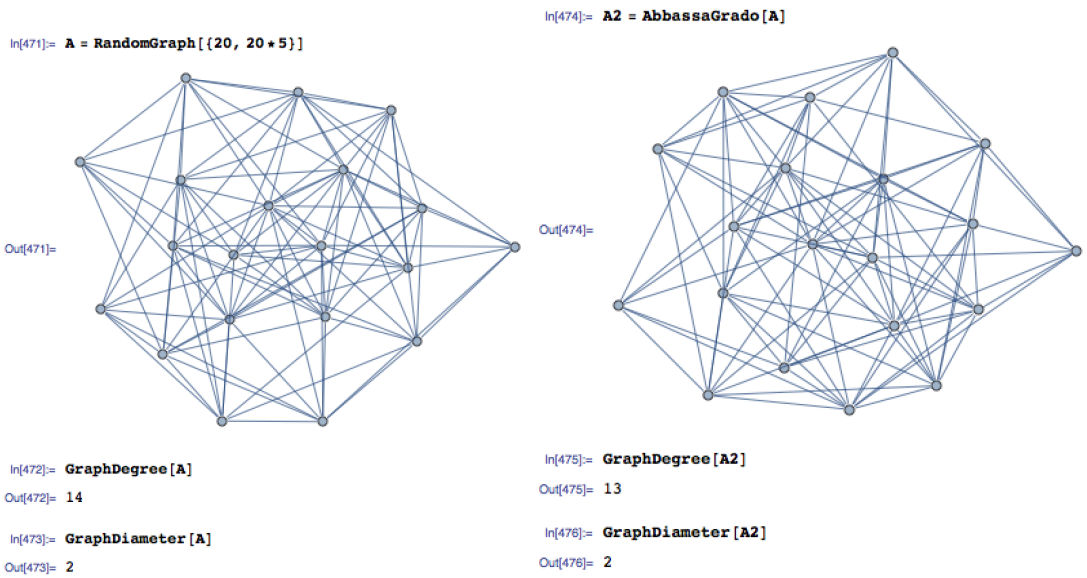


Figura 17 Applicazione della funzione per l'abbassamento del grado

Il grafo random su cui si è applicata la procedura, come si può osservare, ha numero di spigoli approssimabile ad N/2

Per ottenere una diminuzione del diametro è necessario ridurre la distanza massima, un'euristica per ottenere questo risultato consta nel diminuire il numero di vertici periferici. Parliamo di quei vertici che, avendo meno connessioni e quindi un grado più ridotto, risultano meno collegati agli altri nodi del grafo. Si sceglieranno, dunque, i nodi con grado minimo e, per connetterli al

meglio al resto del grafo verranno selezionati altri vertici con grado alto, cercando, ove possibile di evitare di aggiungere spigoli a nodi di grado massimo. Si ha questa premura per lenire l'effetto negativo che questo tipo di risoluzione comporta nei confronti del grado massimo di un grafo. In questo modo sarà possibile, con buone probabilità, diminuire il diametro.

```
In[606]:= AbbassaDiametro[A_] :=
 Module[{AA, i, p, k, Nodo, App, ListaMin, ListaMax, Z},
  AA = A;
  If[GraphDiameter[A] != 1,
   ListaMin = Position[VertexDegree[AA], GraphMinDegree[A]];
   For[i = 1, i ≤ Length[ListaMin], i++,
    p = 1;
    ListaMax = Position[VertexDegree[AA], GraphDegree[A] - p];
    While[Length[ListaMax] == 0 && p < GraphDegree[A],
     p++;
     ListaMax = Position[VertexDegree[AA], GraphDegree[A] - p];
    ];
    If[Length[ListaMax] == 0,
     p = 0;
     ListaMax = Position[VertexDegree[AA], GraphDegree[A] - p];
    ];
    While[Length[ListaMax] == 0,
     ListaMax = Position[VertexDegree[AA], GraphDegree[A] - p];
     p--;
    ];
    Nodo = ListaMin[[1, 1]];
    App = ListaMax[[Random[Integer, {1, Length[ListaMax]}], 1]];
    If[Intersection[EdgeList[AA], {Nodo ↔ App, App ↔ Nodo}] == {} && App != Nodo,

     AA = EdgeAdd[AA, {Nodo ↔ App}];,

     While[Intersection[EdgeList[AA], {Nodo ↔ App, App ↔ Nodo}] != {} || App == Nodo,
      App = Random[Integer, {1, VertexCount[A]}];
     ];
     AA = EdgeAdd[AA, {Nodo ↔ App}];
    ];
    ListaMin = Position[VertexDegree[AA], GraphMinDegree[A]];
    ListaMax = Position[VertexDegree[AA], GraphDegree[A]];

    k = 1;
    While[k < GraphDegree[A] && Length[ListaMax == 0],
     ListaMax = Position[VertexDegree[AA], GraphDegree[A] - k];
     k++;
    ];
    If[Length[ListaMax] == 0,
     ListaMax = Position[VertexDegree[AA], GraphDegree[AA]];
    ];
   ];
  ];
  Return[AA];
 ];
```

Codice 8 Implementazione euristica per diminuzione del diametro

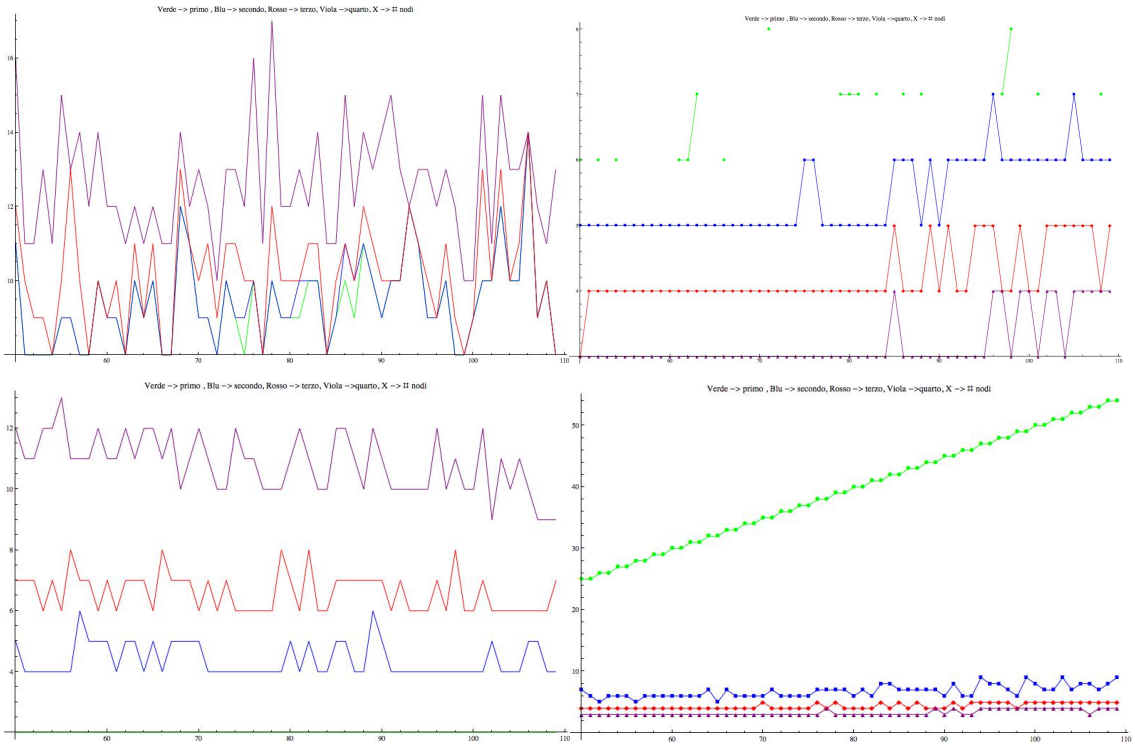


Grafico 10 Applicazione della procedura AbbasaDiametro

I grafici riportati mostrano, a sinistra, l'andamento del grado e, a destra, quello del diametro, In questa occasione si è preferito utilizzare grafi debolmente connessi (numero di archi pari a $2n$), i cui risultati appaiono nei grafici superiori, e grafi ciclici i cui grafici appaiono inferiormente. Questa decisione è stata presa al fine di poter apprezzare un miglioramento sui valori del diametro che, in grafi meno sparsi, avrebbe potuto attestarsi a 2 già dopo poche iterazioni. I diversi colori, invece indicano il diverso numero d'iterazioni su di uno stesso grafo (rispettivamente, verde, blu, rosso e viola indicano 0, 6, 17 e 38 iterazioni).In ascissa si riporta il valore dell'ordine del grafo. I valori risultanti sono stati connessi cosi da rendere più leggibile il grafico.

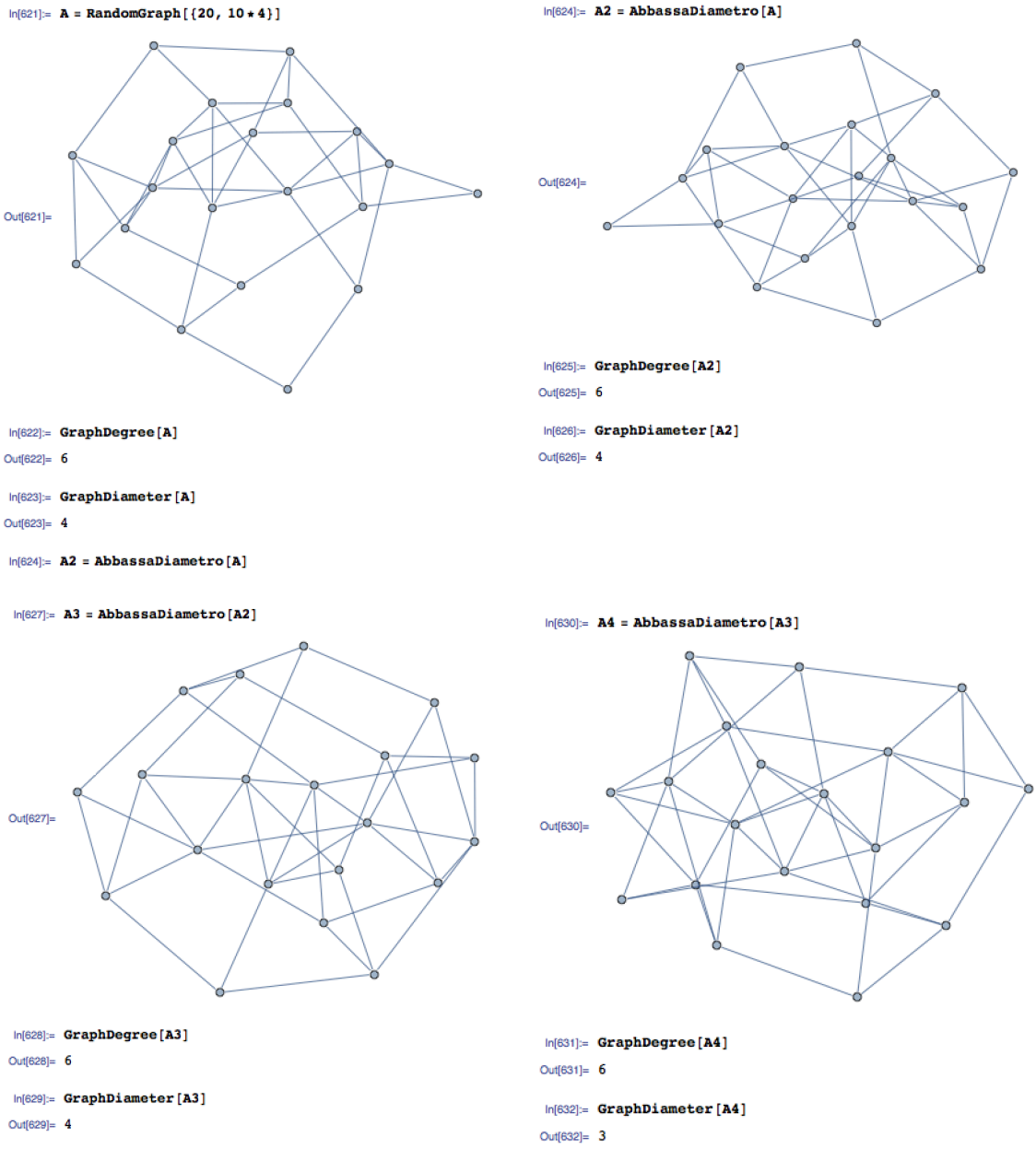


Figura 14 Applicazione della funzione per la riduzione del diametro

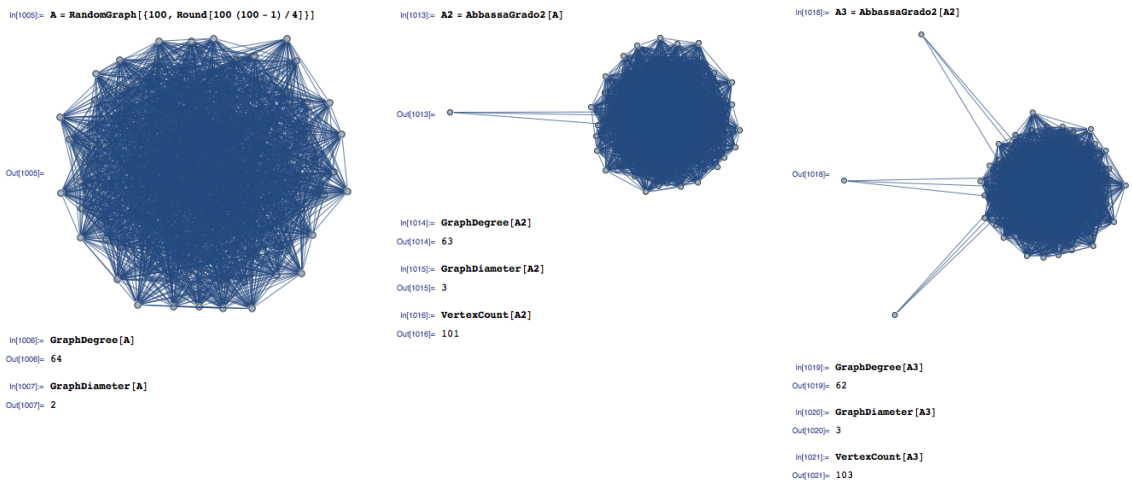
Si sottolinea, anche in queste immagini, che possono essere necessarie diverse iterazioni prima di ottenere il risultato sperato. Per agevolare l'esposizione si sono utilizzati grafi sparsi e di piccole dimensioni.

L'aumento dell'ordine di un grafo, con l'aggiunta di uno o più nodi può provocare con facilità un incremento del diametro. Per fare fronte a questa evenienza si è pensato ad un metodo che unisse l'aggiunta di un nuovo nodo alla diminuzione del grado massimo. L'euristica risultante, designato con A il grafo in esame, segue questa linea:

- si individuano i nodi di grado massimo $U(A)$;

- per ciascun $b \in U(A)$ si individuano i due nodi c, f di grado massimo fra i nodi adiacenti a b;
- si aggiunge, quindi, per ciascun b un nodo a collegandolo a b, c ed f;
- si procede all'eliminazione degli spigoli $b \leftrightarrow c$ e $b \leftrightarrow f$.

Analizzando questo procedimento si osserva che l'ordine aumenta di |U(A)|, tale aumento può però verificarsi anche per il valore del diametro; il grado massimo subirà una diminuzione di un'unità, ammesso che d(A)≥4.

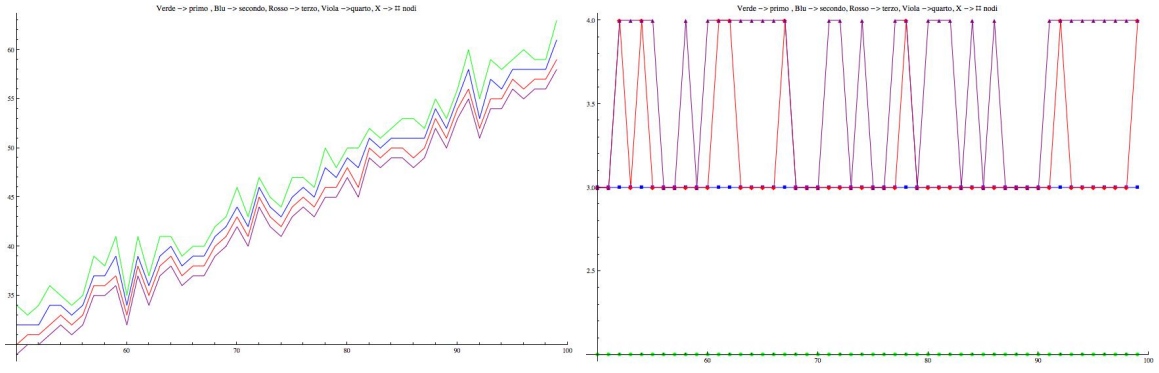


Figua 16 Applicazione del secondo metodo per la riduzione del grado

```
In[988]:= AbbassaGrado2[AA_] :=
  Module[{A, i, j, flag, m, h, t, M, k, max, max2, adL, adLG, adLGS},
   A = AA;
   flag = True;
   j = Max[VertexDegree[A]];
   For[i = 1, i ≤ VertexCount[A], i++,
    If[VertexDegree[A][[i]] == j,
      max2 = -1;
      max = -1;
      adL = VertexComponent[A, i, 1];
      adL = Delete[adL, Position[adL, i][[1, 1]]];
      adLG = List[];
      k = 1;
      While[k ≤ Length[adL],
       If[VertexDegree[A][[adL[[k]]]] ≥ (j - 1),
        adL = Delete[adL, k];,
        adLG = Append[adLG, VertexDegree[A][[adL[[k]]]]];
        k++;
       ];
      ];
      If[Length[adLG] > 0,
       adLGS = Sort[adLG, Greater];
       max = adL[[Position[adLG, adLGS[[1]]][[1, 1]]]];
       max2 = max;
       h = 2;
       While[h ≤ Length[adLG] && max2 == max,
        t = 0;
        While[t ≤ Length[Position[adL, adLGS[[h]]]] && max2 == max,
         t++;
         max2 = adL[[Position[adLG, adLGS[[h]]][[t, 1]]]];
         If[max2 ≠ max, flag = False];

        ];
        h++;
       ];
       If[! flag,
        A = VertexAdd[A, (VertexCount[A] + 1)];
        A = EdgeAdd[A, {(VertexCount[A]) ↔ max, (VertexCount[A]) ↔ max2, (VertexCount[A]) ↔ i}];
        A = EdgeDelete[A, Intersection[EdgeList[A], {max ↔ i, i ↔ max, max2 ↔ i, i ↔ max2}]];
       ];
       flag = True;
      ];
     ];
    ];
   Return[A];
  ];
```

Codice 9 Implementazione secondo metodo per la riduzione del grado



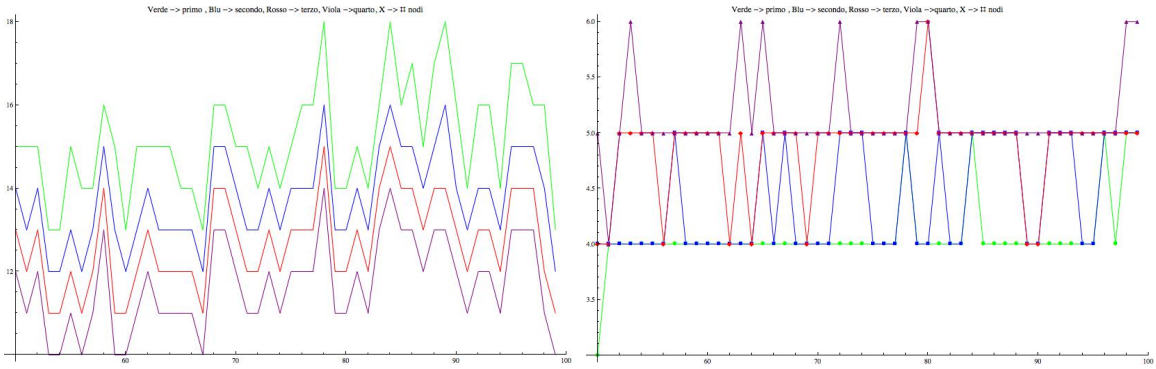


Grafico 11 Applicazione della procedura AbbassaGrado2

I grafici a sinistra indicano i valori del grado massimo, a destra, invece vi sono i valori del diametro. Si sono presi in considerazioni grafi densi nel caso dei primi due grafici e grafi sparsi per gli ultimi due. I dati sono stati registrati ad intervalli di una sola iterazione.

4.1.2 Euristica prima versione

Un primo approccio è stato affrontato perseguendo l'ottimo locale ed eseguendo, quindi, le tre differenti operazioni "AbbassaGrado", "AbbassaDiametro" e "AbbassaGrado2" sullo stesso grafo mantenendo, in seguito, il risultato migliore ottenuto e reiterando il processo più volte. Si è aggiunto un meccanismo di "checkpoint" che mantiene traccia del miglior risultato ottenuto e della conseguente scelta effettuata fra le tre possibili; in caso un certo numero d'iterazioni, stabilito in proporzione al numero massimo di cicli da compiere, non portino a un miglioramento, l'euristica provvede a ripristinare il miglior grafo ottenuto curandosi di effettuare, successivamente, una scelta diversa da quella presa in prima battuta seguendo l'ottimo locale. In caso si esegua il ripristino sopra descritto e, nuovamente, non si avessero miglioramenti, verrà effettuato un ulteriore ripristino prendendo l'ultima scelta rimasta. Se ancora una volta non vi fossero miglioramenti, allora, s'innalzerà la "soglia di tolleranza" di cicli consecutivi senza miglioramenti. Tutti i confronti fra grafi si sono effettuati sulla base del valore $\alpha(G)$ definito in precedenza, cosi da poter avere un metro assoluto di paragone.

```
In[327]:= EuristicaV1[A_, B_] :=
  Module[{Salva, AA, AA2, AA3, Alp, i, SalvaS, S, Soglia, Cambio},
   AA = A;
   AA2 = AA;
   AA3 = AA;
   Salva = AA2;
   Cambio = 0;
   For[i = 1, i ≤ B, i++,
    AA = AbbassaGrado[AA];
    AA2 = AbbassaDiametro[AA];
    AA3 = AbbassaGrado2[AA];
    If[Max[Alpha[AA2], Alpha[AA], Alpha[AA3]] == Alpha[AA2],
     AA = AA2;
     S = 2;,
     If[Max[Alpha[AA2], Alpha[AA], Alpha[AA3]] == Alpha[AA3],
       AA = AA3;
       S = 3;,
       S = 1;
      ];
    ];
    If[Alpha[AA] > Alpha[Salva],
     Salva = AA;
     k = 0;
     Cambio = 0;
     Soglia = 10;
    ];
    If[k == 1, SalvaS = S];
    If[k == Round[B / Soglia],
     Cambio++;
     If[Cambio == 2,
      Soglia = Round[Soglia / 2];
     ];
     SalvaS = (SalvaS %3) + 1
         If[SalvaS == 1,
          AA = AbbassaGrado[Salva];,
          If[SalvaS == 2,
           AA = AbbassaDiametro[Salva];,
           AA = AbbassaGrado2[Salva];
          ];
          k = 0;
         ];
    ];
    k++;
   ];
   Return[Salva];
  ];
```

Codice 10 Implementazione della prima versione dell'euristica

4.1.3 Euristica seconda versione

Per questa seconda versione si è cambiato l'approccio risolutivo. Da un'approssimazione per passi successivi inseguendo l'ottimo locale si è passati

ad un algoritmo che, a seconda delle caratteristiche del grafo, adatta le operazioni da effettuare. In particolare viene considerato elemento discriminante delle caratteristiche del grafo il rapporto fra numero di spigoli e numero di vertici, cercando di ottenere cosi grafi il più possibile regolari: se questo valore si discosta troppo dal grado massimo allora si interverrà con un operazione di riduzione del diametro che, con buone probabilità non porterà ad un aumento del grado massimo, in caso il rapporto sia più bilanciato si proporrà un intervento in entrambe i sensi, infine, se il rapporto dovesse essere prossimo al grado massimo verrà effettuata unicamente la procedura per la riduzione del diametro. In caso il risultato ottenuto non sia ritenuto soddisfacente l'euristica provvederà a perturbare il grafo seguendo un algoritmo che estrae dal grafo $\left\lfloor \frac{(d-1)}{2} \right\rfloor$ nodi di grado alto, partendo dal grado massimo e diminuendo progressivamente il valore di grado dei nodi da eliminare, e un intorno, sempre di $\left\lfloor \frac{(d-1)}{2} \right\rfloor$ nodi di grado basso, partendo dal minimo e aumentando gradualmente il grado richiesto per l'estrazione. I nodi eliminati in questo modo dal grafo vengono disposti secondo una formazione a "stella" e collegati per metà a nodi di grado elevato, vicino ma inferiore al massimo e, per l'altra metà a nodi di grado basso a partire dal grado minimo. Si sono, inoltre posti dei controlli

affinché si riduca la probabilità di ottenere grafi sconnessi.

```
In[682]:= EuristicaV2[A_, B_] :=
          Module[{AA, Appoggio, i, j},
           j = 1;
           AA = A;
           For[i = 1, i < B, i++,
            If[Round[EdgeCount[AA] / VertexCount[AA]] * 4 < GraphDegree[AA],
             AA = AbbassaDiametro[AA];,
             If[Round[EdgeCount[AA] / VertexCount[AA]] * 2 < GraphDegree[AA],
               AA = AbbassaGrado[AA];
               AA = AbbassaDiametro[AA];,
               If[Round[EdgeCount[AA] / VertexCount[AA]] * 1.5 < GraphDegree[AA],
                 If[EdgeCount[AA] > 1.5 * VertexCount && GraphDiameter ≠ Infinity,
                   AA = AbbassaGrado[AA];,
                   AA = AbbassaDiametro[AA];
                   AA = AbbassaGrado[AA];
                  ];
                ];
              ];
            ];
            If[i % B == 0 && Alpha[AA] < Alpha[A],
             AA = Perturba[AA];
             While[GraphDiameter[AA] == Infinity,
              AA = Perturba[AA];
             ];
            ];
           ];
           Return[AA];
          ];
```

Codice 11 Implementazione seconda versione dell'euristica

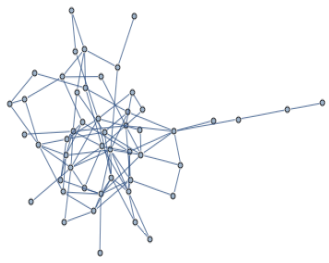
```
In[650]:= Perturba[A_] :=
           Module[{ListaMax, ListaMin, Lista, i, k, j, r, r1, L, AA, New, AAA},
            AA = A;
            k = 0;
            Lista = List[];
            ListaMin = List[];
            ListaMax = List[];
            While[Length[ListaMax] < Round[((GraphDegree[A] - 2) / 2) - 0.5],
             i = 1;
             While[i <= Length[Position[VertexDegree[AA], GraphDegree[AA] - k]] && Length[ListaMax] < Round[((GraphDegree[A] - 2) / 2) - 0.5],
              ListaMax = Append[ListaMax, Position[VertexDegree[AA], GraphDegree[AA] - k][[i, 1]]];
              i++;
             ];
             k++;
            ];
            k = 0;
            While[Length[ListaMin] < Round[((GraphDegree[A] - 2) / 2) - 0.5],
             i = 1;
             While[i <= Length[Position[VertexDegree[AA], GraphMinDegree[AA] + k]] && Length[ListaMin] < Round[((GraphDegree[A] - 2) / 2) - 0.5],
              ListaMin = Append[ListaMin, Position[VertexDegree[AA], GraphMinDegree[AA] + k][[i, 1]]];
              i++;
             ];
             k++;
            ];
            For[i = 1, i ≤ Length[ListaMax], i++,
             Lista = Append[Lista, ListaMax[[i]]];
             AA = VertexDelete[AA, ListaMax[[i]]];
            ];
            For[i = 1, i ≤ Length[ListaMin], i++,
             Lista = Append[Lista, ListaMin[[i]]];
             AA = VertexDelete[AA, ListaMin[[i]]];
            ];
            AAA = AA;
            For[i = 1, i <= Length[Lista], i++,
             r = RandomInteger[{1, VertexCount[AA]}];
             r1 = RandomInteger[{1, VertexCount[AA]}];
             If[VertexCount[AA] ≠ 1,
              While[r1 == r,
               r1 = RandomInteger[{1, VertexCount[AA]}];
               AAA = EdgeAdd[AAA, {Lista[[i]] ↔ VertexList[AA][[r]]}];
              ];
             ];
             AAA = EdgeAdd[AAA, {Lista[[i]] ↔ VertexList[AA][[r]]}];
            ];
            For[i = 2, i <= Length[Lista], i++,
             AAA = EdgeAdd[AAA, {Lista[[i]] ↔ Lista[[1]]}];
            ];
            Return[AAA];
           ];
```

Codice 12 Implementazione del metodo sopra descritto per la perturbazione del grafo

4.1.1 Confronto

I risultati migliori si sono ottenuti con la prima euristica, si osserva anche che il miglioramento in grafi sparsi è più evidente poiché il peso del diametro nella formula del vincolo di Moore è elevato e le due euristiche implementate riescono a ridurlo significativamente.

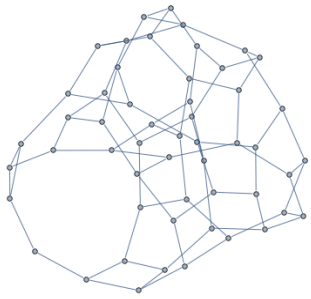
```
In[711]:= A = RandomGraph[{50, 50 * 2}]
```



```
Out[711]=
```

```
In[712]:= N[Alpha[A]]
Out[712]= 0.0000185437
```

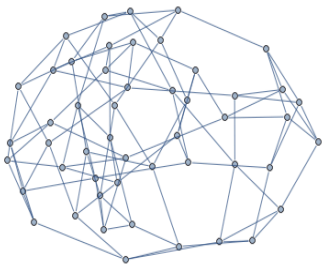
```
In[721]:= X1 = EuristicaV1[A, 1000]
```



```
Out[721]=
```

```
In[722]:= N[Alpha[X1]]
Out[722]= 0.138743
```

```
In[716]:= X2 = EuristicaV2[A, 1000]
```



```
Out[716]=
```

```
In[720]:= N[Alpha[X2]]
Out[720]= 0.103093
```

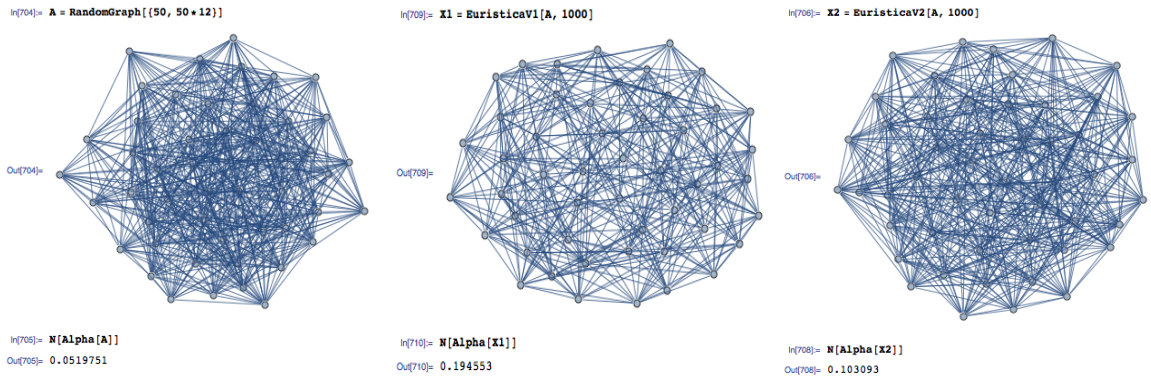


Figura 17 Applicazione delle due Euristiche ad uno stesso Grafo

Già da queste immagini è evidente che i risultati ottenuti dalla prima euristica risultano migliori, rimangono tuttavia distanti dai grafi record per questo problema. È chiaro, inoltre, come i grafi ottenuti dalla seconda euristica siano generalmente più densi. La ricerca dell'ottimo locale risulta comunque una buona strada che andrebbe però percorsa avendo un punto di partenza di buona qualità.

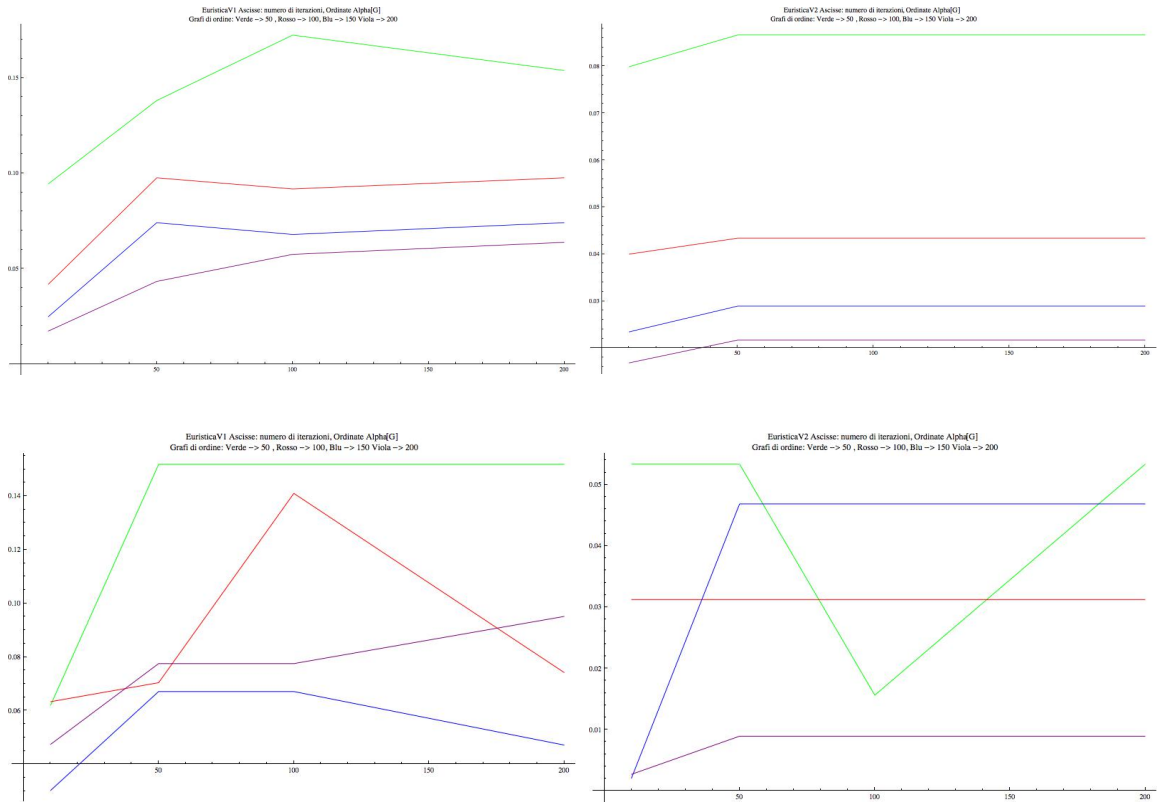


Grafico 12 Applicazione delle due euristiche a grafi densi e sparsi

Dai grafici appare anche come i grafi più densi siano meno suscettibili alle perturbazioni effettuate dalle euristiche e tendano anche a stabilizzarsi con l'aumentare del numero d'iterazioni.

4.2 Creazione di Expanders

"Una famiglia di Expanders è una famiglia di grafi $G_n = (V_n,E_n)$, $|V_n| = n$, tale che ogni grafo sia d_n-regolare, e l'espansione degli spigoli di ogni grafo sia al minimo h, per una costante h assoluta e indipendente da n."

Un grafo si può definire quindi un buon Expander se possiede alte capacità di espansione (che possono portare ad un elevato ordine) ed un grado ridotto, due caratteristiche che accomunano questa categoria di grafi al problema che stiamo esaminando. La caratteristica dell'espansione è definita da un valore h cosi definito:

$$h(G) = \min_{0<|S|\leq\frac{n}{2}} \frac{|\partial(S)|}{|S|}$$

con *S* sottoinsieme di vertici di *G* e $\partial(S)$ è l'insieme di spigoli di confine, cioè tutti gli spigoli che hanno uno e un solo termine in *S.* In letteratura si sono rinvenuti diversi metodi di costruzione, se ne analizzeranno tre.

4.2.1 Costruzione banale

Non si è rinvenuto il nome di questa semplice tecnica di costruzione di Expanders che garantisce una costante di espansione h maggiore di 0 ed un grado massimo pari a 3.

Definito *p* come numero primo e il grafo $G_p = (V_p,E_p)$ nel quale $V_p = \{0,...,p-1\}$, e, per ogni $a \in V_p - \{0\}$, il vertice è connesso a $(a + 1)\%p$, ad $(a - 1)\%p$ e al vertice *i* tale che $(a \times i) \% p$ sia minimo. Il vertice 0 è connesso ad 1, p-1 e ha un auto-arco. Come già evidenziato, contando gli auto-archi, presenti oltre che in 0 anche in 1 e p-1, il grafo è 3-regolare. Poiché la regolarità non è utile ai fini del nostro problema elimineremo gli autoarchi sopra citati.

```
In[804]:= Expanders1[p_] :=
           Module[{k, Lista, i, AA},
            AA = CycleGraph[p];
            For[i = 2, i < p - 1, i++,
             Lista = Table[{Mod[i * j, p]}, {j, 2, p - 1}];
             k = 1 + Position[Lista, Min[Lista]][[1, 1]];
             If[Intersection[EdgeList[AA], {i ⟷ k, k ⟷ i}] == {},
              AA = EdgeAdd[AA, i ⟷ k];
             ];
            ];
            Return[AA];
           ];
```

Codice 13 Implementazione del primo metodo per la creazione di Expanders

Si osserva che i grafi cosi prodotti risultano essere Grafi ciclici con connessioni interne, purtroppo il valore del diametro di questa configurazione risulta elevato, pertanto i valori di $\alpha(G)$ non risultano competitivi.

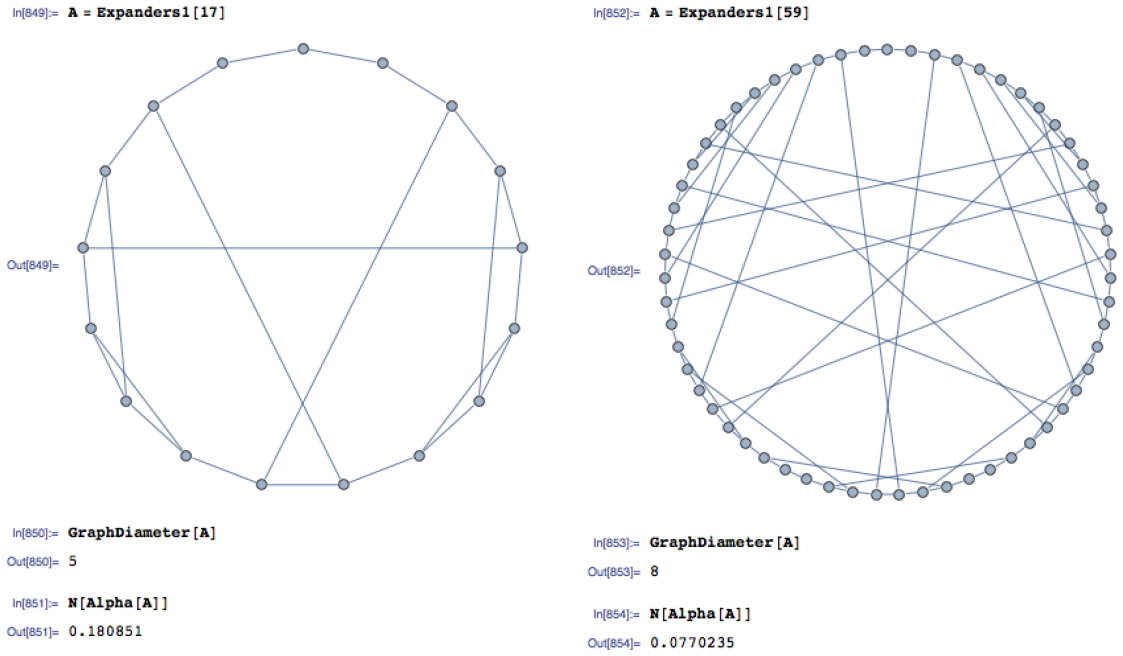
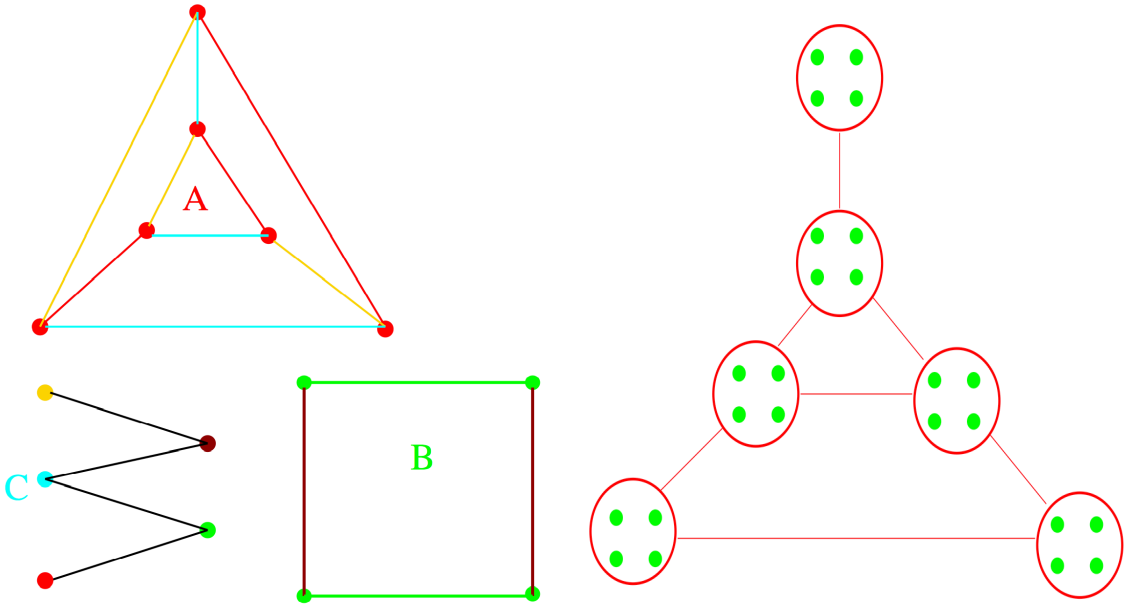


Figura 18 Expanders di ordine 17 e 59

4.2.2 Costruzione tramite prodotto tensore derandomizzato

Questo tipo di prodotto è stato creato per ottenere un compromesso fra le capacità di preservare un basso grado aumentando sensibilmente l'ordine di un grafo (proprie del prodotto a zig-zag che vedremo in seguito), e la necessità di

un'operazione chiusa per una determinata tipologia di grafi di Cayley (tipologia di grafi che possiede buone proprietà di espansione), come il prodotto tensore (o prodotto diretto) che per contro però aumenta significativamente, come già visto dall'analisi di quest'operazione, il grado. L'idea di fondo di questo prodotto dunque è quella di effettuare il prodotto diretto fra due grafi *A X B* eliminandone poi alcuni archi, seguendo la disposizione di un terzo grafo bipartito C, cosi da non intaccarne le proprietà di chiusura e di espansione diminuendone il grado. Si definisce il grafo come grafo bipartito con $d(A)$ nodi destri e $d(B)$ nodi sinistri, quest'operazione produrrà un grafo con |*A*| x |*B*| nodi ed un grado pari a $d(A)l = d(B)r$ dove l e r rappresentano rispettivamente il grado sinistro e destro del grafo bipartito C. Si osserva che se C è un grafo bipartito completo allora si ricade nel normale prodotto diretto fra grafi. L'operazione consta nel creare un grafo formato da |*A*| copie di *B*, effettuare una colorazione degli spigoli di *A* e di *B* e assegnare a ciascun nodo sinistro di *C* uno dei colori *A* e ad ogni nodo destro di C uno dei colori di *B*. Effettuata la colorazione si provvede a scomporre ciascun nodo di *A x B* in un grafo *C* al netto dei suoi spigoli procedendo a collegare tutti i "sotto-nodi" con la medesima colorazione per i quali sussista in *A* o in *B* un arco del medesimo colore. Al termine del processo sopra descritto, se fra due "sotto-nodi" è presente uno spigolo allora tale spigolo collegherà anche i nodi una volta "ricompattati".



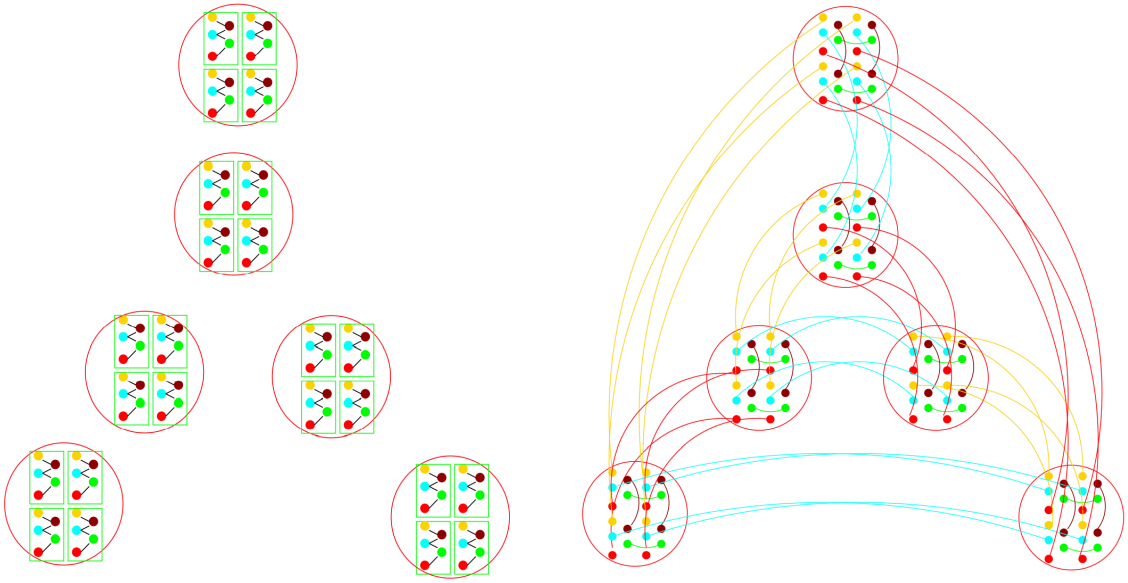


Figura 19 Procedimento per effettuare il prodotto tensore derandomizzato

L'interesse suscitato da questa operazione dalla descrizione ottenuta in letteratura, però, si allevia già dopo queste immagini. Si osserva, infatti che i grafi di Cayley a cui è rivolto questo prodotto hanno un rapporto elevato fra diametro ed ordine. Il grafo risultante erediterà poi tale diametro portando a ripercussioni negative su $\alpha(G)$ che in un grafo di grado 6 e diametro 3, con ordine 24, come quello risultante dal prodotto in figura ha $\alpha(G) = 0.128342$ mentre, un Expander costruito tramite la banale operazione precedentemente descritta ha risultati migliori su un grafo a 29 nodi con un valore di $\alpha(G) = 0.152632$, si è quindi evitata l'implementazione di questa procedura che sarebbe risultata particolarmente complessa.

4.2.3 Prodotto zig-zag

Questo prodotto porta alla formazione di multigrafi che, nel nostro caso, verranno resi grafi comuni tramite l'eliminazione degli spigoli in molteplicità maggiore di 1. Definiamo ora A come un grafo D_1-regolare con N_1 vertici, *B* come un grafo D_2-regolare con D_1 vertici e diamo una colorazione degli spigoli di *A* e *B*. Il prodotto zig-zag $A \circledast B$ è un grafo D_2-regolare con N_1D_1 vertici tali che, per ogni $v \in V_1$, $k \in D_1$, uno spigolo, etichettato con *(i, j)*, connette il

vertice *(v, k)* al vertice *(v[k[i]], k[i][j])* dove con *k[i]* si intende il vertice a cui l'i-esimo spigolo di k conduce mentre *k[i][j]* indica il vertice a cui si è destinati seguendo il j-esimo spigolo di *k[i]*. Si può considerare l'insieme di vertici come un grafo *A* in cui i nodi vengano sostituiti da grafi *B*, in questa ipotesi lo "zig-zag product" connette due vertici seguendo un percorso, per l'appunto, a zig-zag che parte dal vertice *k* del grafo *B* che sostituisce il nodo *v* di *A*, procedendo dunque, dentro al v-esimo grafo *B,* verso il nodo *k[i]* (determinato dall'etichettamento che si sceglie per lo spigolo *(i, j)*), si prosegue quindi con uno spostamento dal v-esimo grafo *B* al v[k[i]]-esimo grafo *B* (da questo spostamento si deduce l'importanza del vincolo per cui *A* deve essere D_1 regolare e *B* avere D_1 vertici) sempre nel nodo k[i]. L'ultimo passo prevede di passare, all'interno del v[k[i]]-esimo grafo *B*, dal nodo k[i]-esimo al k[i][j]-esimo.

```
ZigZagProductGraph[A_, B_] :=
  Module[{i, j, v, k, uno, due, Lista, partenza, arrivo, pos1, pos2, pos3, g, N, coloringB, coloringA},
   N = Round[(VertexCount[B])^(1/10)] + 1;
   coloringB = Coloring[B];
   coloringA = Coloring[A];
   g := Graph[{-1 ↔ -2}, VertexLabels → "Name"];
   For[i = 1, i ≤ VertexCount[A], i++,
    For[j = 1, j ≤ VertexCount[B], j++,
          g = VertexAdd[g, (((VertexList[A][[i]]) * (10^N)) + (VertexList[B][[j]]))];
     ];
   ];
   g = EdgeDelete[g, {-1 ↔ -2}];
   g = VertexDelete[g, {-1, -2}];
   For[v = 1, v ≤ VertexCount[A], v++,
    For[k = 1, k ≤ VertexCount[B], k++,
      partenza = v * (10^N) + k;
      For[i = 1, i ≤ Length[coloringB[[1]]], i++,
       pos1 = Position[VertexList[A], v][[1, 1]];
       pos2 = Position[VertexList[B], k][[1, 1]];
       If[coloringB[[pos2, i]] ≠ 0,
        pos3 = Position[VertexList[B], coloringB[[pos2, i]]][[1, 1]];
        uno = coloringA[[pos1, pos3]];

        For[j = 1, j <= Length[coloringB[[1]]], j++,
         due = coloringB[[pos3, j]];
         arrivo = uno * (10^N) + due;
         If[uno ≠ 0 && due ≠ 0 && Intersection[EdgeList[g],
              {partenza ↔ arrivo,
               arrivo ↔ partenza}] == {} && partenza ≠ arrivo,
          g = EdgeAdd[g, {partenza ↔ arrivo}];
         ];
        ];
       ];
      ];
     ];
   ];
   Return[g];
  ];
```

Codice 14 Implementazione zig-zag product

Si sottolinea che il prodotto a zig zag ha come risultato un grafo di grado massimo $(D_2)^2$. È stata inoltre necessaria l'implementazione di un'euristica per la colorazione degli spigoli richiesta da quest'algoritmo.

```
In[36]:= Coloring[A_] :=
          Module[{i, Nodo1, Nodo2, j, k, Lista, Lista1, ListaAdiacenti, pos1, pos2},
           Lista = List[];
           Lista1 = List[];
           For[j = 1, j <= GraphDegree[A], j++,
            Lista1 = Append[Lista1, 0];
           ];
           For[i = 1, i <= VertexCount[A], i++,
            Lista = Append[Lista, Lista1];
           ];
           For[i = 1, i ≤ EdgeCount[A], i++,
            Nodo1 = EdgeList[A][[i, 1]];
            Nodo2 = EdgeList[A][[i, 2]];
            pos1 = Position[VertexList[A], Nodo1][[1, 1]];
            pos2 = Position[VertexList[A], Nodo2][[1, 1]];
            k = 1;
            While[Lista[[pos1, k]] != 0 || Lista[[pos2, k]] != 0,
             k++;
             If[k > Length[Lista[[pos1]]],
              For[j = 1, j ≤ VertexCount[A], j++,
                Lista[[j]] = Append[Lista[[j]], 0];
               ];
             ];
            ];
            Lista[[pos1, k]] = Nodo2;
            Lista[[pos2, k]] = Nodo1;
           ];
           Return[Lista];
          ];
```

Codice 15 Implementazione euristica per la colorazione dei nodi di un grafo d-regolare

Essendo necessari grafi regolari per effettuare questo prodotto si è pensato di utilizzare un grafo record come K2*C5 (rinominato in figura come *A2*), ovviamente l'unico grafo regolare di ordine 3 (si ricorda che, essendo K2*C5 un grafo 3-regolare, per effettuare questo tipo di prodotto è necessario utilizzare un secondo grafo di ordine 3) è K3.

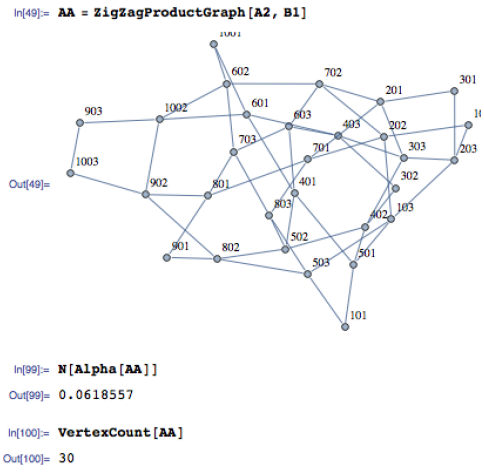


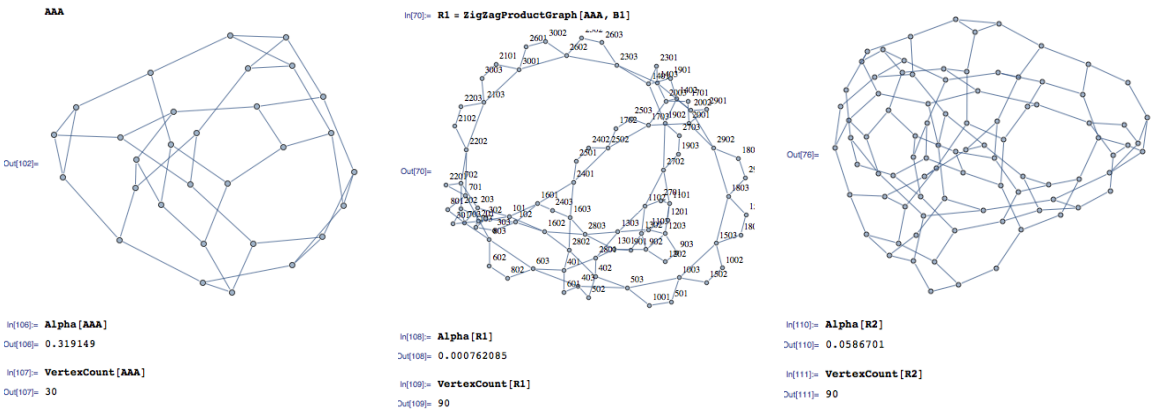
Figura 15 (K2 * C5) ⊛ K3

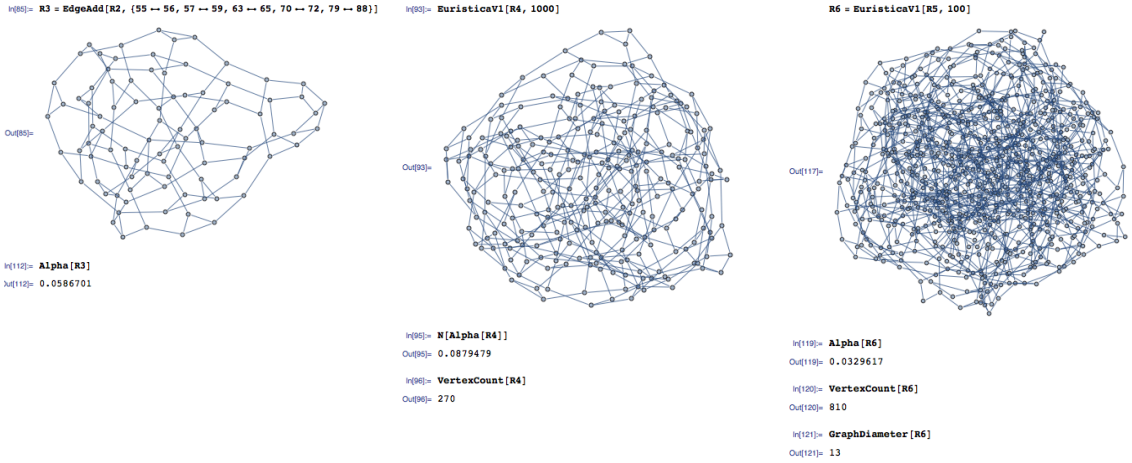
Come si può notare i risultati sono stati buoni ma migliorabili.

4.3 Conclusioni

Visti i risultati ottenuti si è provveduto ad applicare al prodotto (K2 * C5) ⊛ K3 la prima versione delle euristiche sviluppate per l'ottimizzazione dei grafi con ampi margini di miglioramento. Si è poi notato che il grafo risultante era quasi 3-regolare, si sono quindi aggiunti manualmente i nodi mancanti per ottenere la regolarità e si è effettuato nuovamente il prodotto con K3. Questo procedimento è stato reiterato diverse volte poiché i grafi ottenuti dall'elaborazione dell'euristica si sono dimostrati più volte regolari o quasi regolari.

I risultati ottenuti sono i seguenti:





Come si può osservare l'euristica definita porta grandi miglioramenti a questi grafi, nonostante siano ancora lontani dall'essere paragonabili ai record il procedimento può essere interessante. Il punto debole di questo approccio sta nel non effettuare operazioni drastiche per la riduzione del diametro mantenendo, invece, abbastanza costante il grado, con il risultato di grafi abbastanza sparsi. Anche l'ultimo grafo ottenuto è 3-regolare con l'eccezione di 12 nodi, pertanto si potrebbe continuare ad iterare questo procedimento. Essendo però lo "zig-zag product" un'operazione improntata alla creazione di Expanders non dà buoni risultati in merito al diametro che subisce un forte aumento, si ritiene quindi che sarebbe meglio procedere con operazioni più conservative in questo senso.

5. Conclusioni finali

Nonostante non si siano raggiunti risultati paragonabili ai record finora ottenuti, l'analisi effettuata ha evidenziato come gli approcci esaminati possano portare a miglioramenti settoriali, ossia inerenti ad una sola delle caratteristiche in esame. La strada più plausibile per un miglioramento complessivo di un grafo, nella direzione del "degree-diameter problem", sembra essere segnata dall'integrazione di diversi metodi. Si puntualizza, inoltre, che molte altre strade sono aperte ai fini di un'ottimizzazione dei risultati finora ottenuti, da una costruzione ex-novo di un grafo che mantenga le caratteristiche richieste, ad una generazione di un campione di grafi casuali che sottostiano a determinati vincoli. Un'altra possibile direzione, poi, è utilizzare i grafi record rinvenuti finora come punto di partenza per modifiche che portino ad un miglioramento ulteriore. Le strade analizzate sono state scelte per l'interesse personale verso la tipologia di approccio a seguito di un approfondito studio del problema e delle soluzioni finora proposte. In particolare è risultato interessante misurarsi con la creazione di un'euristica di ottimizzazione: nonostante abbiano diverse carenze, le euristiche sviluppate, hanno sondato e utilizzato diversi approcci studiati nel corso degli studi effettuati e possono essere considerate come un punto di partenza per una successiva raffinazione. Si vuole inoltre porre l'accento su come, per tali euristiche, si sia proceduto in maniera autonoma, evitando di utilizzare risultati già ottenuti da terzi. Si è fatta questa scelta nella speranza di giungere, se non ad una svolta, ad un nuovo metodo per ottenere risultati interessanti.

6. Riferimenti bibliografici

- Bibliografia

[1] Béla Bollobás, 2001. *Random Graphs* 2. ed. - Cambridge : Cambridge university press.

[2] Eyal Loz Jozef S̆ira ́n̆, 2008. New record graphs in the degree-diameter problem, *Australasian journal of combinatorics* [online], Department of Mathematics University of Auckland, Volume 41, Pages 63–80. Disponibile su: <http://ajc.maths.uq.edu.au/pdf/41/ajc_v41_p063.pdf> [Data accesso: 12/12/2012]

[3] Byeong Moon Kim, Byung Chul Song e Woonjae Hwang, November 2011. Exponents and diameters of strong products of digraphs, *Electronic Journal of Linear Algebra* [online], ISSN 1081-3810, International Linear Algebra Society Volume 22, pp. 1106-1111. Disponibile su: <http://www.math.technion.ac.il/iic/ela/ela-articles/articles/vol22_pp1106-1111.pdf> [Data di accesso: 14/12/2012].

[4] Paul A. Catlin, Arthur M. Hobbs, Hong-Jian Lai, 2001. Graph family operations. *Discrete Mathematics* [online], 230, 71–97. Disponibile su: <http://www.google.it/url?sa=t&rct=j&q=&esrc=s&source=web&cd=1&cad=rja&ved=0CC8QFjAA&url=http%3A%2F%2Fciteseerx.ist.psu.edu%2Fviewdoc%2Fdownload%3Fdoi%3D10.1.1.98.5616%26rep%3Drep1%26type%3Dpdf&ei=phpGUeWaF9KN4gTi_YDoDg&usg=AFQjCNHx78uQ6NszDBU5JHyalO9DA1kSZA> [Data di accesso: 14/12/2012].

[5] P. R. Hafner, June 1995. Large Cayley Graphs and Digraphs with Small Degree and Diameter [online] Department of Mathematics University of Auckland CDMTCS-005. Disponibile su: <http://www.cs.auckland.ac.nz/CDMTCS/researchreports/004cayley.pdf> [Data di accesso: 09/01/2013].

[6] C. Bermond, C. Delorme, e G. Farhi, 1984. Large graphs with given degree and diameter ii. *Journal of combinatorial theory* [online], Series B 36, 32-48. Disponibile su: <http://www.sciencedirect.com/science/article/pii/0095895684900121> [Data di accesso:09/01/2013].

[7] J. Ćaceres, C. Hernando, M. Mora, I.M. Pelayo, and M.L. Puertas. *Boundary-type sets and product operators in graphs* [online]. Disponibile su: <http://upcommons.upc.edu/e-prints/bitstream/2117/8383/1/Boundary.pdf> [Data accesso: 10/01/2013].

[8] Justin Cranshaw e Hila Becker, December 23, 2004 . Expander Graphs and the zig-zag Product, COMS 4995: *Introduction to Coding Theory* [online]. Disponibile su: <http://www.cs.columbia.edu/~hila/expanders.pdf> [Data di accesso: 14/01/2013]

[9] Fan Chung, Linyuan Lu. *The Diameter of Sparse Random Graphs* [online]. Disponibile su: <http://math.ucsd.edu/~fan/dia.pdf> [Data di accesso: 12/01/2013].

[10] Andrew Brown. *Some graph products and their expansion properties* Information Theory Workshop, 2006. ITW '06 Punta del Este. IEEE [online]. Disponibile su: <http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1633804&tag=1> [Data di accesso: 01/02/2013]

[11] Omer Reingold Salil Vadhan Avi Wigderson. August 1, 2001. *Entropy Waves, The Zig-Zag Graph Product, and*

New Constant-Degree Expanders [online]. Disponibile su: <http://www.math.ias.edu/~avi/PUBLICATIONS/MYPAPERS/SALIL/EXPANDERS/ANNALS/RVW01.pdf> [Data di accesso: 04/02/2013]

- Sitografia

[1] <http://en.wikipedia.org/wiki/Graph_theory>

[2] <http://en.wikipedia.org/wiki/Butterfly_graph>

[3] <http://en.wikipedia.org/wiki/Graph_product>

[4] <http://mathworld.wolfram.com/Graph.html>

[5] <http://mathworld.wolfram.com/GraphProduct.html>

[6] <http://mathworld.wolfram.com/ButterflyGraph.html>

[7]<http://mathworld.wolfram.com/deBruijnGraph.html>

[8] <http://www-mat.upc.es/grup_de_grafs/table_g.html/>

7. Ringraziamenti

Un ringraziamento dovuto, oltre che sentito, va al relatore di questa tesi, il Professor Margara che, oltre avermi assistito in questa tesi, mi ha trasmesso la forte passione per la sua materia.

Un abbraccio va, poi, ai parenti che mi hanno sostenuto e appoggiato durante tutto il mio percorso di studi ed agli amici, sempre presenti e disposti a sopportarmi.