UNIVERSITÀ DEGLI STUDI ROMA TRE
FACOLTÀ DI SCIENZE M.F.N.
CORSO DI LAUREA IN MATEMATICA

Tesi di Laurea in Matematica

Algoritmi efficienti per la soluzione del problema Clique

Candidato
Matteo Acclavio

Relatore
Prof. M. Liverani

ANNO ACCADEMICO 2009-2010
Febbraio 2011

Classificazione AMS: 05C69, 05C85, 68R10, 68W40.

Parole chiave: Algoritmi su grafi, Teoria dei grafi, Clique.

Indice

Introduzione

> "... ma guarda che evidentemente abbiamo degli amici in comune, perché altrimenti non me lo diceva... Che so', tu hai fatto forse, il liceo al Mamiani? Vai in vacanza a Trevignano? Hai lavorato in sala scocche forse? ... No? Una fonderia a Terni? No scusa, scusate, scusate, scusa; io assolutamente non voglio essere invadente, però, io devo risolvere questo problema sennò stanotte non ci dormo! ..."
>
> Susanna la ragazza Facebook (C. Guzzanti)

Al di là dei dilemmi sociali di ogni giorno, fra i problemi classici della teoria della complessità computazionale un ruolo importante è ricoperto dal così detto problema della *clique*. Il termine *clique* (in italiano "cricca") sta proprio ad indicare un gruppo ristretto di elementi strettamente legati tra di loro da relazioni o caratteristiche in comune. Ad esempio possiamo pensare a gruppi di persone con interessi o amicizie comuni. In teoria dei grafi una clique è un sottografo in cui, per ogni coppia di nodi, esiste un collegamento tra essi; in termini più esatti possiamo dire che una clique di un grafo G è un suo sottografo completo massimale. Lo studio delle clique di un grafo permette di capire molte caratteristiche sulla natura del grafo studiato.

L'aspetto interessante della questione, che ne fa un tema che ha suscitato molto interesse negli anni nell'ambito della comunità scientifica, è che risolvere il problema della clique non è semplice come potrebbe sembrare! Anzi, trovarne una soluzione, oltre che fruttare un milione di dollari[1], rivoluzionerebbe l'informatica

[1] Se si riuscisse ad escogitare un algoritmo di complessità polinomiale, in grado di risolvere il problema CLIQUE, si sarebbe trovata anche una risposta (affermativa) al problema "P = NP?", su cui "pende una taglia" da un milione di dollari, appunto!

anche nei suoi aspetti applicativi del quotidiano, con ricadute importanti su questioni legate alla crittografia e alla sicurezza delle informazioni. Il motivo per cui questa tesi si spinge a trattare un problema $\mathcal{NP}$-$\mathcal{C}$ è principalmente dovuto alla curiosità di verificare l'efficienza di un algoritmo ideato durante pomeriggio rubato a studi più "universitari" (facendomi prendere la mano, nel ragionare su un problema così elementare eppure così difficile da risolvere, mi è sembrato talvolta di risolvere un rompicapo, più che portare avanti i miei studi universitari). Risultando inizialmente difficile calcolare la complessità computazionale, parlando con Alexander Gaudillièr, allora assegnista di ricerca a Roma Tre, mi fu suggerito, per verificare la presunta efficienza, di utilizzare il metodo da lui presentato come "corsa dei cavalli": si fanno affrontare le stesse istanze del problema ad algoritmi diversi, verificando così, in maniera sperimentale, quale impiega meno tempo per risolvere il medesimo problema. Naturalmente i risultati ottenuti saranno falsati dall'abilità del programmatore, ma inserendo anche degli opportuni contatori si può avere comunque una stima del numero di operazioni effettuate per risolvere un'istanza del problema di dimensione n.

Dal tentato studio della complessità computazionale mi sono imbattuto in un problema di combinatoria: quali sono le partizioni $\mathfrak{p}$ di un intero che possono descrivere la distribuzione degli spigoli in un grafo? Anche questo problema non sembra di facile soluzione e lascia molto spazio a ricerche su quante altre informazioni, oltre $\mathfrak{p}$, bisogna avere per individuare univocamente un grafo. Alla luce di ciò affrontare il problema in modo sperimentale è apparso più semplice. Grazie al consiglio della prof.ssa Elisabetta Scoppola, dalla quale ho ricevuto informazioni e spiegazioni su alcuni algoritmi probabilistici [1], sono entrato in contatto con il dott. Massimiliano Viale che, con pazienza, mi ha dato un aiuto nell'impostare la programmazione dell'algoritmo per le gare di velocità. Inoltre, essendosi occupato di una sperimentazione analoga nella sua tesi di dottorato [2], mi ha potuto anche dare i programmi di alcuni algoritmi euristici di ricerca della clique con i quali sono stati raccolti i dati sperimentali.

Questa tesi è strutturata su tre capitoli, seguiti da tre appendici. Nel primo capitolo, dopo aver definito cos'è un grafo e alcune delle sue principali proprietà,

viene analizzato il problema della ricerca di una clique dal punto di vista della complessità computazionale. Nel secondo capitolo vengono analizzati alcuni algoritmi finalizzati alla ricerca di clique e le strategie usate, poi viene illustrato l'algoritmo su cui sono state effettuate le ricerche presenti nel terzo capitolo (integrate nell'appendice C). Nelle appendici sono riportati il sorgente del programma implementato in linguaggio C, una breve spiegazione sul formato DIMACS [3] per la rappresentazione di grafi e nell'appendice C un insieme di esempi utili per avere un'intuizione del problema di combinatoria che lega grafi e partizioni.

Capitolo 1

Richiami di teoria dei grafi

In questo capitolo vengono richiamate alcune definizioni di base e alcuni semplici teoremi della teoria dei grafi necessari per comprendere cosa sia una clique, cercando poi di capire il perché della "difficoltà" nel ricercarla; il problema viene analizzato dal punto di vista algoritmico come un problema di ottimizzazione combinatoria.

1.1 Cenni sui grafi

Nelle pagine seguenti indicheremo con $[n]$ l'insieme $\{1,2,3,\ldots,n\}$; se A è un insieme, indicheremo inoltre con $\binom{A}{2}$ l'insieme costituito da tutte le coppie di elementi di A.

Definizione 1 *Un* grafo *(o* grafo semplice*) è una coppia $G=(V,E)$ di insiemi tali che $V \neq \emptyset$ ed E sia un sottoinsieme di $\binom{V}{2}$. Gli elementi di V sono chiamati* vertici *(o* nodi*) del grafo G, mentre gli elementi di E sono detti* spigoli *del grafo. L'insieme dei vertici del grafo G si indica con la notazione $V(G)$ e, analogamente, l'insieme degli spigoli sarà indicato con $E(G)$. La cardinalità di un grafo è il numero dei suoi vertici $|G| = |V(G)|$, mentre il numero dei sui spigoli lo indicheremo con $||G|| = |E(G)|$.*

Notazione Per comodità indicheremo con (a,b) (o equivalentemente(b,a)) lo spigolo $e = \{a,b\}$. Scriveremo inoltre che $x \in e$ con $x \in V(G)$ ed $e = (a,b) \in E(G)$ se $x = a \vee x = b$.

Definizione 2 *Due vertici $x,y \in V(G)$ si dicono* adiacenti *se esiste lo spigolo $(x,y) \in E(G)$. Indicheremo l'insieme dei vertici adiacenti a un sottoinsieme S di $V(G)$ con $N_G(S) = \{x \in V(G) \setminus S \text{ t.c. } (x,y) \in E(G) \text{ per qualche } y \in S\}$ mentre $N_G[S] = N_G(S) \cup S$.*

Notazione Nel caso in cui non ci sia ambiguità sul grafo considerato indicheremo semplicemente $N[x]$ il vicinato di x.

Definizione 3 *Dato un vertice $x \in G$ il grado $\delta_G(x)$ è il numero di vertici adiacenti a x nel grafo G: $\delta_G(x) = |N_G(\{x\})|$.*

Lemma 1 (Della stretta di mano) *Sia G un grafo finito ($|G| < \infty$), allora*

$$\sum_{x \in V(G)} \delta(x) = 2|E(G)|$$

Dimostrazione Contando in due modi differenti le coppie (spigolo–vertice in incidenza), per ogni vertice x ci sono $\delta(x)$ spigoli mentre per ogni spigolo ci sono due vertici. Ne segue la tesi. ■

Definizione 4 *Un grafo G è k-*ridotto *se $\forall x \in G,\ \delta_G(x) \geq k-1$.*

Definizione 5 *Un grafo S_n con $|V(S_n)| = n+1 < \infty$ è una* stella *se esiste $x \in V(S_n)$ detto* centro*, tale che $E(S_n) = \{(x,y), \forall y \in V(S_n) \setminus \{x\}\}$; diremo che un sottoinsieme $S \subset V(G)$ è* stellato *in G se $\forall v \in V(G) \setminus S$ e $\forall s \in S$ risulta $(v,s) \in E(G)$.*

Definizione 6 *Sia G un grafo; il* grafo complementare *G^C è definito ponendo $V(G^C) := V(G)$ e $E(G^C) := \binom{V(G)}{2} \setminus E(G)$.*

Definizione 7 *Un sottografo H di G è un grafo tale che: $V(H) \subseteq V(G)$ e $E(H) \subseteq E(G) \cap \binom{V(H)}{2}$.*

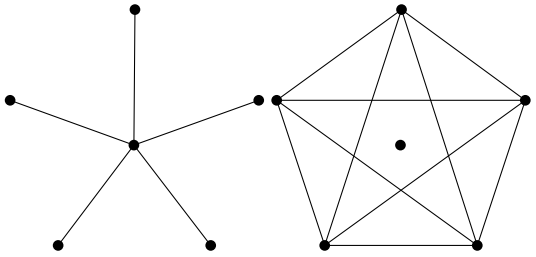


Figura 1.1: Il grafo stella S_5 e il suo complementare

Notazione Sia $S \subseteq V(G)$; indicheremo con G_S il *sottografo indotto* dal sottoinsieme di vertici S: $G_S = (S, E(S))$ t.c. $E(S) = E(G) \cap \binom{S}{2}$.

Proposizione 2 *Sia H sottografo di G; allora $\forall x \in G,\ \delta_H(x) \leq \delta_G(x)$*

Definizione 8 (Eliminazione di un vertice) *Sia $x \in V(G)$, $G - x = (V', E')$ dove $V' = V \setminus \{x\}$ e $E' = E \cap \binom{V'}{2}$. Analogamente, dato $S \subseteq V(G)$, il grafo ottenuto da G eliminando i vertici del sottoinsieme S è il grafo $G - S = (((G - s_1) - s_2) \dots) - s_n$ dove gli $s_i \in S$ sono tutti gli elementi di S.*

Definizione 9 (Isomorfismo tra grafi) *Due grafi G e G' si dicono* isomorfi *se esiste una corrispondenza biunivoca $\psi : V(G) \to V(G')$ tale che per ogni coppia $(x,y) \in \binom{V(G)}{2}$ risulta $(x,y) \in E(G) \Leftrightarrow (\psi(x), \psi(y)) \in E(G')$.*

Osservazione È facile dimostrare che se $|G| = n$ allora G è isomorfo a un grafo G' con $V(G') = [n]$ dove $[n] := \{1, 2, \dots, n\}$.

Definizione 10 *Sia V un insieme di cardinalità n: il* grafo nullo *su V è il grafo $(V, \emptyset)$; il* grafo completo *su V è il grafo $(V, \binom{V}{2})$.*

Notazione Indicheremo con K_n e N_n rispettivamente il grafo completo e il grafo nullo su $[n]$, mentre diremo che un sottografo è un K_n o un N_n se è isomorfo a uno di questi.

Proposizione 3 *Sia $|G| = n$ allora è facile verificare dalle definizioni precedentemente date che:*

- *G è isomorfo ad un sottografo di* $K_m \; \forall \; m \geq n$
- N_m *è isomorfo ad un sottografo di* $G \; \forall \; m \leq n$

Notazione Siano G e G' due grafi; allora il *grafo unione* di G e G' è dato da $G \cup G' := (V(G) \cup V(G'), E(G) \cup E(G'))$.

Definizione 11 *In un grafo una clique C è un sottografo completo di G, cioè tale che* $\forall x, y \in V(C)$ *risulta* $e = (x,y) \in E(G)$*, o equivalentemente* $\forall x \in V(C)$ $, N[x] = V(C)$.

Proposizione 4 *Sia C una clique di un grafo G; C è isomorfa a* $K_{|C|}$.

Definizione 12 *Il* clique number $Cl(G)$ *di G è il più grande intero k tale che esiste in G una clique di dimensione k.*

Proposizione 5 *Per ogni* $k \leq Cl(G)$ *esiste una clique di dimensione k.*

Dimostrazione Sia C una clique di massima cardinalità $k = Cl(G)$ sul grafo G. Preso un qualunque sottoinsieme di $U \subseteq V(C)$, risulta che il grafo indotto da U, G_U, è completo. ■

Definizione 13 (Cammino) *Siano* $x, y \in V(G)$ *due vertici del grafo G. Un* cammino semplice *p da x a y su G è una sequenza* $x = x_1, \ldots, x_n = y$ *di vertici in G tali che* $(x_i, x_{i+1}) \in E(G) \; \forall i = 1, \ldots, n-1$ *e* $x_i \neq x_j \; \forall i \neq j$ *escludendo al più il caso* $x = y$*. Il cammino sarà indicato con* $p : x \rightsquigarrow y$.

Definizione 14 $C_k = ([k], \{(i, i+1) \text{ per } i = 1, \ldots, k-1\})$ *è il grafo cammino di lunghezza k.*

Definizione 15 *Un grafo G si dice connesso se per ogni coppia di vertici* $x, y \in V(G)$*,* $x \neq y$*, esiste un cammino* $p : x \rightsquigarrow y$ *che li collega.*

Osservazione Sia C una clique di G; C è un grafo connesso.

Definizione 16 *Sia G un grafo; la* matrice di adiacenza *di G è la matrice* $A_G = (a_{ij})$ *con*

$$a_{ij} = \begin{cases} 1, & se \; (i,j) \in E(G) \\ 0, & se \; (i,j) \notin E(G) \end{cases}$$

1.2 Complessità computazionale di algoritmi e problemi

La teoria della complessità computazionale si occupa di calcolare le risorse minime necessarie per la risoluzione di un problema mediante un algoritmo $\mathcal{A}$ e confrontare il comportamento di diversi algoritmi su uno stesso problema.

Senza entrare troppo nello specifico possiamo definire un algoritmo $\mathcal{A}$ come una sequenza finita di operazioni finalizzate alla risoluzione di un problema; ciascuna operazione della sequenza deve essere un elementare e non ambigua; deve essere univocamente determinato quale operazione debba essere eseguita a ciascun passo dell'algoritmo; inoltre si richiede che l'algoritmo termini, determinando la soluzione dell'istanza del problema, dopo aver eseguito un numero finito di operazioni.

Dato un problema $\mathcal{P}$, un'*istanza* I di $\mathcal{P}$ viene identificata dall'insieme di informazioni che sono necessarie per definirla. Tali informazioni sono i dati che, forniti ad un algoritmo risolutore $\mathcal{A}$ consentono a tale algoritmo di risolvere l'istanza I del problema $\mathcal{P}$. Il numero di informazioni con cui viene identificata un'istanza del problema viene chiamato, in gergo, la *dimensione* del problema.

Il numero di operazioni t necessarie per risolvere un problema dipenderà sia dall'istanza I che dall'algoritmo $\mathcal{A}$ scelto, cioè $t = f_{\mathcal{A}}(I)$. Tale funzione $f_{\mathcal{A}}(I)$ è chiamata *complessità computazionale* dell'algoritmo $\mathcal{A}$ sull'istanza I. Tale funzione, vista la finitezza del numero di operazioni di un algoritmo, è sempre definita al variare della cardinalità di I. Per poter quindi definire in modo più operativo tale funzione e poter confrontare l'efficienza di diversi algoritmi per la soluzione di uno stesso problema, si considera sempre il caso peggiore per ognuno di esse. Inoltre per evitare di considerare fattori poco rilevanti, si considera unicamente l'ordine di grandezza confrontando quindi solo i comportamenti asintotici al crescere di $|I|$.

Definizione 17 *Data $f(n) : \mathbb{R} \to \mathbb{R}$ si definiscono i seguenti insiemi di funzioni:*

$$\begin{aligned} O(f(n)) &= \{g(n) : \exists\, c > 0,\, n_0 > 0 \text{ t.c.} 0 \leq g(n) \leq cf(n), \forall n > n_0\} \\ \Omega(f(n)) &= \{g(n) : \exists\, c > 0,\, n_0 > 0 \text{ t.c. } g(n) > cf(n), \forall n > n_0\} \\ \Theta(f(n)) &= O(f(n)) \cap \Omega(f(n)) \end{aligned}$$

Definizione 18 *Sia $\mathcal{A}$ un algoritmo, I l'istanza che rappresenta un caso tra i più sfavorevoli, $n = |I|$ allora se $f_{\mathcal{A}}(I) \in O(f(n))$ diremo che $\mathcal{A} \in O(f(n))$.*

La complessità computazionale di un problema è quindi definita come la complessità del miglior algoritmo risolutivo. Definita la complessità computazionale di un problema possiamo individuare le seguenti famiglie di problemi:

- $\mathcal{P}$: la famiglia dei problemi che ammettono un algoritmo risolutivo di complessità computazionale polinomiale ($\mathcal{A} \in O(n^k) \exists k \in \mathbb{N}$, $\mathcal{A}$ algoritmo risolutivo del problema);
- $\mathcal{NP}$: la famiglia di problemi che ammettono un algoritmo di verifica di tipo polinomiale (data una presunta soluzione del problema, è possibile stabilire se questa sia realmente una soluzione con un algoritmo di complessità polinomiale).

Proposizione 6 $\mathcal{P} \subseteq \mathcal{NP}$.

Dimostrazione Sia σ una ipotetica soluzione del problema $\mathcal{B}$ sull'istanza α. Se esiste un algoritmo $\mathcal{A}$ in grado di calcolare la soluzione di α in tempo polinomiale, allora l'algoritmo di verifica del problema consiste nel calcolare la soluzione γ di α tramite $\mathcal{A}$ e controllare se $\gamma = \sigma$. ■

Definizione 19 *Un problema $\mathcal{A}$ si dice* riducibile *ad un problema $\mathcal{B}$ se $\forall \alpha$ istanza di $\mathcal{A}$ è possibile associare un'istanza $\beta = \mathcal{T}(\alpha)$ di $\mathcal{B}$ tale che la soluzione calcolata per β sia soluzione anche per α. Se quest'operazione può essere compiuta da un algoritmo $\mathcal{T}$ in tempo polinomiale allora si dirà che il problema $\mathcal{A}$ è* riducibile in tempo polinomiale *al problema $\mathcal{B}$ (in simboli: $\mathcal{A} \leq_P \mathcal{B}$).*

Una particolare sottofamiglia di $\mathcal{NP}$ è quella che viene chiamata la classe dei problemi $\mathcal{NP}$-completi (indicata anche con $\mathcal{NP}$-$\mathcal{C}$) definita in modo seguente.

Definizione 20 *$\mathcal{A}$ è $\mathcal{NP}$-$\mathcal{C}$ se:*

- $\mathcal{A} \in \mathcal{NP}$;

- $\forall \mathcal{B} \in \mathcal{NP}\ ,\ \mathcal{B} \leq_{\mathcal{P}} \mathcal{A}$.

Questa proprietà sicuramente isola una famiglia in $\mathcal{NP}$, ma a priori non sappiamo se sia vuota o meno. Il teorema di Cook ci assicura l'esistenza di problemi $\mathcal{NP}$-$\mathcal{C}$.

Definizione 21 (Problema SAT) *Dato un insieme di variabili booleane $u_i \in U$ e un insieme di relazioni C tra queste variabili, esiste una scelta delle u_i tale che $\forall c \in C$, c sia vera?*

Teorema 7 (Cook) *Il problema* SAT *è $\mathcal{NP}$-$\mathcal{C}$. [4]*

Possiamo quindi concludere che

Teorema 8 *Siano $\mathcal{A}$ e $\mathcal{B}$ due problemi allora:*

1. *se $\mathcal{A} \leq_{\mathcal{P}} \mathcal{B}$ e $\mathcal{B} \in \mathcal{P}$ allora anche $\mathcal{A} \in \mathcal{P}$;*
2. *se esiste un problema $\mathcal{A} \in \mathcal{NP}$-$\mathcal{C}$ tale che $\mathcal{A} \in \mathcal{P}$ allora $\mathcal{NP}$-$\mathcal{C} = \mathcal{P}$.*

Dimostrazione Esistendo un algoritmo polinomiale $\mathcal{T}$ per trasformare le istanze di $\mathcal{A}$ in istanze di $\mathcal{B}$ e un algoritmo polinomiale per la risoluzione delle istanze di $\mathcal{B}$ allora $\forall \alpha$ istanza di $\mathcal{A}$, $f_{\mathcal{A}}(\alpha) = f_{\mathcal{B}}(\mathcal{T}(\alpha)) \cdot f_{\mathcal{T}}(\alpha) = f_{\mathcal{B}}(\beta) \cdot f_{\mathcal{T}}(\alpha)$ che sappiamo essere polinomiale perché prodotto di espressioni polinomiali.

$\mathcal{A}$ è $\mathcal{NP}$-$\mathcal{C} \Rightarrow \forall \mathcal{B} \in \mathcal{NP},\ \mathcal{B} \leq_{\mathcal{P}} \mathcal{A}$ per definizione. Supponendo che esista un problema $\mathcal{A}$ $\mathcal{NP}$-$\mathcal{C}$ che ammetta algoritmo di risoluzione polinomiale segue dal punto precedente che $\forall \mathcal{B} \in \mathcal{NP},\ \mathcal{B} \in \mathcal{P}$. ■

Il teorema appena dimostrato ci dice sostanzialmente che, se si scoprisse un algoritmo polinomiale per calcolare la soluzione di un qualsiasi problema $\mathcal{NP}$-$\mathcal{C}$, allora tutta la classe dei problemi $\mathcal{NP}$ collasserebbe in $\mathcal{P}$, cioè $\mathcal{P} = \mathcal{NP}$. Altri problemi $\mathcal{NP}$-$\mathcal{C}$ noti sono:

Definizione 22 (3SAT) *Dato un insieme di variabili Booleane $u_i \in U$ e un insieme di relazioni C tra 3 di queste variabili, esiste una scelta delle u_i t.c. $\forall c \in C$, c sia vera?*

Definizione 23 (VERTEXCOVER) *Dato grafo G e un intero $k < |G|$ esiste un sottoinsieme $U \subseteq V(G)$ t.c. $|U| = k$ e che il suo vicinato $N[U]$ sia tutto $V(G)$?*

Definizione 24 (HAMILTONIANCIRCUIT) *Dato grafo G esiste un cammino Hamiltoniano sul grafo G? In altri termini: esiste un cammino p da x a x, $p = (x = x_1, x_2, \ldots, x_{n-1}, x_n = x_1 = x)$, tale che $x_i \neq x_j$ per ogni $i \neq j$, $i, j = 1, 2, \ldots, n-1$ e $V(G) = \{x_1, \ldots, x_{n-1}\}$?*

Definizione 25 (CLIQUE) *Dato grafo G qual è il suo clique number $Cl(G)$?*

1.3 Il problema della clique

Dato un grafo G e un intero $k > 0$, esiste in G
una clique di dimensione maggiore o uguale a k?

La risposta a questa domanda (problema k-CLIQUE) è meno banale di quanto sembri:

Teorema 9 *Il problema k-CLIQUE è $\mathcal{NP}$-$\mathcal{C}$*

Dimostrazione Il problema VERTEXCOVER con istanza il grafo G e l'intero $|G| - k$ è riducibile in tempo polinomiale al problema k-CLIQUE sul grafo G^C.

Sia U una copertura di vertici di G^C, $|U| = k$ se $e = (x,y) \notin E(G) \Rightarrow x \in U \vee y \in U$ ne segue che $x \notin U \wedge y \notin U \in\Rightarrow e = (x,y) \in E(G)$ e quindi $\forall x, y \in V(G) \setminus U \; \exists e = (x,y) \in E(G)$ cioè la tesi che $V(G) \setminus U$ sono i vertici di una clique in G di dimensione $|G| - k$.

Naturalmente è valido anche il viceversa: sia C una clique di G di dimensione $k \Rightarrow U = V(G) \setminus C$ è una copertura di vertici di G^C infatti $\forall e = (x,y) \in E(G^C)$, $x \notin C \vee y \notin C$ altrimenti e sarebbe un vertice della clique C quindi $e \in E(G)$.

Una dimostrazione più immediata si ha sapendo che CLIQUE è $\mathcal{NP}$-$\mathcal{C}$ e può essere risolto iterando k-CLIQUE al più $|G|$ volte. ■

Gli sbocchi applicativi della ricerca delle clique su un grafo sono applicabili in vari ambiti che fanno uso di schematizzazioni tramite grafi:

- Progettazione reti

- Studio social network
- Risoluzione altri problemi relativi ai grafi (numero cromatico, *hardness*)
- Studio di modelli fisici (metastabilità, teoria di percolazione)

Naturalmente, a seconda del tipo di applicazioni, non è strettamente necessario fornire una soluzione esatta della dimensione della clique massima, ma è possibile accontentarsi di soluzioni approssimate, molto più semplici da calcolare.

Capitolo 2

Algoritmi di ricerca

Vista l'appartenenza del problema della clique alla classe $\mathcal{NP}$-$\mathcal{C}$ i vari algoritmi di ricerca che ne affrontano la soluzione possono essere suddivisi in due famiglie:

- Algoritmi di ricerca "esaustiva", che producono soluzioni esatte in tempi elevati; tali algoritmi sostanzialmente procedono per intersezioni successive di "vicinati": si parte da un vertice e si considera l'insieme dei vertici ad esso adiacenti (il suo *vicinato*), si seleziona un altro vertice nel grafo indotto (le modalità della scelta caratterizzano i vari algoritmi), si considera il vicinato e si procede iterando questo procedimento finché si possono continuare a selezionare nuovi vertici; quando si arriva a k vertici selezionati allora avremo una soluzione.

- Algoritmi euristici basati su modelli fisico-probabilistici, che cercano di esplorare nella maniera più "astuta" il minor numero di possibile di sottoinsiemi di $V(G)$, avvicinandosi il più possibile alla soluzione, senza però avere mai la certezza dell'esattezza della stessa.

Andremo quindi ad analizzare alcuni dei più comuni algoritmi di ricerca per il problema k-CLIQUE e l'algoritmo CLIQUER per la ricerca della clique massima, vista la sua particolare efficienza.

2.1 Algoritmi esatti

2.1.1 Ricerca esaustiva

Il primo algoritmo (RICERCAESAUSTIVA) che andremo ad analizzare applica quello che viene definito metodo "a forza bruta", cioè procede con un controllo esaustivo su tutti i $\binom{|G|}{k}$ possibili sottoinsiemi di $V(G)$ di cardinalità k.

Algoritmo 1 RICERCAESAUSTIVA(G,k)

```
1: Sia 𝒰 := {sottoinsiemi di V(G) di cardinalità k}
2: fintanto che non trovo una clique ripeti
3:     per ogni U_i ∈ 𝒰 ripeti
4:         verifica se U_i è una clique
5:     fine-ciclo
6: fine-ciclo
```

Proposizione 10 RICERCAESAUSTIVA $\in O(n^k k^2)$.

Dimostrazione L'algoritmo analizza tutti i $\binom{|G|}{k} \in O(n^k)$ sottoinsiemi, verificando l'esistenza di tutti i $\binom{k}{2} \in O(k^2)$ spigoli. ■

2.1.2 Cliquer

L'algoritmo CLIQUER [7] basa la strategia risolutiva su un attento ordinamento dei vertici per poi operare una selezione dei vertici per cercare di ingrandire la dimensione della clique massima. Considerando sottoinsiemi sempre più grandi, $S_i = \{v_i, \ldots, v_n\} \subseteq V(G)$, verifica preliminarmente se, nell'insieme che sta considerando, può essere contenuta una clique più grande di quella massima trovata fino a quel momento. La verifica avviene sfruttando tutti i dati sulle clique che vengono via via individuate.

L'algoritmo è suddiviso in due procedure distinte: CLIQUER richiama, iterativamente, la funzione ricorsiva CLIQUE; la pseudo-codifica delle due procedure è riportata di seguito.

Algoritmo 2 CLIQUER(G)

1: $\max = 0$
2: **per ogni** $i = 1, \ldots, n = |G|$ **ripeti**
3: $\quad S_i = \{s_i, \ldots, s_n\}$
4: **fine-ciclo**
5: **per ogni** $i = n, n-1, n-2, \ldots, 1$ **ripeti**
6: $\quad found = false$
7: $\quad$ CLIQUE$(S_i \cap N[v_i], 1)$
8: $\quad c[i] = \max$
9: **fine-ciclo**

Algoritmo 3 CLIQUE$(U, size)$

1: Sia $C = V(G)$
2: **se** $|U| = 0$ **allora**
3: $\quad$ **se** $size > \max$ **allora**
4: $\quad\quad \max = size$
5: $\quad\quad$ salva U come clique
6: $\quad\quad found = true$
7: $\quad$ **fine-condizione**
8: $\quad$ return
9: **fine-condizione**
10: **fintanto che** $U \neq \emptyset$ **ripeti**
11: $\quad$ **se** $size + |U| \leq \max$ **allora**
12: $\quad\quad$ return
13: $\quad$ **fine-condizione**
14: $\quad i = \min\{j$ tale che $v_j \in U\}$
15: $\quad$ **se** $size + c[i] \leq \max$ **allora**
16: $\quad\quad$ return
17: $\quad$ **fine-condizione**
18: $\quad U = U \setminus \{v_i\}$
19: $\quad$ CLIQUE$(U \cap N[v_i], size)$
20: $\quad$ **se** $found = true$ **allora**
21: $\quad\quad$ return
22: $\quad$ **fine-condizione**
23: **fine-ciclo**
24: return

Come si può notare, per ogni $i \in [n]$, $c[i] = c[i+1]$ oppure $c[i] = c[i+1]+1$. Questo significa che quando vado a considerare la dimensione della clique contente anche il vertice v_j con $j > i$, questa avrà al più dimensione $c[j]+k$ dove k è la "profondità" in cui si trova l'algoritmo (il numero di chiamate di CLIQUE), che corrisponde alla cardinalità della clique che stiamo cercando di ingrandire.

2.2 Algoritmi euristici

Gli algoritmi presentati in questa sezione cercano di esplorare nella maniera più efficiente il minor numero di possibile di sottoinsiemi di $V(G)$, avvicinandosi il più possibile alla soluzione, senza però garantire la certezza dell'esattezza della stessa: più spesso la soluzione individuata è una buona approssimazione della soluzione ottima.

2.2.1 Algoritmo Greedy

L'algortimo GREEDY ricerca la clique per intersezione di vicinati successivi. Partendo dal grafo iniziale, seleziona arbitrariamente un vertice v, imposta $U = \{v\}$, calcola il grafo indotto $G_{N[U]}$ e prosegue la ricerca selezionando gli altri vertici da aggiungere ad U in $N(U)$.

L'algoritmo termina dopo aver selezionato al più $|G|$ vertici. Inoltre sappiamo anche che il grafo finale G_{U_f} è completo perché, per ogni $c_i < c_j \in U_f$, esiste (c_i, c_j), dove l'ordine dei vertici in U_f è dato dall'ordine con cui essi sono stati accodati in U_f. Quindi sappiamo che per ogni $i \neq j$ risulta $e = (c_i, c_j) \in E(G_{U_f})$.

L'esito della ricerca varia a seconda dell'ordine in cui vengono selezionati i vertici: se ad esempio nel selezionare il primo vertice l'algoritmo ne seleziona uno "sfortunato", che fa parte di diverse clique tutte di dimensioni minori di $\bar{k}$, allora la clique massima che l'algoritmo potrà trovare sarà di dimensione $\bar{k}-1$.

Il tempo di esecuzione dell'algoritmo varia in base alla dimensione di $N[U_t]$, per $t = 1, \ldots, f$. In media sarà dell'ordine di $O(n^{\alpha f})$ con n^{α} il costo (nel peggiore dei casi) per calcolare il grafo $G_{N[U_t]}$ a partire da $G_{N[U_{t-1}]}$ dove U_t è l'insieme U al tempo t ($G_{N[U_0]} = G$) e f la dimensione della clique trovata.

Algoritmo 4 GREEDY(G)

1: sia $C = V(G)$ e $U = \emptyset$
2: **fintanto che** $C \setminus U \neq \emptyset$ **ripeti**
3: seleziona $v \in C \setminus U$
4: $C = C \cap N[v]$
5: $U = U \cap \{v\}$
6: **fine-ciclo**

2.2.2 Algoritmo Dismantling

L'algoritmo DISMANTLING procede in maniera quasi speculare rispetto all'algoritmo GREEDY: considera ad ogni passo un vertice v di grado minimo e lo elimina dal grafo. L'algoritmo termina quando nel grafo rimangono k vertici di grado $k-1$. L'inesattezza dell'algoritmo risulta evidente se si considera un grafo $K_{k-1} \cup G$ tale che $Cl(G) = k' < k-1$ e G k-ridotto . In questo caso l'algoritmo eliminerà per primi tutti i vertici del sottografo K_{k-1} e la clique massima che potrà trovare sarà di dimensione k'.

Algoritmo 5 DISMANTLING(G)

1: sia $C = V(G)$
2: seleziona $v \in C$ tale che δ_v sia minimale
3: $C = C \setminus \{v\}$
4: **per ogni** $w \in N(v)$ **ripeti**
5: decrementa $\delta(w)$ di 1
6: **fine-ciclo**
7: **fintanto che** $V(G) \setminus U \neq \emptyset$ **ripeti**
8: Seleziono $v \in C \setminus U$
9: $C = C \cap N[v]$
10: $U = U \cap \{v\}$
11: **fine-ciclo**

Il tempo di esecuzione dell'algoritmo è dell'ordine di $O(n^{\alpha(n-\bar{k})})$ con n^{α} il costo per calcolare il grafo $G - \{v\}$ nel caso peggiore e $\bar{k}$ la dimensione della clique trovata.

2.2.3 Algoritmi metodo Monte Carlo

Basati su modelli fisico-matematici, gli algoritmi di tipo Monte Carlo ricercano la clique studiando l'insieme $\mathcal{X}$ delle 2-colorazioni di un grafo e una Hamiltoniana

definita sull'insieme $\mathcal{X}$.

Definizione 26 *Sia G un grafo, una* k-colorazione *di G è una funzione* $\sigma : V(G) \to \{0,\dots,k-1\}$.

Osservazione Si noti che l'insieme $\mathcal{X}$ delle 2-colorazioni di G è in corrispondenza biunivoca con l'insieme $\mathcal{S}(G)$ dei sottografi $H = G - S$ con $S \subseteq V(G)$. Infatti, per ogni $H \in \mathcal{S}(G)$, H è univocamente determinato dall'insieme dei vertici $V(H)$ e possiamo supporre $i \in V(H) \iff \sigma_i = 1$ (dove con σ_i indichiamo il colore del vertice i-esimo). Indicheremo quindi con σ sia la colorazione che il grafo ad essa associato.

Possiamo quindi definire una Hamiltoniana $H(\sigma)$ a valori in $\mathbb{R}$:

$$H(\sigma) := \sum_{(i,j)} J_{ij}\sigma_i\sigma_j - \frac{h}{2}\sum_{i\in V(G)} \sigma_i$$

con $h > 0$ e la funzione J definita sugli spigoli di G in modo seguente

$$J_{ij} := \begin{cases} 1 & \text{se} \quad (i,j) \in E(G) \\ 0 & \text{se} \quad (i,j) \notin E(G) \end{cases}$$

Osservazione Sia $\mathcal{K}(G)$ l'insieme delle clique di G. Se $\sigma \notin \mathcal{K}(G)$, $V(\sigma) = C \cup A$ con C i vertici della clique massima di σ e $|A| > 0 \Rightarrow \forall i \in A$, $H(\sigma) = H(\sigma - i) + \sum_j J_{ij}\sigma_i - \frac{h}{2}$, da cui si ricava il seguente risultato.

Proposizione 11 *Se* $h < 1$ *allora* $H(\sigma)$ *è minimale per ogni* $\sigma \in \mathcal{K}(G)$ *ed è il minimo per una clique massima.*

L'idea alla base degli algoritmi Monte Carlo è di muoversi attraverso i sottografi di G passando da una configurazione all'altra con probabilità positiva soltanto se σ e σ' differiscono di un solo nodo: $P(\sigma,\sigma') = q(\sigma,\sigma')e^{-\beta[H(\sigma)-H(\sigma')]_+)}$ con $q(\sigma,\sigma') > 0$ se e solo se le due configurazioni differiscono per uno stato e $\beta \in \mathbb{R}$.

L'algoritmo in sostanza inizierà a muoversi tra i sottografi di G tendendo a esaminare con maggiore probabilità i sottografi σ con valore $H(\sigma)$ minimale. Dopo un numero "sufficientemente grande"[1] di operazioni l'algoritmo si ferma in una clique σ con valore $H(\sigma)$ molto vicino al minimo assoluto che viene raggiunto nella clique massima del grafo.

[1] Si utilizzano risultati riguardanti le catene di Markov per ricavarlo; a questo proposito si veda ad esempio [8].

2.3 Un nuovo algoritmo

L'algoritmo studiato in questa tesi si basa su una semplice osservazione:

Proposizione 12 *Sia G un grafo e C_k una sua clique di dimensione k. Allora, per ogni $c \in C$, risulta $\delta_G(V) \geq k-1$.*

Dimostrazione Sapendo che per ogni $v \in C$ risulta $\delta_C(v) = k-1$ e che per ogni sottografo H di G risulta $\delta_H(v) \leq \delta_G(v)$, segue la tesi. ■

Quindi nella ricerca di una k-clique possiamo escludere a priori i vertici di G di grado minore di k. Per fare questo è sufficiente un semplice algoritmo TAGLIAGRAFO.

Algoritmo 6 TAGLIAGRAFO(G,k)

1: **per ogni** $w \in V(G)$ **ripeti**
2: **se** $\delta(v) \leq k$ **allora**
3: $G = G - v$
4: **fine-condizione**
5: **fine-ciclo**
6: stop

Osservando attentamente, l'algoritmo TAGLIAGRAFO non ci assicura che il grafo restituito sia k-ridotto . Ad esempio se si considera il grafo stella S_{k+1}, si osserva subito che alla fine dell'operazione di "riduzione" TAGLIAGRAFO(S_{k+1}) avremo il grafo formato dal un solo nodo (il centro della stella) che banalmente non è un grafo k-ridotto . Ma se il grafo G ha cardinalità finita iterando un numero finito di volte l'algoritmo TAGLIAGRAFO restituirà un grafo k-ridotto .

Possiamo quindi definire l'algoritmo TG_k in modo seguente:

Algoritmo 7 $TG_k(G)$

1: **fintanto che** $\forall v \in V(G)$, $\delta(v) \geq k$ **ripeti**
2: **per ogni** $w \in V(G)$ **ripeti**
3: **se** $\delta(v) \leq k$ **allora**
4: $G = G - v$
5: **fine-condizione**
6: **fine-ciclo**
7: **fine-ciclo**
8: stop

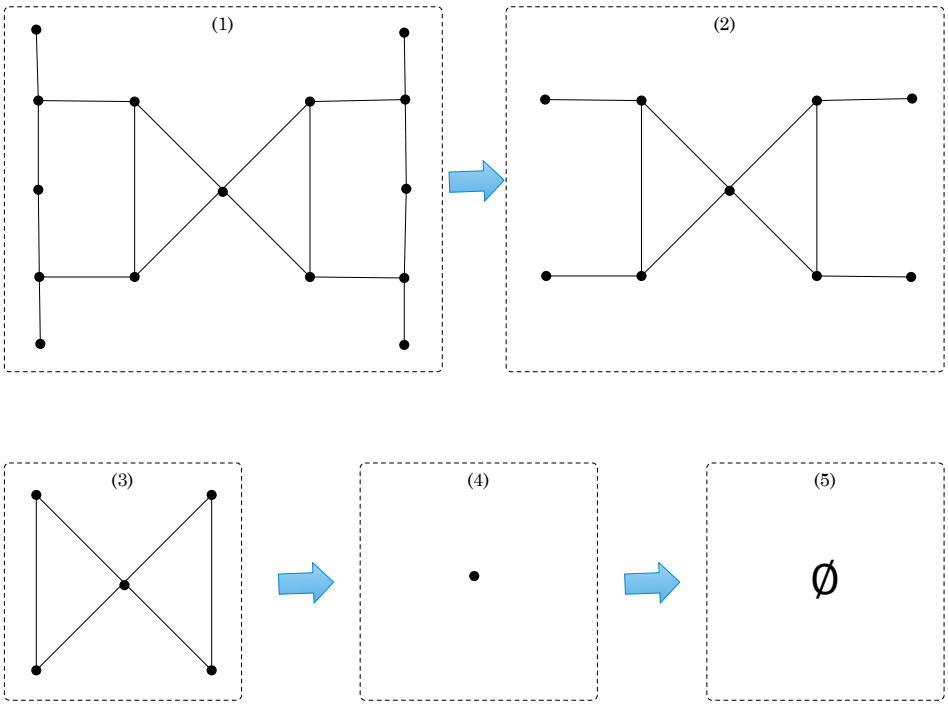


Figura 2.1: Un esempio di come agisce TG_k ($k = 3$) su un grafo che alla fine rimane vuoto

Sia n il numero di vertici di G allora il numero di operazioni di TG_k sarà un numero delle volte (al più n) il numero di operazioni di TAGLIAGRAFO, ma che, pur variando la configurazione dei gradi dei vertici del grafo, al più si avranno n eliminazioni di vertici. Supponendo quindi il costo per il calcolo del grado di un vertice $\delta_G(v)$ costante mentre il costo dell'eliminazione di un vertice sia $O(n)$, allora $\mathrm{TG}_k \in O(n^2)$ nel caso in cui restituisca il grafo vuoto[2].

Operando così su grafi k-ridotto sicuramente si escluderanno dei vertici che per motivi di grado non possono far parte di una k-clique. Per comodità andremo a definire una colorazione sui vertici in maniera da poter considerare l'analisi del grafo come se fosse condotta su livelli. Questa "piramide" di grafi avrà ad ogni livello una copia di un sottografo contenuto nei livelli più bassi: a partire dal grafo "base" G_0, opererà quindi con due funzioni per ricercare localmente all'interno del grafo: la prima (ONEUP) andrà, selezionando un vertice v, a costruire il grafo

[2] Alcune stime sul numero di operazioni di TG_k date nel capitolo 3.1

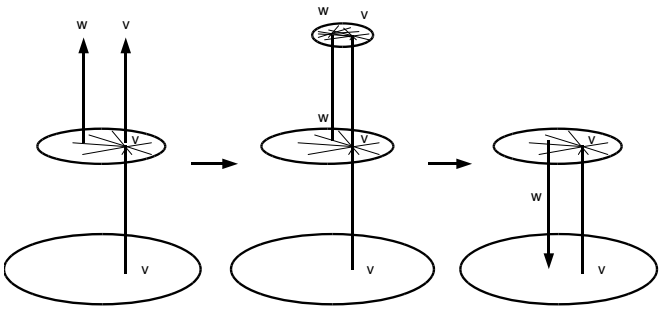


Figura 2.2: Un esempio di come agisce l'algoritmo sul grafo: a partire da un grafo $G_{N[C]}$ tramite ONEUP seleziona un vertice w e va a costruire il grafo al livello superiore, se tale grafo non può contenere una k-clique allora elimina l'ultimo livello costruito e ricostruisce il grafo $G_{N[C]} - w$ tramite ONEDOWN.

generato dai suoi vicini al livello superiore per continuare lo studio in un grafo più piccolo; la seconda (ONEDOWN) servirà ad eliminare l'ultimo piano di questa piramide, per tornare a studiare il grafo sottostante scartando da questo il vertice v usato in precedenza da ONEUP per costruire il grafo al livello superiore. Il motivo dello scarto è dovuto al fatto che il grafo G_{l+1} al livello $l+1$ della piramide, è stato costruito tramite l'intersezione degli $l+1$ vicinati dei vertici $v_1, \ldots, v_{l+1}$. Una volta dimostrato che non può esistere una k-clique contenente $v_1, \ldots, v_{l+1}$ si continua la ricerca a partire dai vertici $v_1, \ldots, v_l$ escludendo in qualsiasi modo che v_{l+1} venga riconsiderato nelle future chiamate di ONEUP sul grafo indotto da $v_1, \ldots, v_l$.

Osservazione Un vertice eliminato al livello l è presente al livello $l-1$. Infatti la non esistenza di un k-clique contenente $v_1, \ldots, v_{l-1}, x$ e il vertice y non implica la non esistenza di una k-clique contenente $v_1, \ldots, v_{l-1}, y$.

Gran parte dell'efficienza dell'algoritmo sarà data da come l'algoritmo ONEUP sceglie il vertice v [3]. Infatti come si può notare la dimensione del grafo restituito da questo dipende unicamente da $\delta(v)$.

[3] Vedi cap.3.2.1.

Algoritmo 8 ONEUP(G,C)

1: seleziona $v \in V(G)$
2: $G = G_{N[v]}$
3: accoda v a C
4: stop

Algoritmo 9 ONEDOWN(G,C)

1: Sia v l'ultimo vertice in U
2: $U = U \setminus \{v\}$
3: $V(G) = N[U]$
4: **per ogni** $x \in N[U], y \in V(G_0)$ **ripeti**
5: **se** $e = (x,y) \notin E(G)$ **allora**
6: $E(G) = E(G) \cup \{e\}$
7: **fine-condizione**
8: **fine-ciclo**
9: $G = G - v$
10: stop

L'ultimo strumento di cui abbiamo bisogno è di un rapido algoritmo di controllo che ci dica se , dato un grafo G, esso può contenere una k-clique. È facile vedere che:

Proposizione 13 *G contiene una k-clique* $\Rightarrow |G| \geq k \wedge ||G|| \geq \frac{k(k-1)}{2}$.

Quindi invece di andare a verificare l'esistenza di tutti gli spigoli tra i vari vertici, possiamo effettuare un più semplice controllo verificando solo se il grafo è abbastanza "grande" per contenere una k-clique.

L'algoritmo di ricerca, quindi, andrà a cercare nel grafo $G_0 = \mathrm{TG}_k(G)$ la k-clique ricercando nei grafi $G_{\cap_{c \in C} N[c]}$ indotti dall'insieme C (i cui elementi possiamo indicizzare a seconda dell'ordine in cui vengono aggiunti a C) che risulterà sempre stellato in G per costruzione. Siccome ad ogni passaggio di livello viene iterato un TG_k sappiamo anche che ad ogni passaggio G è k-ridotto allora sapremo anche che la clique trovata sarà di dimensione almeno k. Inoltre sappiamo che il grafo G_C è completo per motivi analoghi a quelli presentati per l'algoritmo GREEDY.

L'algoritmo terminerà quindi appena $|C| = k$ o G_0 sia il grafo vuoto in un tempo finito. Il grafo G_0 si svuoterà nel caso in cui, non esistendo la k-clique, verranno scartati progressivamente tutti i suoi vertici.

Algoritmo 10 CLIQUECONTROL(G,k)

1: $return := 1$
2: **se** $|G| \leq k$ **allora**
3: il grafo non può contenere una k-clique
4: $return = 0$
5: **fine-condizione**
6: **se** $|G| = k$ **allora**
7: **se** $||G|| = \frac{k(k-1)}{2}$ **allora**
8: il grafo è una k-clique
9: $C = G$
10: $return =$ trovata clique
11: **fine-condizione**
12: **se** $||G|| \leq \frac{k(k-1)}{2}$ **allora**
13: il grafo non può contenere una k-clique
14: $return = 0$
15: **fine-condizione**
16: **fine-condizione**
17: stop

Algoritmo 11 CLIQUEBOY(G,k)

1: $G_0 = \mathrm{TG}_k(G)$
2: $G = G_0$
3: $info = \textsc{CliqueControl}(G,k)$
4: **se** $info =$ trovata clique **allora**
5: G è una k-clique
6: **fine-condizione**
7: **se** $info = 0$ **allora**
8: G non contiene k-clique
9: **fine-condizione**
10: **se** $info = 1$ **allora**
11: **fintanto che** trovata clique oppure $|G_0| = 0$ **ripeti**
12: **se** $\textsc{CliqueControl}(G,k) = 1$ **allora**
13: $G = \textsc{OneUp}(G)$
14: $G = \mathrm{TG}_k(G)$
15: **fine-condizione**
16: **se** $\textsc{CliqueControl}(G,k) = 0$ **allora**
17: **se** $|C| > 1$ **allora**
18: $G = \textsc{OneDown}(G,C)$
19: $G = \mathrm{TG}_k(G)$
20: **altrimenti**
21: $G_0 = G_0 - c$ (dove $C = \{c\}$)
22: $G_0 = \mathrm{TG}_k(G_0)$
23: **fine-condizione**
24: **fine-condizione**
25: **fine-ciclo**
26: **fine-condizione**
27: stop

Capitolo 3

Risultati ricerca

Per il calcolo della complessità computazionale dell'algoritmo CLIQUEBOY bisognerà cominciare a capire quanto la funzione TG_k riduca le dimensioni del grafo. Una volta capito ciò si potrà cercare di stimare quante volte CLIQUEBOY chiami le funzioni ONEUP e ONEDOWN prima esaurire la ricerca sul grafo. Le due strategie utilizzate per la scelta dei vertici per la ricerca presentano ognuna vantaggi e svantaggi:

- Ricerca massima: l'algoritmo seleziona ogni volta un nuovo vertice v di grado relativamente massimo. La probabilità che v sia contenuto nella k-clique risulterà maggiore, ma, nel caso in cui il vertice non appartenga ad una k-clique, bisognerà verificare molte più configurazioni prima di scartarlo
- Ricerca minima: selezionando il vertice relativamente minimo, il tempo necessario affinché l'algoritmo effettui tutti i controlli prima di scartarlo (nel caso non faccia parte di una k-clique) è minore, ma lo è anche la probabilià che appartenga ad una clique di grandi dimensioni.

Proposizione 14 *Sia* $C \subset V(G)$, $|C| = k$, $|G| = n$ *e* $b = \frac{2||G||}{n(n-1)}$ *la densità di un grafo con* $Cl(G) \geq k$*; allora*

- $P_1 = P(C \text{ è } k\text{-clique }) = b^{\frac{k(k-1)}{2}}$.
- $P_2 = P(v \in C \; k\text{-clique}) = \binom{\delta(v)}{k} b^{\frac{k(k-1)}{2}}$

Dimostrazione Supponiamo che G ammetta una clique di dimensione almeno k (in caso contrario $P_1 = 0$) allora, visto che la probabilità che esista lo spigolo

$(u,v) \in E(G) = b$ e una k-clique ha $\frac{k(k-1)}{2}$ spigoli, segue la tesi. Sia quindi $\delta(v)$ il grado di un vertice v, questa sarà anche la cardinalità di $N[v]$ che ammette $\binom{\delta(v)}{k}$ sottoinsiemi di k elementi i quali a loro volta hanno probabilità P_1 di essere una clique[1]. ■

Da questa piccola proposizione si capisce chiaramente che la modalità di scelta dei vertici dovrebbe influenzare l'efficienza dell'algoritmo. Anche se non sappiamo quindi stimare il numero di chiamate delle funzioni ONEUP e ONEDOWN, possiamo dare una stima della loro complessità computazionale:

Proposizione 15 *Sia $n = |G_0|$ e $n_i = |G_i|$ la cardinalità del grafo al livello i; allora risulta* ONEUP $\in O(n^2)$ *e* ONEDOWN $\in O(n^2)$

Dimostrazione ONEUP deve ricercare il vertice v di grado massimo/minimo scorrendo tutti i vertici del grafo ($O(n)$) per poi eliminare $n - \delta(v) \in O(n)$ vertici. Eliminare un vertice costa esattamente $\Theta(n_i) \in O(n)$ quindi ONEUP $\in O(n + n\Theta(n_i)) \subseteq O(n^2)$, mentre ONEDOWN, dovendo andare reinserire tra tutti gli n vertici del grafo G_0 e gli n_{i-1} vertici di G_{i-1} gli spigoli, effettuerà $O(nn_{i-1}) \subseteq O(n^2)$ operazioni, supponendo che il tempo per reinserire uno spigolo sia costante. ■

Osservazione Il valore $n = |G_0|$ in realtà non è fisso ma varia durante l'algoritmo diminuendo all'aumentare dei vertici scartati. Quello che possiamo quindi concludere è che l'algoritmo CLIQUEBOY $\in O(n^{\alpha+2})$ con α il numero di chiamate di ONEUP e ONEDOWN e che tale ordine di grandezza sarà sicuramente molto maggiore della complessità computazionale reale.

3.1 Grafi e partizioni

Quello che si nota nell'algoritmo è che un unico fattore che determina l'efficienza del programma è il numero di vertici che vengono tagliati dall'algoritmo TG_k su un grafo G. Per i motivi che andremo a vedere risulta più facile studiare il numero di vertici eliminati da TAGLIAGRAFO che dipende solamente da quella che possiamo definire come la *partizione associata al grafo*.

[1] Ad una più attenta analisi si potrebbe migliorare la stima considerando che del grafo G è noto il grado di ogni vertice.

Definizione 27 *Una partizione di un intero n è una successione* $\mathfrak{p} := \{p_i\}$ *decrescente, non nulla e finita di numeri naturali t.c.* $||\mathfrak{p}|| = \sum_{k=1}^{n_\mathfrak{p}} p_i = n$ *con* $n_\mathfrak{p} = |\mathfrak{p}|$

Notazione Indicheremo quindi con $\mathfrak{P}_n$ l'insieme delle partizioni di un intero n e con $\mathfrak{P} := \bigcup_{n\in\mathbb{N}} \mathfrak{P}_n$, diremo che $n_\mathfrak{p}$ è la lunghezza di una partizione.

Definizione 28 *Possiamo quindi definire la funzione*

$$\begin{array}{rccl} \Phi: & \mathcal{G} & \to & \mathfrak{P} \\ & G & \to & \mathfrak{p}_G \quad = \{p_i = \delta(v_i)\ v_i \in \bar{G}\} \end{array}$$

dove $\mathcal{G}$ *è l'insieme dei grafi con un numero finito di vertici e* $\bar{G}$ *un grafo isomorfo a* $G - S$ *con* S *l'insieme dei vertici di grado* 0 *e* $\delta(i) \geq \delta(j)\ \forall i > j$ *vertici di* $\bar{G}$.

Esempio Alcuni $\mathfrak{p}_G$ notevoli:

$\mathfrak{p}_{K_{n+1}} = \{\underbrace{n, \cdots, n}_{n+1}\}$

$\mathfrak{p}_{S_{n+1}} = \{n, \underbrace{1, \cdots, 1}_{n}\}$

$\mathfrak{p}_{G^C} = \{n - \delta(v_n), \cdots, n - \delta(v_1)\}$ dove $n = |G|$

$\mathfrak{p}_{C_n} = \{\underbrace{2, \cdots, 2}_{n}\}$

Osservazione Φ sicuramente non è suriettiva su $\mathfrak{P}$ perché per il lemma della stretta di mano $\sum_{x\in V(G)} \delta(x) = 2|E(G)|$ quindi l'immagine di Φ è contenuta in $\mathfrak{P}_2 := \{\text{partizioni dei numeri pari}\}$.

Tramite la definizione di Φ possiamo indurre una partizione dell'insieme $\mathcal{G}$ in base alla partizione $\mathfrak{p}_G$.

Definizione 29 *Due grafi* G *e* G' *si dicono equipartiti* ($G \sim_\Phi G'$) *se* $\mathfrak{p}_G = \mathfrak{p}_{G'}$.

Teorema 16 *G isomorfo a* $G' \Rightarrow G \sim_\Phi G'$.

Dimostrazione Due grafi si dicono isomorfi se esiste una corrispondenza biunivoca che conserva le adiacenze e le "non adiacenze" tra i vertici; quindi in particolare un isomorfismo mantiene il grado di ogni vertice. Il viceversa naturalmente non è vero perché la funzione Φ non tiene conto dei vertici di grado 0. ■

Definizione 30 *Una partizione* $\mathfrak{p}$ *si dice n-graficabile se esiste* $G \in \mathcal{G}$ *tale che* $|G| = n$ *e* $\Phi(G) = \mathfrak{p}$.

La domanda che sorge spontaneo porsi è: per quali partizioni $\mathfrak{p}$ di un intero $2m$ esiste un grafo (semplice) con n vertici e m spigoli? (quando $\mathfrak{p}$ è n-graficabile)

Teorema 17 *Sia* $\mathfrak{p} = \{p_i\}$ *partizione di* $2m$ *n-graficabile allora:*

1. $p_i \leq n-1$ *per ogni i;*
2. $\|\mathfrak{p}\| \geq \lfloor \frac{n}{2} \rfloor$*;*
3. $n_{\mathfrak{p}} = |G - \{$*vertici di grado* $0\}|$ *per ogni* $G \in \Phi^{-1}(\mathfrak{p}_n)$*;*
4. *se esiste un sottoinsieme stellato S in G allora* $|S| \leq min_{v \in V(G)} \delta(v)$*;*
5. *siano* $\mathfrak{p}_1$ *e* $\mathfrak{p}_2$ *due partizioni rispettivamente* n_1 *e* n_2*-graficabili con* $n_1 + n_2 = n$*; allora* $\mathfrak{p}_1 \cup \mathfrak{p}_2$ *definito come un riordinamento decrescente di* $\mathfrak{p} = \{p_{1_1}, \ldots, p_{1_{n_1}}, p_{2_1}, \ldots p_{2_{n_2}}\}$ *è n-graficabile.*

Dimostrazione

1. Supponiamo per assurdo che $p_i \geq n$; allora esiste $v \in V(G)$ tale che $\delta(v) = p_i \geq n$. Allora per il teorema delle "gabbie di piccioni" di Dirichlet deve esistere un vertice w di G collegato tramite 2 spigoli a v.

2. Per ogni $p_i \in \mathfrak{p}$ risulta $p_i \geq 1$.

3. Definito $V_0 = \{$vertici di grado $0\}$, $\forall v \in V(G) \setminus V_0$, $\delta(v) \geq 1$. Allora segue che $\Phi(G) = \Phi(G - V_0)$ quindi $G \in \Phi^{-1}(\mathfrak{p}_n)$ deve avere $n_{\mathfrak{p}}$ vertici di grado non nullo.

4. L'insieme V_{n-1} dei vertici di grado $n-1$ è stellato in G, ne segue che $\forall v \in V_{n-1}$ e $\forall w \in V(G)$ t.c. $w \neq v$, w è adiacente a $v \Rightarrow \delta(w) \geq |V_{n-1}|$.

5. Sia G_1 un grafo in $\Phi^{-1}(\mathfrak{p}_1)$ e G_2 un grafo in $\Phi^{-1}(\mathfrak{p}_2) \Rightarrow \mathfrak{p} = \Phi(G_1 \cup G_2)$ è n-graficabile.

■

Proposizione 18 *Data* $\mathfrak{p}$ *n-graficabile e c fissato, le seguenti affermazioni risultano verificate:*

- *possono esistere grafi equipartiti con differente numero di componenti connesse;*
- *possono esistere grafi* G *e* $G' \in \{G \in \Phi^{-1}(\mathfrak{p})$ *con c componenti connesse,* G *privo di vertici di grado* $0\}$ *non isomorfi.*

Dimostrazione

- I grafi C_6 e $K_3 \cup K_3$ sono equipartiti.
- Sia $K = K_3 - e$, $C = C_6 - e$ con e uno spigolo qualsiasi del relativo grafo $G = C \cup K_3$ e $G' = C_6 \cup K \Rightarrow$ sono equipartiti ma non isomorfi infatti $\mathfrak{p}_G = \mathfrak{p}_{G'} = \{2,2,2,2,2,2,2,1,1\}$.

■

Tornando all'algoritmo CLIQUEBOY si osserva che le scelte su dove andare a cercare la clique all'interno del grafo e come questo viene ridotto tramite la funzione TAGLIAGRAFO (e quindi TG_k) dipendono, a meno di isomorfismi, unicamente dalla partizione $\mathfrak{p}_G$ ad esso associata. Le uniche stime sul grafo TAGLIAGRAFO(G) ricavate, $\mathfrak{p}_G$, sono espresse dalla seguente proposizione:

Proposizione 19 *Sia* $n = |G|$, $m = ||G||$, $t = |V_k| = \{v \in V(G)$ *t.c.* $\delta(v) < k\}$ *e* $s = \sum_{v \in V_0} \delta(v)$. *Allora risulta:*

- $|taglia\ grafo_k(G)| = n - t$
- $m - s \leq ||\text{TAGLIAGRAFO}(G)|| \leq m - s + \sum_{v \in V_0} \overline{\delta(v) - t}^{+}$, *dove con* $\overline{x}^{+}$ *indichiamo la parte positiva di x.*

La minorazione della seconda stima si verifica supponendo che per ogni $v \in V_0$ questo abbia uno spigolo che lo colleghi ad ogni $w \in V_0 \setminus \{v\}$; naturalmente si tratta di una stima migliorabile noti i $\delta(v)$ dei vertici in V_0.

3.2 Risultati al calcolatore

3.2.1 Stima della complessità

Vista la difficoltà nello stimare in modo combinatorio-probabilistico la complessità computazionale dell'algoritmo, è stato implementato nello stesso un sottoprogramma per calcolare il numero di chiamate delle funzioni ONEUP e ONEDOWN delle

quali abbiamo dato una stima. Nella tabella sottostante vengono riportati i valori sperimentali. Le colonne $\#Cl(G)$ e $\#Cl(G)+1$ riportano il numero di chiamate delle funzioni ONEUP e ONEDOWN (se indicato $+\infty$ vuol dire che il programma è stato terminato dopo qualche ora prima del termine dell'esecuzione senza risultato) mentre scarto indica i vertici scartati dall'algoritmo prima di trovare la k-clique di dimensione $Cl(G)$.

Grafo	mod.	$\|G\|$	$\|\|G\|\|$	$Cl(G)$	$t(Cl(G))$	scarto	$t(Cl(G)+1)$
DSJC125.5	min	125	3891	10	2330	2	8778
DSJC125.5	max	125	3891	10	6561	23	6456
DSJC125.9	min	125	6961	34	14994419	4	51280380
DSJC125.9	max	125	6961	34	$+\infty$		$+\infty$
DSJC250.5	min	250	15668	12	192434	62	263148
DSJC250.5	max	125	15668	12	59938	3	266214
DSJC500.5	min	500	62624	13	2582756	13	20403196
DSJC500.5	max	500	62624	13	62600	0	19621018
c-fat200-1	max	200	1534	14	7	0	36
c-fat200-1	min	200	1534	14	9	110	6
c-fat200-2	max	200	3235	24	1	0	78
c-fat200-2	min	200	3235	24	20	176	18
c-fat200-5	max	200	8473	58	30	0	86
c-fat200-5	min	200	8473	58	55	84	52
c-fat500-1	max	500	4459	14	8	0	122
c-fat500-1	min	500	4459	14	360	11	8
c-fat500-2	max	500	9139	26	14	0	196
c-fat500-2	min	500	9139	26	23	240	20
c-fat500-5	max	500	23191	64	33	0	234
c-fat500-5	min	500	23191	64	372	58	
c-fat500-10	max	500	46627	126	64	0	244
c-fat500-10	min	500	46627	126	123	248	120
hamming6-2	max	64	1824	32	17	0	33848
hamming6-2	min	64	1824	32	869	0	2716
hamming6-4	max	64	1824	4	3	0	436
hamming6-4	min	64	1824	4	3	0	294
johnson8-4-4	max	70	1855	14	6	0	18508
johnson8-4-4	min	70	1185	14	1159	0	11446
johnson16-2-4	max	120	5460	8	7	0	6809014
johnson16-2-4	min	120	5460	8	7	0	6173746
p_hat300-1	max	300	10933	8	78	0	7464
p_hat300-1	min	300	10933	8	9009	131	14648
p_hat300-2	max	300	21928	25	78	148	7464
p_hat300-2	min	300	21928	25	489357	148	981636
keller.4	max	171	9435	11	11	0	1450454
keller.4	max	171	9435	11	4	0	1323614

3.2.2 Tempi di verifica

L'algoritmo CLIQUEBOY è pensato per la verifica dell'esistenza di una k-clique in un grafo G. Nelle prossime tabelle verranno riportati il numero $t_G(k)$ di chiamate delle funzioni ONEUP e ONEDOWN durante l'esecuzione del programma su uno

stesso grafo al variare di k. Si potrà osservare la differenza $\Delta_G(k_i,k_j) := t_G(k_1) - t_G(k_2)$ per $k_i,k_j \geq Cl(G)$ è molto maggiore della stessa per $k_i,k_j \leq Cl(G)$. Si potrà notare che in alcuni casi l'andamento di $t_G(k)$ subisce delle irregolarità. Questo è dovuto al fatto che, quando il programma si trova all'interno di una k-clique, questo termina restituendo immediatamente la stessa senza continuare a chiamare funzioni ONEUP.

Tabella 3.1: Grafo: "c-fat200-1.clq"; numero vertici: 200, numero spigoli: 1.534, modalità di ricerca: max

dim clique	esito	scarto	$t_G(k)$
3	SI	0	4
4	SI	0	5
5	SI	0	6
6	SI	0	7
7	SI	0	8
8	SI	0	9
9	SI	0	10
10	SI	0	11
11	SI	0	12
12	SI	0	7
13	NO	200	36
14	NO	200	34
15	NO	200	18
>15	NO	200	1

Tabella 3.2: Grafo: "c-fat200-1.clq"; numero vertici: 200, numero spigoli: 1.534, modalità di ricerca: min

dim clique	esito	scarto	$t_G(k)$
3	SI	0	4
4	SI	0	5
5	SI	0	6
6	SI	0	7
7	SI	0	8
8	SI	0	9
9	SI	0	10
10	SI	0	2
11	SI	100	11
12	SI	110	9
13	NO	200	6
14	NO	200	4
15	NO	200	2
>15	NO	200	1

Tabella 3.3: Grafo: “c-fat200-2.clq”; numero vertici: 200, numero spigoli: 3.235, modalità di ricerca: max

dim clique	esito	scarto	$t_G(k)$	dim clique	esito	scarto	$t_G(k)$
3	SI	0	4	19	SI	0	20
4	SI	0	5	20	SI	0	21
5	SI	0	6	21	SI	0	22
6	SI	0	7	22	SI	0	23
7	SI	0	8	23	SI	0	24
8	SI	0	9	24	SI	0	1
9	SI	0	10	25	NO	200	78
10	SI	0	11	26	NO	200	62
11	SI	0	12	27	NO	200	60
12	SI	0	13	28	NO	200	44
13	SI	0	14	29	NO	200	42
14	SI	0	15	30	NO	200	26
15	SI	0	16	31	NO	200	24
16	SI	0	17	32	NO	200	8
17	SI	0	18	33	NO	200	6
18	SI	0	19	>33	NO	200	1

Tabella 3.4: Grafo: “c-fat200-2.clq”; numero vertici: 200, numero spigoli: 3.235, modalità di ricerca: min

dim clique	esito	scarto	$t_G(k)$	dim clique	esito	scarto	$t_G(k)$
3	SI	0	4	19	SI	0	20
4	SI	0	5	20	SI	0	21
5	SI	0	6	21	SI	0	22
6	SI	0	7	22	SI	0	2
7	SI	0	8	23	SI	154	23
8	SI	0	9	24	SI	176	20
9	SI	0	10	25	NO	200	18
10	SI	0	11	26	NO	200	16
11	SI	0	12	27	NO	200	14
12	SI	0	13	28	NO	200	12
13	SI	0	14	29	NO	200	10
14	SI	0	15	30	NO	200	8
15	SI	0	16	31	NO	200	6
16	SI	0	17	32	NO	200	4
17	SI	0	18	33	NO	200	2
18	SI	0	19	>33	NO	200	1

Tabella 3.5: Grafo: “c-fat200-5.clq”; numero vertici: 200, numero spigoli: 8.473, modalità di ricerca: max

dim clique	esito	scarto	$t_G(k)$	dim clique	esito	scarto	$t_G(k)$
3	SI	0	4	44	SI	0	45
4	SI	0	5	45	SI	0	46
5	SI	0	6	46	SI	0	47
6	SI	0	7	47	SI	0	48
7	SI	0	8	48	SI	0	49
8	SI	0	9	49	SI	0	50
9	SI	0	10	50	SI	0	51
10	SI	0	11	51	SI	0	52
11	SI	0	12	52	SI	0	53
12	SI	0	13	53	SI	0	54
13	SI	0	14	54	SI	0	55
14	SI	0	15	55	SI	0	56
15	SI	0	16	56	SI	0	57
16	SI	0	17	57	SI	0	58
17	SI	0	18	58	SI	0	30
18	SI	0	19	59	NO	200	86
19	SI	0	20	60	NO	200	84
20	SI	0	21	61	NO	200	80
21	SI	0	22	62	NO	200	78
22	SI	0	23	63	NO	200	74
23	SI	0	24	64	NO	200	72
24	SI	0	25	65	NO	200	68
25	SI	0	26	66	NO	200	66
26	SI	0	27	67	NO	200	62
27	SI	0	28	68	NO	200	60
28	SI	0	29	69	NO	200	56
29	SI	0	30	70	NO	200	54
30	SI	0	31	71	NO	200	50
31	SI	0	32	72	NO	200	48
32	SI	0	33	73	NO	200	44
33	SI	0	34	74	NO	200	42
34	SI	0	35	75	NO	200	38
35	SI	0	36	76	NO	200	36
36	SI	0	37	77	NO	200	32
37	SI	0	38	78	NO	200	30
38	SI	0	39	79	NO	200	26
39	SI	0	40	80	NO	200	24
40	SI	0	41	81	NO	200	20
41	SI	0	42	82	NO	200	18
42	SI	0	43	83	NO	200	12
43	SI	0	44	84	NO	200	10
				>84	NO	200	1

Tabella 3.6: Grafo: “c-fat200-5.clq”; numero vertici: 200, numero spigoli: 8.473, modalità di ricerca: min

dim clique	esito	scarto	$t_G(k)$	dim clique	esito	scarto	$t_G(k)$
3	SI	0	4	44	SI	0	45
4	SI	0	5	45	SI	0	46
6	SI	0	7	46	SI	0	47
7	SI	0	8	47	SI	0	48
8	SI	0	9	48	SI	0	49
9	SI	0	10	49	SI	0	50
10	SI	0	11	50	SI	0	51
11	SI	0	12	51	SI	0	52
12	SI	0	13	52	SI	0	53
13	SI	0	14	53	SI	0	54
14	SI	0	15	54	SI	0	55
15	SI	0	16	55	SI	0	56
16	SI	0	17	56	SI	0	2
17	SI	0	18	57	SI	28	57
18	SI	0	19	58	SI	84	55
19	SI	0	20	59	NO	200	52
20	SI	0	21	60	NO	200	50
21	SI	0	22	61	NO	200	48
22	SI	0	23	62	NO	200	46
23	SI	0	24	63	NO	200	44
24	SI	0	25	64	NO	200	42
25	SI	0	26	65	NO	200	40
26	SI	0	27	66	NO	200	38
27	SI	0	28	67	NO	200	36
28	SI	0	29	68	NO	200	34
29	SI	0	30	69	NO	200	32
30	SI	0	31	70	NO	200	30
31	SI	0	32	71	NO	200	28
32	SI	0	33	72	NO	200	26
33	SI	0	34	73	NO	200	24
34	SI	0	35	74	NO	200	22
35	SI	0	36	75	NO	200	20
36	SI	0	37	76	NO	200	18
37	SI	0	38	77	NO	200	16
38	SI	0	39	78	NO	200	14
39	SI	0	40	79	NO	200	12
40	SI	0	41	80	NO	200	10
41	SI	0	42	81	NO	200	8
42	SI	0	43	82	NO	200	6
43	SI	0	44	83	NO	200	4
				84	NO	200	2
				>84	NO	200	1

Tabella 3.7: Grafo: “c-fat500-1.clq”; numero vertici: 500, numero spigoli: 4.459, modalità di ricerca: max

dim clique	esito	scarto	$t_G(k)$
3	SI	0	4
4	SI	0	5
5	SI	0	6
6	SI	0	7
7	SI	0	8
8	SI	0	9
9	SI	0	10
10	SI	0	11
11	SI	0	12
12	SI	0	13
13	SI	0	14
14	SI	0	8
15	NO	500	122
16	NO	500	46
17	NO	500	44
18	NO	500	24
>18	NO	500	1

Tabella 3.8: Grafo: “c-fat500-1.clq”; numero vertici: 500, numero spigoli: 4.459, modalità di ricerca: min

dim clique	esito	scarto	$t_G(k)$
3	SI	0	4
4	SI	0	5
5	SI	0	6
6	SI	0	7
7	SI	0	8
8	SI	0	9
9	SI	0	10
10	SI	0	11
11	SI	0	12
12	SI	0	2
13	SI	348	13
14	SI	360	11
15	NO	500	8
16	NO	500	6
17	NO	500	4
18	NO	500	2
19	NO	500	1

Tabella 3.9: Grafo: “c-fat500-2.clq”; numero vertici: 500, numero spigoli: 9.139, modalità di ricerca: max

dim clique	esito	scarto	$t_G(k)$	dim clique	esito	scarto	$t_G(k)$
3	SI	0	4	20	SI	0	21
4	SI	0	5	21	SI	0	22
5	SI	0	6	22	SI	0	23
6	SI	0	7	23	SI	0	24
7	SI	0	8	24	SI	0	25
8	SI	0	9	25	SI	0	26
9	SI	0	10	26	SI	0	14
10	SI	0	11	27	NO	500	196
11	SI	0	12	28	NO	500	160
12	SI	0	13	29	NO	500	158
13	SI	0	14	30	NO	500	122
14	SI	0	15	31	NO	500	120
15	SI	0	16	32	NO	500	84
16	SI	0	17	33	NO	500	82
17	SI	0	18	34	NO	500	46
18	SI	0	19	35	NO	500	44
19	SI	0	20	36	NO	500	24
				>36	NO	500	1

Tabella 3.10: Grafo: “c-fat500-2.clq”; numero vertici: 500, numero spigoli: 9.139, modalità di ricerca: min

dim clique	esito	scarto	$t_G(k)$	dim clique	esito	scarto	$t_G(k)$
3	SI	0	4	20	SI	0	21
4	SI	0	5	21	SI	0	22
5	SI	0	6	22	SI	0	23
6	SI	0	7	23	SI	0	24
7	SI	0	8	24	SI	0	2
8	SI	0	9	25	SI	216	25
9	SI	0	10	26	SI	240	23
10	SI	0	11	27	NO	500	20
11	SI	0	12	28	NO	500	18
12	SI	0	13	29	NO	500	16
13	SI	0	14	30	NO	500	14
14	SI	0	15	31	NO	500	12
15	SI	0	16	32	NO	500	10
16	SI	0	17	33	NO	500	8
17	SI	0	18	34	NO	500	6
18	SI	0	19	35	NO	500	4
19	SI	0	20	36	NO	500	2
				>36	NO	500	1

Tabella 3.11: Grafo: “johnson8-4-4.clq”; numero vertici: 70, numero spigoli: 1.855, modalità di ricerca: max

dim clique	esito	scarto	$t_G(k)$	dim clique	esito	scarto	$t_G(k)$
3	SI	0	4	29	NO	70	254
4	SI	0	5	30	NO	70	218
5	SI	0	6	31	NO	70	182
6	SI	0	7	32	NO	70	152
7	SI	0	8	33	NO	70	128
8	SI	0	9	34	NO	70	104
9	SI	0	10	35	NO	70	86
10	SI	0	11	36	NO	70	68
11	SI	0	12	37	NO	70	56
12	SI	0	13	38	NO	70	44
13	SI	0	14	39	NO	70	32
14	SI	0	6	40	NO	70	30
15	NO	70	18508	41	NO	70	28
16	NO	70	12054	42	NO	70	26
17	NO	70	7884	43	NO	70	24
18	NO	70	5384	44	NO	70	22
19	NO	70	3812	45	NO	70	20
20	NO	70	2750	46	NO	70	18
21	NO	70	2124	47	NO	70	16
22	NO	70	1626	48	NO	70	14
23	NO	70	1254	49	NO	70	12
24	NO	70	1006	50	NO	70	10
25	NO	70	808	51	NO	70	8
26	NO	70	582	52	NO	70	6
27	NO	70	450	53	NO	70	4
28	NO	70	334	54	NO	70	2
				>54	NO	70	1

Tabella 3.12: Grafo: “johnson8-4-4.clq”; numero vertici: 70, numero spigoli: 1.855, modalità di ricerca: min

dim clique	esito	scarto	$t_G(k)$	dim clique	esito	scarto	$t_G(k)$
3	SI	0	4	29	NO	70	342
4	SI	0	5	30	NO	70	276
5	SI	0	6	31	NO	70	220
6	SI	0	7	32	NO	70	174
7	SI	0	8	33	NO	70	136
8	SI	0	9	34	NO	70	98
9	SI	0	10	35	NO	70	80
10	SI	0	6	36	NO	70	60
11	SI	0	11	37	NO	70	50
12	SI	0	2219	38	NO	70	44
13	SI	0	1592	39	NO	70	40
14	SI	0	1159	40	NO	70	38
15	NO	70	11446	41	NO	70	36
16	NO	70	8190	42	NO	70	32
17	NO	70	6084	43	NO	70	28
18	NO	70	4572	44	NO	70	24
19	NO	70	3520	45	NO	70	20
20	NO	70	2578	46	NO	70	18
21	NO	70	1832	47	NO	70	16
22	NO	70	1294	48	NO	70	14
23	NO	70	970	49	NO	70	12
24	NO	70	784	50	NO	70	10
25	NO	70	656	51	NO	70	8
26	NO	70	570	52	NO	70	6
27	NO	70	488	53	NO	70	4
28	NO	70	408	54	NO	70	2
				>54	NO	70	1

Tabella 3.13: Grafo: “hamming6-2.clq”; numero vertici: 64, numero spigoli: 1.824, modalità di ricerca: max

dim clique	esito	scarto	$t_G(k)$	dim clique	esito	scarto	$t_G(k)$
3	SI	0	4	35	NO	64	13628
4	SI	0	5	36	NO	64	8586
5	SI	0	6	37	NO	64	5484
6	SI	0	7	38	NO	64	3794
7	SI	0	8	39	NO	64	2612
8	SI	0	9	40	NO	64	1690
9	SI	0	10	41	NO	64	1060
10	SI	0	11	42	NO	64	674
11	SI	0	12	43	NO	64	452
12	SI	0	13	44	NO	64	302
13	SI	0	14	45	NO	64	212
14	SI	0	15	46	NO	64	154
15	SI	0	16	47	NO	64	104
16	SI	0	17	48	NO	64	74
17	SI	0	18	49	NO	64	52
18	SI	0	19	50	NO	64	38
19	SI	0	20	51	NO	64	28
20	SI	0	21	52	NO	64	18
21	SI	0	22	53	NO	64	12
22	SI	0	23	54	NO	64	10
23	SI	0	24	55	NO	64	8
24	SI	0	25	56	NO	64	6
25	SI	0	26	57	NO	64	4
26	SI	0	27	58	NO	64	2
27	SI	0	28	59	NO	64	1
28	SI	0	29	60	NO	64	1
29	SI	0	30	61	NO	64	1
30	SI	0	31	62	NO	64	1
31	SI	0	32	>62	NO	64	1
32	SI	0	17				
33	NO	64	33848				
34	NO	64	21344				

Tabella 3.14: Grafo: “hamming6-2.clq”; numero vertici: 64, numero spigoli: 1.824, modalità di ricerca: min

dim clique	esito	scarto	$t_G(k)$	dim clique	esito	scarto	$t_G(k)$
3	SI	0	4	35	NO	64	1484
4	SI	0	5	36	NO	64	1116
5	SI	0	6	37	NO	64	850
6	SI	0	7	38	NO	64	654
7	SI	0	8	39	NO	64	504
8	SI	0	9	40	NO	64	390
9	SI	0	10	41	NO	64	304
10	SI	0	11	42	NO	64	236
11	SI	0	12	43	NO	64	182
12	SI	0	13	44	NO	64	140
13	SI	0	14	45	NO	64	108
14	SI	0	15	46	NO	64	84
15	SI	0	16	47	NO	64	66
16	SI	0	17	48	NO	64	52
17	SI	0	13	49	NO	64	40
18	SI	0	23	50	NO	64	30
19	SI	0	32	51	NO	64	22
20	SI	0	369	52	NO	64	16
21	SI	0	190	53	NO	64	12
22	SI	0	101	54	NO	64	10
23	SI	0	1400	55	NO	64	8
24	SI	0	822	56	NO	64	6
25	SI	0	650	57	NO	64	4
26	SI	0	419	58	NO	64	2
27	SI	0	271	59	NO	64	1
28	SI	0	3337	60	NO	64	1
29	SI	0	2402	61	NO	64	1
30	SI	0	1735	62	NO	64	1
31	SI	0	1250	63	NO	64	1
32	SI	0	869				
33	NO	64	2716				
34	NO	64	1976				

3.3 Considerazioni finali

Un'analisi a posteriori ci fa notare la strategia usata dall'algoritmo CLIQUEBOY risulta essere una combinazione della strategia *greedy* unita ad una routine di *dead end elimination*, cioè un'eliminazione preventiva delle configurazioni che non posso essere soluzioni del problema (la funzione TG_k), molto più attenta della strategia *Dismantling* di riduzione casuale del grafo.

Per come risulta strutturato attualmente, l'algoritmo risulta particolarmente efficiente nei seguenti casi:

- i vertici della clique massima hanno grado massimo (o minimo[2]) all'interno del grafo;
- le clique risultano poco connesse tra di loro (ci sono pochi spigoli che collegano i vertici appartenenti a clique differenti);
- grafi che rappresentati nello spazio euclideo hanno la probabilità di esistenza di uno spigolo tra due vertici inversamente proporzionale alla distanza degli stessi (ad esempio il grafo delle conoscenze interpersonali rappresentato su una carta geografica)[3];
- grafi random in cui la distribuzione dei gradi dei vertici è prossima alla dimensione della clique cercata o ha una varianza alta [4].

Per quanto visto a proposito della funzione TG_k (cap. 2.3), i suoi casi peggiori (cioè l'eliminazione di tutti o di un gran numero di vertici dal grafo) rappresenta un caso favorevole per la ricerca perché, restituendo un grafo con pochi vertici, riduce il numero di configurazioni prese in esame. Una possibile via per l'ottimizzazione dell'algoritmo potrebbe essere quella di trovare un modo efficiente di selezione dei vertici che permetta il maggior numero di "potature" prima di iniziare ogni ricerca locale o una volta che la ricerca abbia dato esito negativo.

[2] A seconda di come si scelgono i vertici durante l'esecuzione dell'algoritmo.

[3] In questo tipo di grafi la strategia di ricerca locale *greedy* unita al *dead end elimination* permette di scartare un gran numero di configurazioni.

[4] Questi sono i casi in cui TG_k opera un gran numero di tagli.

Il vero discriminante per l'efficienza dell'algoritmo risulta essere il modo in cui, di volta in volta, vengono scelti i vertici su cui operare la ricerca locale. Un ulteriore studio sui grafi associati alle partizioni potrebbe, forse, rivelare qualche informazione che permetta di scartare altre configurazioni alleggerendo quindi il numero di quelle da analizzare.

Altri tentativi di miglioramento dell'efficienza dell'algoritmo potrebbero essere condotti selezionando il vertice su cui operare la ricerca non semplicemente in base al grado del vertice, ma in base ad una valutazione che tenga conto anche della densità del vicinato del vertice (cioè il rapporto tra il numero massimo di spigoli del grafo indotto dal vicinato di un vertice e il numero reale di questi) o implementando una sorta di *page rank*[9] mirato ad una valutazione (probabilistica) della grandezza della clique massimale contenente il vertice.

In definitiva l'algoritmo risulta suscettibile a varie ipotesi di miglioramento, risulta ancora aperto il problema del calcolo della sua complessità computazionale e il problema che questo ha fatto emergere delle partizioni graficabili. Insomma: per ora, come ci si poteva aspettare, il "probelma da un milione di dollari" è ancora aperto, ma in compenso abbiamo più domande di quante ne avevamo all'inizio di cui però comprendiamo meglio la natura!

Appendice A

Sorgenti dei programmi

A.1 Il programma CLIQUEBOY

Il programma in linguaggio C riportato nelle pagine seguenti, basandosi sull'algoritmo CLIQUEBOY (Algoritmo 11, cap. 2.3), permette la verifica dell'esistenza di una k-clique e l'individuazione della clique massima, sia tramite una ricerca sequenziale crescente che tramite una ricerca binaria (partendo dal valore $2\log_b(n)$[1]). Il programma lavora a livelli: il grafo viene posto al livello 1, i vertici che non possono appartenere a nessuna k-clique vengono progressivamente isolati nel livello -1 per evitare ripescaggi. Il programma poi esplora il grafo salendo di livello selezionando i vertici e accodandoli alla clique C. Quando scende ricostruisce il grafo $G_N[C']$ dove $C' = C \setminus \{v_{|C|}\}$ cioè il grafo generato dal vicinato della clique C meno l'ultimo vertice inserito.

```
1  #include <stdlib.h>
2  #include <stdio.h>
3  #include <string.h>
4  #include <math.h>
5
6  notazione:
7      - contatore
8      + stringa
9      +- stinga ordinata := stringa con il valore associato
10         al verice v nella v-esima posizione
11     ++ matrice
12
13 variabili:
14
```

[1] Valore più probabile per $Cl(G)$ [5].

```
++ nghBrs := matrice di adiacenza di G
+- degTot := stringa ordinata dei gradi dei vertici
- n := |G|
- nTmp := |G_{liv}| (lunghezza di nodeTmp)
- mTmp := ||G_{liv}||
+ nodeTmp := stinga contenente i vertici del grafo
+ liv := quota del sollevamento del grafo
+- inGraph := stringa ordinata del livello piu' alto in
                     cui si trova il vertice
+- degTmp := l'equivalente di degTot di G_{liv}
+- inClique := stringa ordinata dei livelli in cui si
                      inserisce il vertice nella clique
+ cliqueTmp := stringa contenente i vertici della clique
- dClique := grandezza della clique (lunghezzadi cliqueTmp)

- swN := intero per selezionare la modalitÃ  del
               programma per ogni N
- upper := upperbound del clique number
- lower := lower bound del clique number
- cont := un contatore globale
- mis := contatore per i vertici scartati dalla ricerca


int **nghBrs ;
int *degTot ;

int n;
int nTmp ;
int mTmp ;
int *nodeTmp ;

int liv;
int *inGraph ;
int *degTmp ;

int *inClique;
int *cliqueTmp;
int dClique;

int sw1;
int sw2;
int upper;
int lower;
int cont;
int mis;

char nome[128];
char output[128];

```

```
64  void allocaSpazio() {
65    int i;
66    nghBrs = (int**)calloc(n,sizeof(int*)) ;
67    inGraph = (int*)calloc(n,sizeof(int)) ;
68    degTot = (int*)calloc(n,sizeof(int)) ;
69    degTmp = (int*)calloc(n,sizeof(int)) ;
70    nodeTmp = (int*)calloc(n,sizeof(int)) ;
71    inClique = (int*)calloc(n,sizeof(int));
72    cliqueTmp = (int*)calloc(n,sizeof(int));
73    for(i = 0 ; i < n ; i++) nghBrs[i]= (int*)calloc(n,sizeof(int));
74    return;
75  }
76
77  void stampaClique (){
78    int i;
79    printf("Clique trovata: dim=%d\t", dClique);
80    if(dClique>0){
81      for(i=0; i<dClique-1;i++){
82        printf(" %d : ", cliqueTmp[i]+1);
83      }
84      printf(" %d \n ", cliqueTmp[i]+1);
85    } else printf("empty clique\n");
86    return;
87  }
88
89  void stampaGrafo (){
90    int i,j;
91    for (i=0; i<n; i++){
92      if (inGraph[i]==liv){
93        if (inClique[i]!=0) printf("->\t");
94        else printf("\t");
95        printf(" %d : \t", i+1);
96        for (j=0; j<n; j++) {
97          if (inGraph[j]==liv)
98            printf("%d \t ", nghBrs[i][j]);
99        }
100       printf("\n" );
101     }
102   }
103   printf("|G|=%d, ||G||=%d \n Vertici nel grafo: \t", nTmp, mTmp);
104   for (i=0; i<nTmp; i++)
105     printf("%d\t", nodeTmp[i]+1); printf("\n");
106   printf("registro clique:\t");
107   stampaClique();
108   return;
109 }
```

Date queste prime funzioni per l'assegnazione dello spazio in memoria e per la gestione dei grafi, le funzioni impostano il file di input, salvano le informazioni richieste, e leggono i grafi scritti in formato DIMACS (vedi appendice B).

```
void imposta (){
 FILE *out;
  out=fopen(output, "at");
  if(sw2==1) fprintf(out, "Grafo: %s\t numero vertici:%d\t
                    numero spigoli: %d\tmodalita' di ricerca: max\n",
                    nome, n, mTmp);
  if(sw2==2) fprintf(out, "Grafo: %s\t numero vertici:%d\t
                    numero spigoli: %d\tmodalita' di ricerca: min\n",
                    nome, n, mTmp);
  fclose(out);
  return;
}

void memo (int K, int cont, int goal){
  FILE *out;
  out=fopen(output, "at");
  if(goal==0) fprintf(out, "dimensione clique=%d \tesito: NO\t
               #vertici scartati=%d\t# operazioni=%d\n", K, mis, cont);
  if(goal==1) fprintf(out, "dimensione clique=%d \tesito: SI\t
               #vertici scartati=%d\t# operazioni=%d\n", K, mis, cont);
  fclose(out);
  return;
}

void leggiGrafoDIMACS(){
  int i,j,m,v1,v2;
  FILE *f;
  char c;
/*inizializzo i contatori*/
  liv=1;
  mTmp=0;
  nTmp=0;
  dClique=0;
/*scorre il file, la prima lettera della riga indica
     il tipo di informazioni*/
  if (f = (fopen(nome, "r"))) {
    while (!feof(f)) {
      c = fgetc(f);
      switch (c) {
/*c sono commenti, passo alla riga sotto*/
      case 'c':
    while (c != '\n') {
      c = fgetc(f);
  }
  break;
```

```
/* linea del formato del grafo, p FORMAT NODES EDGES
FORMAT deve essere edge per i grafi costruiti
       per il problema della clique,
NODES il numero di vertici
EDGES il numero di spigoli
*/
     case 'p':
  fscanf(f, "%s %d %d", &c, &n, &m);
  allocaSpazio();
/*svuoto le stringhe... serve? calloc non le imposta ==0?*/
    for( i = 0 ; i < n ; i++){
      inClique[i] = 0;
      inGraph[i] = 0;
      degTot[i] = 0;
      degTmp[i] = 0;
      for(j = 0 ; j < n ; j++) nghBrs[i][j]=0;
    }
  break;
/*iniziano gli spigoli:
e VERTEX1 VERTEX2
Attenzione:
i vertici per comodita' nel programma partono da 0
mentre nei file da 1, si riscalano

aggiorno la matrice di adiacenza
se il/i vertice/i non era ancora presente nel grafo
lo aggiungo sia ad inGraph che a nodeTmp
aumento di 1/2 il valore di nTmp
aumento di 1 il valore di mTmp
aumento di 1 il grado di entrambi i vertici
*/
     case 'e':
  fscanf(f, "%d %d", &v1, &v2);
  v1 = v1 - 1 ;
  v2 = v2 - 1 ;
  nghBrs[v1][v2]=1;
  nghBrs[v2][v1]=1;
  degTot[v1]++;
  degTot[v2]++;
  mTmp++;
  if(inGraph[v1]==0){
    inGraph[v1]=1;
    nodeTmp[nTmp]=v1;
    nTmp++;
  }
  if(inGraph[v2]==0){
    inGraph[v2]=1;
    nodeTmp[nTmp]=v2;
    nTmp++;
```

```
  }
  break ;
      }
    }
  }
  else {
    printf("Errore nella lettura del file.\n\n");
    n = -1;
  }
  for(i=0; i<n;i++) degTmp[i] = degTot[i] ;
/*printf("GRAFO MADRE;\n");
stampaGrafo();*/
  return;
}
```

Qui iniziano le funzioni per la ricerca vera e propria della clique. La funzione $togliNodo(v)$ elimina il nodo v dal grafo al livello in cui si trova e mettendolo nel grafo un livello sotto, abbassa il grado di tutti i suoi vicini di 1, diminuisce di $\delta(v)$ il numero di spigoli del grafo al livello ed elimina v dalla lista dei vertici del grafo.

```
void togliNodo(int v) {
  int i,w;
/*printf("sto togliendo il vertice %d\n", v+1);*/
  for(i = 0 ; i < nTmp ; i++) {
    w=nodeTmp[i];
    if(nghBrs[w][v]==1){
  degTmp[w]-- ;
  degTmp[v]-- ;
  mTmp-- ;
    }
  }
  inGraph[v]-- ;
  i=0;
  while(nodeTmp[i] != v) i++ ;
  nodeTmp[i] = nodeTmp[nTmp - 1] ;
  nTmp-- ;
  return;
}
```

La funzione $TG(K)$ implementa l'algotimo $\mathrm{TG}_k(G)$ (algoritmo 7 ,cap. 2.3), mentre $TGkill(K)$ ripete le stesse operazioni di $TG(K)$ nel grafo al livello 1 eliminando definistivamente i vertici dal grafo G_1 su cui si sta cercando la clique.

```
void TG(int K) {
  int i, t;
  int flag=1 ;
/*printf("\nTG\n");*/
```

```
  while(flag==1 && nTmp!=0){
    flag=0;
    t=nTmp;
    for(i = 0 ; i < t ; i++) {
      if(degTmp[nodeTmp[i]] < K-1) {
    togliNodo(nodeTmp[i]);
    flag = 1 ;
      }
    }
  }
  return;
}

void TGkill(int K){
  int i;
  TG(K);
  for(i=0;i<n;i++)if(inGraph[i]==0){
   inGraph[i]=-1;
   mis++;
printf("\t\t# vertici scartati:%d\n", mis);
  }
  return;
}
```

La funzione ONEUP implementa l'algoritmo ONEUP (algortimo 8, cap 2.3). A seconda della modalità con cui si fa girare il programma questa sceglierà il primo (rispetto l'ordine con cui sono stati accodati i vertici durante la lettura) vertice di grado massimo/minimo. Il grafo viene "sollevato" portando tutto il grafo al livello superiore e poi abbassando i vertici non adiacenti al vertice selezionato.

```
void oneUp() {
  int i ,j,v,w,t,deg, degM;
/*cerco il vertice di deg massimo o minimo*/
  if(sw2==1){
    degM=0;
    for(i = 0 ; i < nTmp ; i++) {
     j = nodeTmp[i] ;
     deg=degTmp[j];
     if(degM < deg && inClique[j] == 0) {
       degM = degTmp[j] ;
       v = j ;
       t=i;
     }
   }
  }else{
    degM=n;
    for(i = 0 ; i < nTmp ; i++) {
```

```
18      j = nodeTmp[i] ;
19      deg=degTmp[j];
20      if(degM > deg && inClique[j] == 0) {
21        degM = degTmp[j] ;
22        v = j ;
23        t=i;
24      }
25     }
26    }
27    inClique[v]=1;
28    cliqueTmp[dClique]=v;
29    dClique++;
30  /*tiro su il vicinato tirando prima tutto il livello su e
31   poi abbassando il tutto mantenendo alto solo il vicinato*/
32    for(i = 0 ; i < nTmp ; i++) {
33      w=nodeTmp[i];
34      inGraph[w]++;
35    }
36    t=nTmp;
37    for(i=0;i<t; i++){
38      w=nodeTmp[i];
39      if(nghBrs[v][w]==0 && v != w ) togliNodo(w);
40    }
41    liv++;
42    return;
43  }
```

La funzione *recuperaNodo*(v) serve per recuperare un nodo v dal un livello sottostante e riinserirlo nel grafo ristabilendo tutti gli spigoli che lo connettono con i vertici del grafo al livello.

```
1   void recuperaNodo(int v){
2     int i;
3     for(i=0;i<n;i++){
4       if(inGraph[i]==liv || inGraph[i]==liv+2)if(nghBrs[i][v]==1){
5         degTmp[i]++ ;
6         degTmp[v]++ ;
7         mTmp++ ;
8       }
9     }
10    nodeTmp[nTmp]=v;
11    nTmp++;
12    inGraph[v]++;
13    return;
14  }
```

La funzione ONEDOWN, come sopra, implementa l'algoritmo ONEDOWN(algoritmo 9 cap. 2.3) recuperando tutti i vertici posti un livello sotto con la funzione

recuperaNodo appena vista: viene provvisoriamente creato un livello intermedio tra il livello dove stiamo lavorando e il sottostante per evitare di contare più volte del dovuto gli spigoli tra i vertici che vengono recuparati.

```
void oneDown (){
  int i;
  liv--;
  for(i=0;i<n;i++) if(inGraph[i]==liv+1) inGraph[i]++;
  for(i=0;i<n;i++) if(inGraph[i]==liv) recuperaNodo(i);
  for(i=0;i<n;i++) if(inGraph[i]>=liv+1) inGraph[i]=liv;
  dClique-- ;
  togliNodo(cliqueTmp[dClique]);
  inClique[cliqueTmp[dClique]]=0;
  return;
}
```

Quando si esclude definitivamente un vertice v dal grafo è perché abbiamo escluso un numero sufficente di suoi vicini da far si che $\delta(v) < K$, quindi questo verrà scartato da un'operazione di $TGkill(K)$. In questo caso per continuare la ricerca della clique sugli altri vertici, è necessario ricostruitre il grafo al livello 1, evitando però di riconsiderare i vertici scartati (che sono stati posto al livello 0).

```
void rebuildGraph (){
  int i,j;
  liv=1;
  nTmp=0;
  mTmp=0;
  dClique=0;
  for(i=0;i<n;i++)if(inGraph[i]!=-1){
   degTmp[i]=0;
   inGraph[i]=1;
   inClique[i]=0;
   nodeTmp[nTmp]=i;
   nTmp++;
   for(j=0;j<n;j++)if(nghBrs[i][j]==1 && inGraph[j]!=-1)degTmp[i]++;
   mTmp=mTmp+degTmp[i];
  }
  mTmp=mTmp/2;
  return;
}
```

La funzione $cliqueControl(K)$ implementa l'algoritmo 10. Restituisce:

- flag==-1 manca spazio per una clique di dimensione K (troppi pochi vertici o spigoli)

- flag==1 il grafo è una clique di dimensione K
- flag==2 ho una clique di dimensione K nel registro della clique e il grafo è un supergrafo di questa
- flag==0 c'è lo spazio per una clique di dimensione K

```
int cliqueControl (int K) {
  int flag;
  flag=0;
  if(nTmp < K) flag= -1;
  if(nTmp == K){
    if(mTmp == K*(K-1)/2) flag=1;
    if(mTmp < (K*(K-1))/2 ) flag=-1;
  }
  if(nTmp>K && dClique==K)(flag=2);
  return(flag);
}
```

Come l'algoritmo CLIQUEBOY (11) la funzione *cliqueBoy*(K) ricerca nel grafo se esiste una clique di dimensione K utilizzando successive chiamate di ONEUP, ONEDOWN, e $TG(K)$.

```
int cliqueBoy (int K){
  int flag, goal=0, w=0;
  mis=0;
  TGkill(K);
  flag=cliqueControl(K);
/*prima flag==1 => il grafo Ã¨ una clique, il grafo non e'
   una clique , inizio la ricerca */
  if(flag!=1){
    while(flag!=1 && goal==0){
/*flag==0 => il grafo e' abbastanza spazioso per
  poter ospitare una clique=> salgo di un livello*/
     if(flag==0){
       oneUp();
       TG(K);
     }
/*flag==-1=> il grafo non e' abbastanza spazioso
  se sto a un livello >1 scendo di un livello,
  altrimenti scarto il grafo*/
     if(flag==-1 ) {
      if(dClique <= 1 ){
        if(dClique==1){
  inGraph[cliqueTmp[0]]=-1;
  mis++;
printf("\t\t# vertici scartati:%d\n", mis);
```

```
25          rebuildGraph();
26          TGkill(K);
27         }
28        if(cliqueControl(K) == -1) goal=2;
29       }else{
30   oneDown();
31   TG(K);
32       }
33      }
34     if(flag==2) goal=1;
35    flag=cliqueControl(K);
36 cont++;
37     }
38   }
39 /*se sono uscito con flag ==1 potrei non aver inserito
40    tutti i vertici della clique nel registro della clique*/
41   if(flag==1){
42     goal=1;
43     while(dClique!=K){
44       while(inGraph[w]!=liv || inClique[w] !=0 ) w++;
45 /*printf("aggiungo il vertice %d alla clique\n", w+1);*/
46      inClique[w]=1;
47      cliqueTmp[dClique]=w;
48      dClique++;
49     }
50   }
51   if(goal==2)goal=0;
52   if(flag==-1)goal=0;
53   if(sw1>3) memo(K, cont, goal);
54   return(goal);
55 }
```

La funzione $binarySearch(K, goal)$ in base all'esito della ricerca della k-clique restituisce il successivo valore per cui va cercata la clique massima, se la ricerca sta iniziando restituisce il valore $2\log_b n$ con b la densità del grafo.

```
1  int binarySearch (int K, int goal){
2    float b;
3    int bound;
4    if(K==0 && goal==0){
5      b=((float)n*(float)(n-1)) / (float)(2*mTmp);
6      bound=(int)(2*log(log(n))/log(n))+1;
7      K=2*(int)(log(n)/log(b));
8      lower=0;
9      upper=n-1;
10   }else{
11    if(upper > lower){
12      if(goal==0){
```

```
        upper=K;
        K=(K+lower)/2;
      }
      if(goal==1){
        lower=K;
        K=(K+upper)/2;
      }
    }
   }
return(K);
}
```

Le due funzioni che seguono sono funzioni di supporto per salvare e recuperare le k-clique ricercate altrimenti il programma le dimenticherebbe all'inizio della ricerca di una k'-clique.

```
int *copyClique (int *p){
   int i;
   if(p == NULL) p= (int*)calloc(n+1,sizeof(int));
   p[0]=dClique;
   for(i=0;i< dClique ;i++) p[i+1]=cliqueTmp[i];
   return(p);
}
void pasteClique (int *p){
   int i;
   dClique=p[0];
   for(i=0;i<dClique;i++) cliqueTmp[i]=p[i+1];
   return;
}
```

La funzione main seleziona la modalità in cui utilizzare il programma.

```
int main (void){
  int K=0, goal=0, cg=1, flag=0, *p=NULL, sw3, val;
  printf("Programma di ricerca Clique\n\nSELEZIONARE TIPO DI RICERCA:\n
1 Ricerca di una clique di dimensione nota\n
Ricerca della clique massima:\n
2 Ricerca binaria\n
3 Ricerca crescente\n
4 Calcola i tempi per ogni dimensione
\n\nMODALITA' SELEZIONATA:");
  scanf("%d", &sw1);
  printf("\nSelezionare modalita' di selezione dei vertici:\n
1 Scegli un vertice relativamente massimo\n
2 Scegli un vertice relativamente minimo\n\n
MODALITA' SELEZIONATA:");
  scanf("%d", &sw2);
if(sw1==4){
```

```
  printf("\nSelezionare fin quando continuare a calcolare i tempi:\n
1 Tutti\n
2 Clique number+1 (caso peggiore)\n
3 Fino ad un valore assegnato\nMODALITA' SELEZIONATA:");
  scanf("%d", &sw3);
}
if(sw1==4 && sw3==3){
  printf("\nInserire il valore massimo su cui cercare al clique:");
  scanf("%d", &val);
}
/*legge il nome del file*/
  printf("Nome del file: ");
  scanf("%s",nome);
  leggiGrafoDIMACS();
  if(n>0){
/*ricerca su K assegnato*/
   if(sw1==1){
     printf("dimensione clique da trovare:");
     scanf("%d", &K);
     TG(K);
     liv=1 ;
     goal=cliqueBoy(K);
   }
/*ricerca binaria*/
   if(sw1==2){
     while(flag!=1){
         cg=K;
         K=binarySearch(K, goal);
         if(K != cg ) {
            leggiGrafoDIMACS();
             printf("cerco clique di dimensione %d\n", K);
            goal=cliqueBoy(K);
         }
         else flag=1;
      }
     goal=1;
   }
/*ricerca sequenziale crescente*/
   if(sw1==3){
     K=2;
     if( mTmp >= 1) goal=1;
     while(goal!=0){
         printf("Trovata clique di dimensione %d\n", K);
         K++;
         p=copyClique(p);
         if(nTmp!= n-mis) rebuildGraph();
         goal=cliqueBoy(K);
      }
     pasteClique(p);
```

```
66      K--;
67      goal=1;
68    }
69  /* calcolo tempi per ogni k*/
70    if(sw1==4){
71      printf("nome file output: ");
72      scanf("%s", output);
73      imposta();
74      for(K=3; K<n;K++){
75        printf("cerco clique di dimensione %d\n", K);
76        if(cliqueBoy(K)==0) if(sw3==2) K=n;
77        if(sw3==3 && val==K) K=n;
78        leggiGrafoDIMACS();
79        cont=0;
80      }
81    }
82  /*output esito ricerca*/
83    if(sw1!=4){
84      if(goal==1) {
85        printf("trovata clique di dimensione %d:\n", K);
86        stampaClique();
87      }
88      else printf("il grafo non contine clique di dimensione %d\n", K);
89      printf("# totale operazioni: %d\n", cont);
90    }
91  }
92   printf("\nEND\n");
93   return(0);
94  }
```

Appendice B

Grafi DIMACS

I grafi utilizzati per la raccolta dei dati sperimentali. Sono disponibili in rete e vengono utilizzati per la sperimentazione di nuovi algoritmi su grafi [3]. I file *DIMACS* sono memorizzati in ASCII o in binario. Il formato in binario risulta particolarmente difficile da visualizzare ma permette di risparmiare spazio in memoria. Memorizzati in ASCII presentano una struttura intuitiva di descrizione del grafo presentando, una volta date le dimensioni del grafo (numero di vertici e numero di spigoli), la lista degli spigoli.

Il primo carattere presente su ogni riga fornisce le informazioni sul contenuto della stessa. Le differenti tipologie di informazioni contenute nelle righe sono le seguenti:

- **Comment Lines.**

  ```
  c :Linea di commento
  ```

- **Problem Line.**

  ```
  p  edge #vertici #spigoli
  ```

- **Edge Descriptors.**

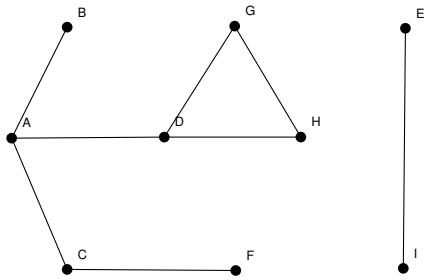
  ```
  e vertice1 vertice2
  ```

- **Solution line**

  ```
  s una soluzione del problema (vertici di una clique massima)
  ```

Esempio:

```
c Grafo di prova
p edge 9 8
e A B
e A C
e A D
e C F
e D G
e D H
e E I
e G H
s D G H
```



Appendice C

Grafi equipartiti

A seguito viene riportata la classificazione delle partizioni (ordinate lessicograficamente) $\mathfrak{p}$ di lunghezza compresa tra 2 e 5 e delle relative classi di isomorfismo di grafi (privi di vertici di grado 0). Trattandosi di grafi molti piccoli i problemi derivanti dalla non iniettività della funzione Φ risultano di semplice soluzione, ma giÃ da questi semplici esempi (vedi tabella C.3, C.4, C.5,C.6) si può intuire la difficoltà del problema (combinatorio) di individuare quanti e quali altri parametri sono necessari per poter univocamente determinare un grafo a partire da una partizione.

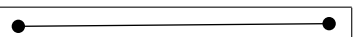
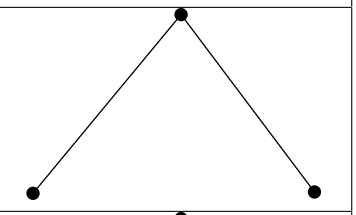
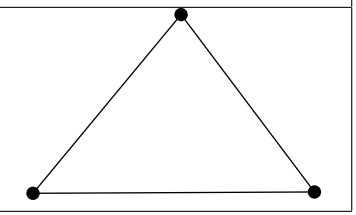
$n=2$	$k=2$	1,1	
$n=3$	$k=2$	2,1,1	
$n=3$	$k=3$	2,2,2	

Tabella C.1: Grafi relativi alle partizioni

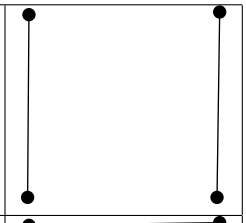
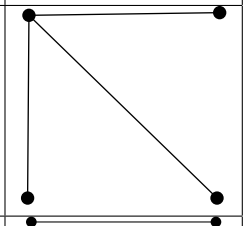
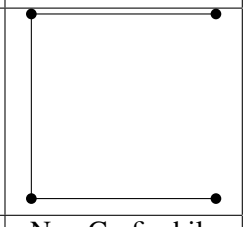
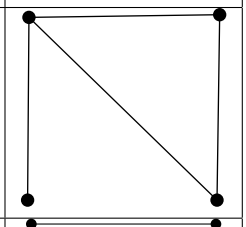
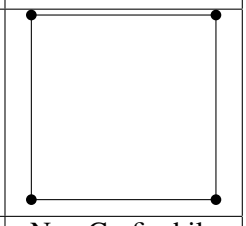
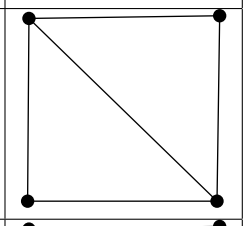
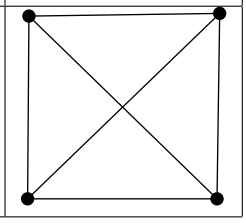
$n=4$	$k=2$	1,1,1,1	
$n=4$	$k=3$	2,2,1,1	
$n=4$	$k=3$		
$n=4$	$k=4$	3,3,1,1	Non Graficabile
$n=4$	$k=4$	3,2,2,1	
$n=4$	$k=4$	2,2,2,2	
$n=4$	$k=5$	3,3,3,1	Non Graficabile
$n=4$	$k=5$	3,3,2,2	
$n=4$	$k=6$	3,3,3,3	

Tabella C.2: Classi di equivalenza dei grafi di 4 vertici

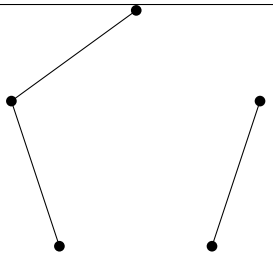
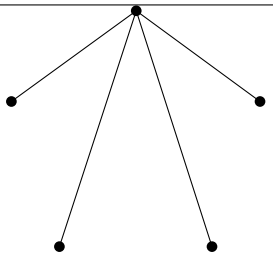
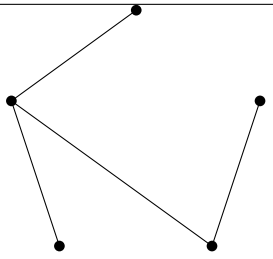
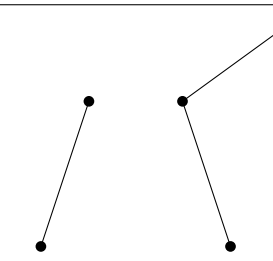
$n = 5$	$k = 3$	2,1,1,1,1	
$n = 5$	$k = 4$	4,1,1,1,1	
$n = 5$	$k = 4$	3,2,1,1,1	
$n = 5$	$k = 4$	2,2,2,1,1	

Tabella C.3: Classi di equivalenza dei grafi di 5 vertici, 4 spigoli

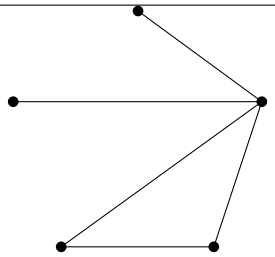
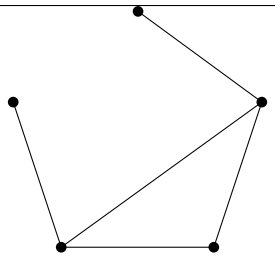
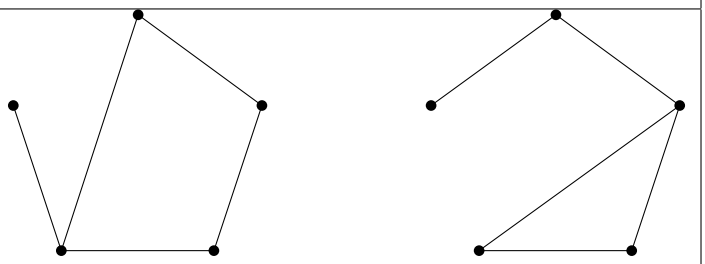
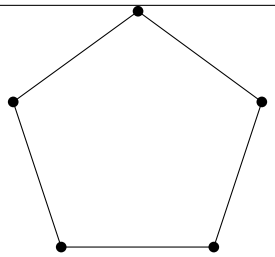
$n = 5$	$k = 5$	4,3,1,1,1	Non Graficabile
$n = 5$	$k = 5$	4,2,2,1,1	
$n = 5$	$k = 5$	3,3,2,1,1	
$n = 5$	$k = 5$	3,2,2,2,1	
$n = 5$	$k = 5$	2,2,2,2,2	

Tabella C.4: Classi di equivalenza dei grafi di 5 vertici, 5 spigoli

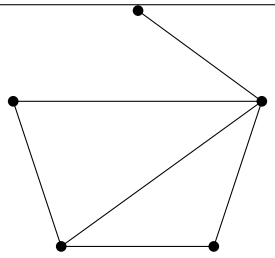
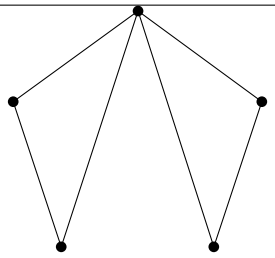
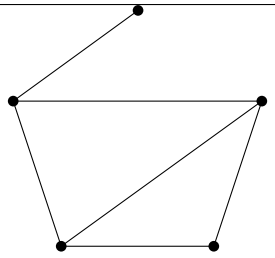
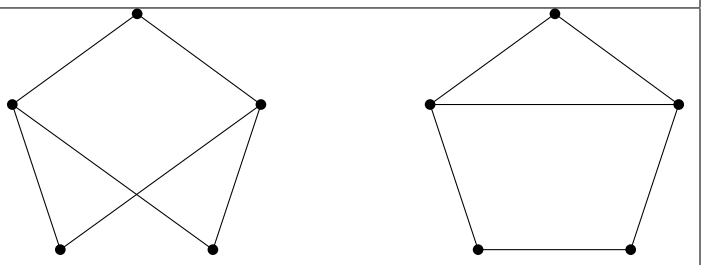
$n = 5$	$k = 6$	4,4,2,1,1	Non Graficabile
$n = 5$	$k = 6$	4,3,3,1,1	Non Graficabile
$n = 5$	$k = 6$	4,3,2,2,1	
$n = 5$	$k = 6$	4,2,2,2,2	
$n = 5$	$k = 6$	3,3,3,2,1	
$n = 5$	$k = 6$	3,3,2,2,2	

Tabella C.5: Classi di equivalenza dei grafi di 5 vertici, 6 spigoli

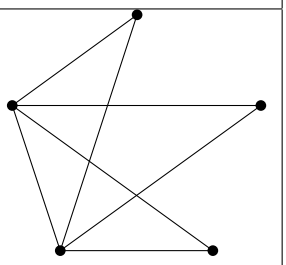
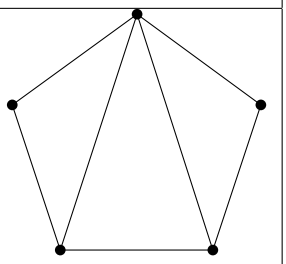
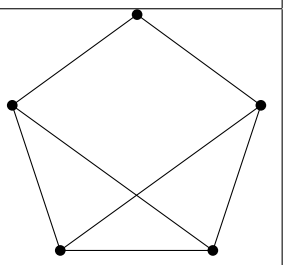
$n = 5$	$k = 7$	4,4,4,1,1	Non Graficabile
$n = 5$	$k = 7$	4,4,3,2,1	Non Graficabile
$n = 5$	$k = 7$	4,4,2,2,2	
$n = 5$	$k = 7$	4,3,3,2,2	
$n = 5$	$k = 7$	3,3,3,3,2	

Tabella C.6: Classi di equivalenza dei grafi di 5 vertici, 6 spigoli

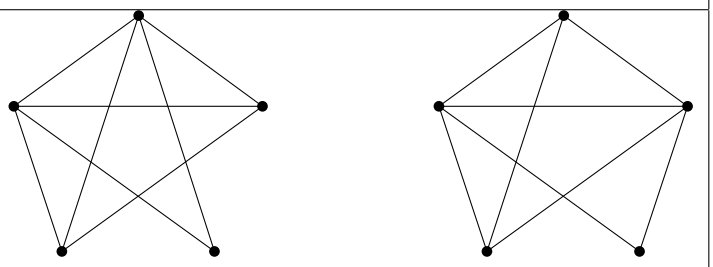
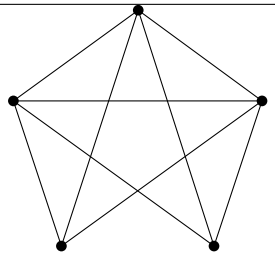
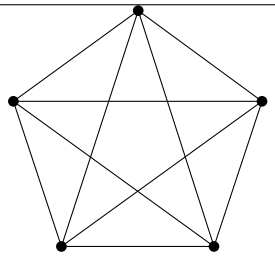
$n = 5$	$k = 8$	4,4,4,3,1	Non Graficabile
$n = 5$	$k = 8$	4,4,3,3,2	
$n = 5$	$k = 9$	4,4,4,4,2	Non Graficabile
$n = 5$	$k = 9$	4,4,4,3,3	
$n = 5$	$k = 10$	4,4,4,4,4	

Tabella C.7: Classi di equivalenza dei grafi di 5 vertici, da 8 a 10 spigoli

Bibliografia

[1] A. Iovinella, B. Scoppola, E. Scoppola, *Some Spin Glass Ideas Applied to the Clique Problem*, Journal of Statistic Physics, 126:895-915(21), 2007

[2] M. Viale, *Il Problema della Massima Clique: Teoria & Pratica*, tesi di dottorato in Fisica presso l'Università degli Studi Roma Tre (http://www.fis.uniroma3.it/dottorato_tesi/Viale_Viale M.pdf)

[3] Center for Discrete Mathematics and Theoretical Computer Science (`http://dimacs.rutgers.edu/`)

[4] T.H. Cormen, C.E. Leiserson, R.L. Rivest, C. Stein, *Introduzione agli algoritmi e strutture dati*, Seconda edizione, McGraw-Hill, 2005

[5] B. Bollobas, *Random Graph*, Cambridge University Press, 2001

[6] M. R. Garey, D. S. Johnson, *Computers and interactability. A guide to the Theory of NP-Completeness*, W.H. Freeman and Company, 1979

[7] P. Östergard, *A fast algorithm for the maximum clique problem*, Discrete Applied Mathematics 120 (2002) 197-207 (http://users.tkk.fi/∼pat/cliquer.html)

[8] M. Jerrum, *Large cliques elude the metropolis process.* Random Struct. Algorithms 3(4):347-359 (1992)

[9] S. Brin, L. Page, *The anatomy of a large-scale hypertextual Web search engine*, Computer Networks and ISDN Systems, 30, 1998