AR Analisi di Reti

2010/2011 M.Di Ianni

Clustering di reti wireless ad-hoc

Autori: P. Crescenzi, M. Di Ianni, F.Canovai, L.Frilli

In questa dispensa studiamo un particolare problema di clustering che trova applicazione nelle reti di sensori.

Come vedremo nel Paragrafo 4.1, in cui introdurremo il problema in generale, con il termine "clustering" si intende la definizione di sottoinsiemi di nodi in una rete che soddisfino un qualche criterio di vicinanza. Il caso di interesse in questa dispensa considera una rete corrispondente ad un grafo UDG (Unit Disk Graph) che, come ricordiamo, ben si presta a modellare una rete di sensori, tutti in grado di trasmettere i dati che rilevano con lo stesso raggio di trasmissione. Il problema di clustering che studiamo in questa dispensa consiste nel partizionare l'insieme di sensori nel minimo numero di sottoinsiemi tali che esiste un nodo in ciascun sottoinsieme cui tutti gli altri nodi contenuti nello stesso sottoinsieme possono trasmettere i propri dati. Come vedremo nel Paragrafo 4.2, tale problema è connesso alla questione del risparmio energetico dei sensori e quindi, in ultima analisi, della sopravvivenza stessa della rete. Nel Paragrafo 4.3 studieremo la complessità computazionale del problema decisionale soggiacente, provandone la **NP**-completezza. In conseguenza di tale risultato, nel Paragrafo 4.4 studieremo l'approssimabilità del problema di ottimizzazione e descriveremo un algoritmo $O(\ln n)$-approssimante, dove n è il numero di nodi della rete. Infine, nel Paragrafo 4.5 ci concentreremo nella ricerca di algoritmi approssimanti distribuiti, presentando un algoritmo randomizzato dovuto a Luby.

4.1 Problemi di Clustering

Il termine *Clustering*, o analisi dei cluster (dal termine inglese *cluster analysis* introdotto da Robert Tryon nel 1939), è un sinonimo per *decomposizione di un insieme di entità in "gruppi naturali"* con l'obiettivo di *classificare* tali entità ed individuare nella rete da esse definita proprietà rilevanti per un dato scopo. La popolarità del clustering va individuata nelle analogie che esso presenta con il meccanismo della percezione umana. Tipicamente, quando un oggetto (ad esempio, una rosa rossa) colpisce uno dei nostri sensi (ad, esempio, la vista) il meccanismo con cui riconosciamo tale oggetto opera, per così dire, per livelli decrescenti di astrazione: ci accorgiamo, prima di tutto, che l'oggetto è rosso, poi riconosciamo che si tratta di un fiore, ed infine lo classifichiamo come fiore di tipo rosa. Allo stesso modo, tendiamo naturalmente a raggruppare gli oggetti (e i concetti) che ci circondano in categorie: ad esempio, i libri sono classificati (in una libreria, ma anche nella nostra mente) rispetto al loro contenuto (storia, scienze, letteratura,

ecc.), ciascun gruppo è poi ulteriormente suddiviso in sottogruppi (ad esempio, letteratura medievale, rinascimentale, contemporanea, ecc.), e così via. Tipicamente, individuiamo una relazione piuttosto forte fra gli elementi interni allo stesso gruppo e una relazione di gran lunga più debole fra elementi che abbiamo classificato differentemente. Un gran numero di tecniche di ricerca ed elaborazione delle informazioni sono basate su queste considerazioni. Si pensi, ad esempio, alla ricerca di un libro su di un qualche argomento: innanzi tutto, vengono selezionati i libri che vertono su argomenti ad esso correlati, e poi essi (e solo essi) vengono esaminati in modo più approfondito. La struttura ricorsiva della suddivisione in argomenti e sotto-argomenti (ossia, la struttura del clustering dell'insieme dei dati) induce naturalmente una applicazione ricorsiva che ben si presta ad essere utilizzata come base per il progetto di un metodo di esplorazione e ricerca automatico.

Alla luce di quanto sinora esposto, dovrebbe essere chiaro che, quando parleremo di clustering, ci riferiremo ad un insieme di tecniche di analisi dei dati volte alla selezione e raggruppamento di elementi *omogenei* in un insieme. Ci sono due aspetti principali da tenere in considerazione in questo ambito: il primo è l'aspetto algoritmico, ossia come trovare tali decomposizioni e quanto è complesso trovarle, mentre il secondo è l'aspetto relativo alla qualità delle soluzioni trovate. Per quanto riguarda entrambi gli aspetti, sono state proposte numerose soluzioni, ossia, tecniche di decomposizione e misure della qualità delle decomposizioni prodotte, dipendentemente dalle (numerose) discipline in cui tale problema è stato studiato.

Tutte le tecniche di clustering si basano sul concetto di *distanza* tra due elementi. Nella teoria classica del clustering, gli elementi vengono rappresentati all'interno di opportuni spazi metrici, e la distanza fra coppie di elementi è, in qualche modo, correlata alla loro similarità. In effetti, la bontà delle analisi ottenute dagli algoritmi di clustering dipende molto dalla scelta della metrica, e quindi dal modo in cui viene calcolata la distanza.
Gli algoritmi di clustering raggruppano gli elementi sulla base della loro distanza reciproca, e, quindi, l'appartenenza o meno ad un cluster dipende da quanto l'elemento preso in esame è distante dall'insieme stesso: si richiede, cioè, che la coesione intra-cluster sia molto più elevata che la coesione inter-cluster. In quest'ottica, l'input ideale di un algoritmo di clustering consiste in un insieme di clique (o di strutture prossime a clique) disgiunte le une dalle altre (o collegate da "pochi" archi).
È naturalmente possibile parlare di modelli di clustering in cui viene richiesto che i cluster e le connessioni fra i cluster abbiano proprietà strutturali più complesse che non la semplice vicinanza (rispetto alla metrica utilizzata). Essi sono genericamente chiamati Role Assignments (assegnazioni di ruoli).

Le tecniche di clustering più utilizzate sono di tipo incrementale, ossia, raggruppano gli elementi di una rete in clusters partendo da un raggruppamento iniziale e procedendo, poi, ad una serie di successivi raffinamenti. Le tecniche di clustering incrementali possono essere di tipo bottom-up o top-down.
Un algoritmo incrementale di tipo bottom-up considera, all'inizio della computazione, ogni elemento della rete come un cluster a sé stante e, successivamente, provvede a fondere i cluster pi vicini. L'algoritmo continua il processo di fusione di cluster vicini fino ad ottenere un numero prefissato di cluster, oppure fino a quando la distanza minima tra i cluster non supera un certo valore di soglia.
Un algoritmo incrementale di tipo top-down considera inizialmente un unico cluster contenente tutti gli elementi della rete e, successivamente, inizia a suddividere il cluster in cluster di dimensioni inferiori. Il criterio che guida la suddivisione sempre quello di cercare di ottenere elementi omogenei. L'algoritmo procede fino a quando non ha raggiunto un numero prefissato di cluster.

Esistono vari criteri per classificare le tecniche di clustering comunemente utilizzate.
Un primo criterio è la possibilità che un elemento possa o meno essere assegnato a più clusters:

- *Clustering esclusivo*: ogni elemento può essere assegnato ad uno ed ad un solo gruppo. I clusters risultanti, quindi, non possono avere elementi in comune. Questo approccio è anche detto Hard Clustering.
- *Clustering non-esclusivo*: un elemento può appartenere a più cluster. Questo approccio è noto anche con il nome di Soft Clustering o Fuzzy Clustering (dal termine usato per indicare la logica fuzzy).

Un'altra suddivisione delle tecniche di clustering tiene conto del criterio utilizzato per suddividere lo spazio fra i vari cluster:

- *Clustering Partitivo* (detto anche k-clustering): per definire l'appartenenza di un elemento ad un cluster viene utilizzata una qualche funzione distanza da un punto rappresentativo del cluster (centroide, medioide, ecc. ...),

avendo prefissato il numero di gruppi della partizione risultato.

- *Clustering Gerarchico*: viene costruita una gerarchia di partizioni caratterizzate da un numero (de)crescente di gruppi, visualizzabile mediante una rappresentazione ad albero (dendrogramma), in cui sono rappresentati i passi di accorpamento/divisione dei gruppi.

4.2 Clustering, reti di sensori e risparmio energetico

Per consentire la raccolta da parte di una stazione base dei dati collezionati dai vari sensori, possiamo pensare di far comunicare ogni nodo con la stazione base. Questa semplice soluzione presenta un notevole svantaggio rappresentato dalla distanza tra la stazione base ed i vari sensori, in quanto per permettere a un sensore di comunicare con la stazione base è necessario che il raggio di comunicazione di ogni sensore sia superiore alla propria distanza da essa. Se aggiungiamo a questo anche il fatto che spesso tali reti sono costituite da sensori identici tra loro (e quindi con lo stesso raggio di comunicazione) arriviamo ad avere un raggio di comunicazione che è superiore alla distanza tra la stazione base ed il sensore ad essa più lontano, soluzione particolarmente dispendiosa in quanto il costo energetico speso in una comunicazione tra due sensori è pari al quadrato della loro distanza. La soluzione che possiamo perciò usare è quella di raggruppare i nodi in cluster e poi di eleggere un leader per ogni cluster; in questo modo i nodi non comunicheranno direttamente con la stazione base, bensì ogni nodo trasmetterà al leader del cluster di cui fa parte e sarà poi tale leader ad effettuare la comunicazione.

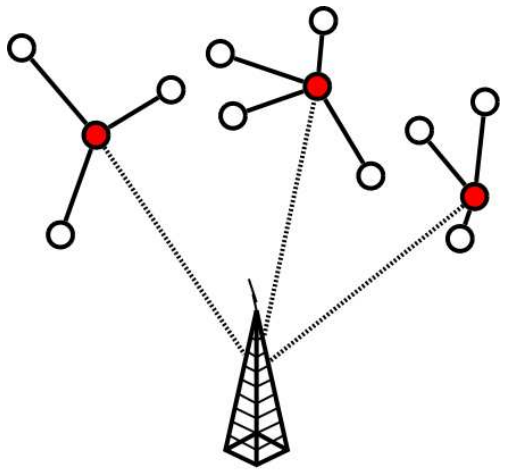

Figura 4.1: Solo i leader dei cluster trasmettono alla stazione base.

Possiamo anche pensare a una divisione in cluster come a un sistema di routing. Si avrà così una struttura simile per concetto all'idea di sottoreti IP. Distingueremo un routing locale effettuato fra nodi in un cluster ed un routing globale effettuato fra cluster. Ogni cluster avrà bisogno di un nodo leader che si occupi del routing. È inoltre possibile estendere ulteriormente la gerarchia raggruppando ulteriormente i cluster (super-cluster). Questa struttura a livelli ci permette di diminuire, oltre alla energia richiesta per le trasmissioni, la dimensione delle tabelle di routing dei vari sensori.

Prima di procedere, ricordiamo che tutti i sensori appartenenti ad una stessa rete trasmettono con il medesimo range. Pertanto, anche in questo caso, adottiamo il modello già utilizzato per l'analisi della connessione, e rappresentiamo la rete di sensori mediante uno Unit Disk Graph $G = (V, E)$: l'insieme dei nodi V è un insieme di punti del piano euclideo e due nodi u e v sono adiacenti (ossia, $(u, v) \in E$) se e solo se la loro distanza geometrica $d(u, v)$ è non maggiore di 1 (ossia, a meno di normalizzazioni, u e v sono adiacenti se e solo se possono ascoltarsi reciprocamente).

Per i nostri scopi, illustrati sopra, intendiamo suddividere G in cluster in modo tale che:

- ogni cluster abbia un nodo leader;
- ogni nodo abbia un leader nel suo vicinato;

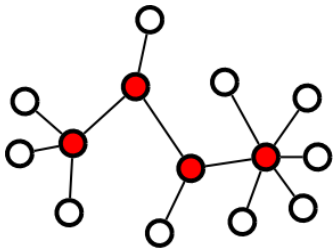

Figura 4.2: I leader dei vari cluster sono capaci di instradare correttamente i pacchetti.

- il numero di cluster sia limitato: questo significa che vogliamo utilizzare "pochi" nodi che trasmettono con range elevato (i cluster leaders).

Rispetto alla classificazione che abbiamo presentato nel precedente paragrafo, osserviamo che ci stiamo concentrando su un problema di clustering esclusivo. Inoltre, poiché ci interessa avere un numero limitato di cluster leader (ossia, un numero limitato di nodi che trasmettono con un range elevato), ci occuperemo di clustering partitivo. Infine, la metrica utilizzata sarà quella euclidea e la proprietà di coesione intra-cluster che richiederemo sarà quella per cui i nodi di ogni cluster inducano nel grafo di comunicazione G un albero radicato nel cluster leader e i cui soli archi siano quelli uscenti dalla radice (grafo a *stella*).

Affronteremo, innanzi tutto, tale problema dal punto di vista della teoria della complessità computazionale, dimostrandone la NP-completezza. In virtù di questo risultato, ci muoveremo, succesivamente, nella direzione della ricerca di soluzioni approssimate.

4.3 Minimum Dominating Set

Cominciamo con l'osservare che il problema di clustering di interesse in questa dispensa consiste nella ricerca di un insieme di nodi *cluster leaders* che si occupino di inviare i messaggi a nodi appartenenti al livello gerarchico superiore (Routing gerarchico). Tali leader devono avere la caratteristica di essere *vicini* a tutti i nodi del proprio cluster. Più formalmente, dato il grafo di comunicazione $G = (V,E)$ relativo all'insieme di sensori V, vogliamo individuare un sottoinsieme dei nodi $D \subseteq V$ tale che ogni nodo in $V - D$ sia collegato ad un nodo in D, ossia, vogliamo trovare un *insieme dominante* per il grafo di comunicazione indotto da G ed r:

Definizione 4.1(Dominating Set): *Dato un grafo $G = (V,E)$, un* insieme dominante *per G è un sottoinsieme D di V tale che*

$$\forall v \in V \setminus D) \; [\exists d \in D : \; (v,d) \in E].$$

Un algoritmo banale per calcolare un insieme dominante di un grafo è quello che definisce tutti i nodi come dominanti. Questa soluzione, tuttavia, non risponde alle esigenze che ci hanno condotto all'introduzione di questo problema in quanto, di fatto, non introduce alcuna gerarchia all'interno della rete. In effetti, per essere interessante, un insieme dominante deve essere *significativamente più piccolo* dell'insieme dei nodi della rete. Più in particolare, poiché il nostro obiettivo è quello minimizzare il numero dei leader in quanto essi sono soggetti ad un maggiore consumo energetico (dovendo effettuare un numero più elevato di comunicazioni e a distanza maggiore rispetto agli altri nodi), siamo interessati al calcolo dell'insieme dominante di *cardinalità minima*, ossia, alla soluzione del problema di ottimizzazione MINIMUM DOMINATING SET (in breve, MIN-DS).

Esempio. In Figura 4.3 è evidenziato un insieme dominante $D_1 = \{1,4,5\}$ del grafo. Tale insieme è dominante in quanto i nodi non marcati sono connessi ad almeno un nodo in D_1; infatti il nodo 2 è connesso ai nodi 1 e 4, il nodo 3 è connesso al nodo 5, il nodo 6 è connesso ai nodi 4 e 5 ed infine il nodo 7 è connesso al nodo 5. Se consideriamo però l'insieme $D_2 = \{2,6\}$ mostrato in Figura 4.4 abbiamo un nuovo insieme dominante e, dato che $|D_2| < |D_1|$, possiamo affermare che l'insieme D_1 non è il minimo insieme dominante del grafo. Inoltre, dato che non è possibile trovare un insieme dominante composto da un unico nodo, possiamo affermare che D_2 è il minimo insieme dominante del grafo.

Osserviamo ora che, per molte applicazioni (ad esempio, per potere istradare i messaggi fra i nodi leader nel caso in cui l'applicazione sia il progetto di uno schema di routing gerarchico), non è sufficiente trovare un minimo insieme

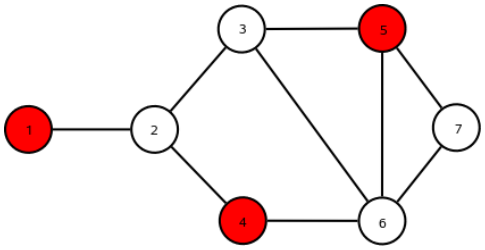

Figura 4.3: Un insieme dominante del grafo.

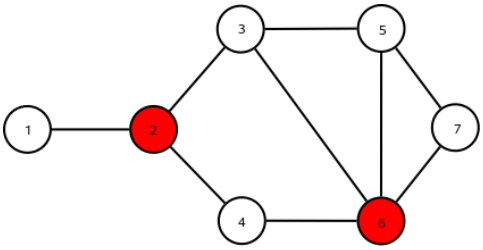

Figura 4.4: MDS del grafo.

dominante, ma è necessario che l'insieme selezionato sia *connesso*. Il problema corrispondente è detto MINIMUM CONNECTED DOMINATING SET (in breve, MIN-CDS). In Figura 4.5 mostriamo un esempio di minimo insieme dominante connesso.

Quello che vorremmo poter fare è, quindi, trovare il minimo insieme dominante (o minimo insieme dominante connesso) del corrispondente grafo. Per studiare la complessità di tali problemi, al solito, accanto ai problemi di ottimizzazione consideriamo i problemi decisionali DOMINATING SET e CONNECTED DOMINATING SET (in breve, DS e CDS) ad essi soggiacenti: dati un grafo G ed un intero k ci chiediamo se esiste un insieme dominante (eventualmente connesso) D per G tale che $|D| \leq k$.
Purtoppo il problema DS è un problema **NP**-completo e non è quindi possibile progettare un algoritmo per il calcolo del minimo insieme dominante che operi in tempo polinomiale a meno che non valga $\mathbf{P} = \mathbf{NP}$. Per dimostrare la completezza di DS introduciamo un altro problema tipico di teoria dei grafi.

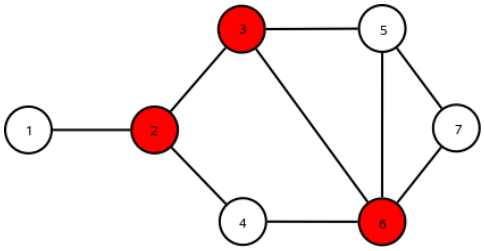

Figura 4.5: Un MCDS del grafo.

Definizione 4.2(Vertex Cover): *Dato un grafo $G = (V,E)$ un insieme S di vertici è detto essere un ricoprimento tramite vertici* (vertex cover) *per G se valgono:*

- $S \subseteq V$;
- $\forall$ *arco* $(u,v) \in E$, $u \in S$ *o* $v \in S$.

In questo caso, quello che cerchiamo è un insieme S di nodi tale che ogni arco di G ha almeno uno dei suoi due vertici inserito in S.
Il problema di ottimizzazione MINIMUM VERTEX COVER (in breve, MIN-VC) cerca un ricoprimento tramite vertici di un grafo che abbia cardinalità minima. Il corrispondente problema decisionale VERTEX COVER (in breve, VC) consiste nel chiedersi, dati un grafo G ed un intero k, se esiste un ricoprimento tramite vertici di G di cardinalità al più k.
Un esempio pratico di applicazione della soluzione del problema di ottimizzazione è il problema del dislocamento delle pattuglie di polizia, nel quale ci si chiede quale è il minimo numero di pattuglie da dislocare agli incroci in città in modo tale da poter controllare tutte le strade. In Figura 4.6 mostriamo un minimo ricoprimento tramite vertici del grafo visto nelle precedenti figure.

Teorema 4.1: *Il problema* VERTEX COVER *è **NP**-completo.*

Dimostrazione: Il problema è in **NP**: infatti se $G = (V,E)$ ammette un vertex cover di cardinalità k è deciso dall'algoritmo che, in $|V|$ passi non deterministici, costruisce un insieme V' (decidendo, ad ogni passo, se inserire o meno un nuovo nodo in V') e poi decide deteministicamente se $|V'| \leq k$ e se V' è un vertex cover per G.

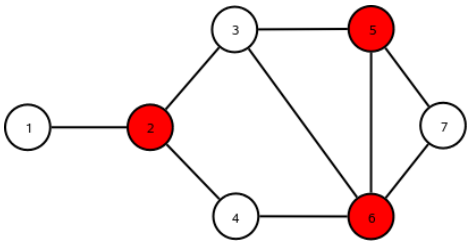

Figura 4.6: MVC del grafo.

Per quanto rigurada la completezza, utilizziamo una riduzione dal problema 3SAT in cui, dati un insieme di variabili booleane X ed una funzione $f : X \to \{vero, falso\}$ in forma normale congiuntiva con clausole di esattamente 3 letterali (forma 3CNF), ci chiediamo se f è soddisfacibile, ovvero, se esiste almeno un assegnazione $\phi : X \to \{vero, falso\}$ che soddisfa f.
Sia allora $f = C_1 \wedge C_2 \wedge \ldots \wedge C_m$ in cui $C_j = l_{j1} \vee l_{j2} \vee l_{j3}$ e $l_{ji} \in X \vee \bar{l_{ji}} \in X$, per $j = 1, \ldots, m$ e $i = 1,2,3$. La riduzione che associa ad f un grafo $G = (V,E)$ ed un intero k opera nel seguente modo. Per ogni variabile $x_i \in X$ creiamo due nodi nel grafo, etichettati con x_i ed $\bar{x_i}$ e collegati da un arco. Compito di questi nodi è rappresentare i due possibili valori di verità assegnabili alla variabile, *vero* e $falso$. Successivamente, per ogni clausola presente nella formula creiamo un grafo completo (ogni vertice è collegato a tutti i restanti vertici) di tre nodi che rappresentano i tre letterali presenti nella clausola. Infine colleghiamo questi nodi appena creati con il nodo rappresentante il medesimo letterale creato nel primo passo. Questa operazione di collegamento di nodi è l'unica parte della riduzione che dipende effettivamente da quali letterali sono presenti in ogni singola clausola. Un esempio di questa costruzione può essere visto in Figura 4.7.

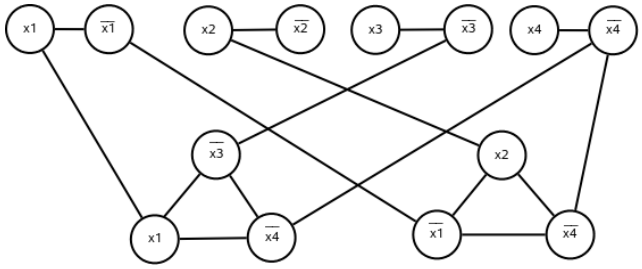

Figura 4.7: Il grafo per la formula $(x_1 \vee \bar{x}_3 \vee \bar{x}_4) \wedge (\bar{x}_1 \vee x_2 \vee \bar{x}_4)$.

A questo punto possiamo affermare che la formula da cui siamo partiti è soddisfacibile se e solo se esiste un ricoprimento tramite vertici del grafo ottenuto di cardinalità pari a

$$k = n + (|C_1| + \cdots + |C_m|) - m = n + 3m - m = n + 2m$$

dove con C_j indichiamo la j-esima clausola, m rappresenta il numero di clausole e n il numero di letterali.

Verifichiamo come un assegnamento ϕ che soddisfa f implichi l'esistenza di un ricoprimento del grafo della cardinalità specificata. Dato che $f(\phi(X)) = vero$, allora dovrà essere $\phi(C_j) = \phi(l_{j1}) \vee \phi(l_{j2}) \vee \phi(l_{j3}) = vero$ per ogni clausola C_j; quindi almeno un letterale l_{ji} all'interno della clausola sarà soddisfatto dall'assegnamento ϕ. Per ogni letterale l_{ji} della clausola inseriamo in V' il vertice x_h se $l_{ji} = x_h$ e $\phi(x_i) = vero$ o il vertice $\bar{x_i}$ se $l_{ji} = \bar{x_h}$ e $\phi(x_i) = falso$. Inoltre, per ogni $j = 1, \ldots, m$, scegliamo l_{ji} tale che $\phi(l_{ji}) = vero$ (tale letterale esista poiché ϕ soddisfa f) ed inseriamo in V' i nodi corrispondenti ai rimanenti due letterali di C_j. In questo modo i nodi in V' coprono tutti gli archi che collegano le coppie di nodi corrispondenti agli elementi di X, tutti gli archi che collegano i nodi corrispondenti ai letterali delle clausole, e tutti gli archi che collegano i letterali alle clausole. Ovvero, V' è un vertex cover per G. Valutiamo, ora, la cardinalità di V': innanzitutto, V' contiene n nodi che rappresentano il valore di verità assegnato ad ogni letterale e poi, per ogni clausola C_j, 2 ulteriori nodi selezionati per coprire il corrispondente grafo completo. Pertanto, $|V'| = n + 2m$.

Supponiamo, ora, che il grafo G ammetta un vertex cover $V' \subseteq V$ di cardinalità pari a $n + 2m$ e vediamo come è possibile ottenere da V' un assegnamento ϕ che soddisfi f. Osserviamo che, per ogni $x_i \in X$, V' deve contenere il nodo associato ad x_i oppure il nodo associato a $\bar{x_i}$ (altrimenti l'arco $(x_i, \bar{x_i})$ non sarebbe coperto): nel primo caso poniamo $\phi(x_i) = vero$, nel secondo $\phi(x_i) = falso$. Questa assegnazione soddisfa la formula perché tutti e tre gli archi che

collegano le clausole ai letterali sono coperti, ma solo due vertici nelle clausole possono appartenere a V' (altrimenti sarebbe$|V'| > n + 2m$). Di conseguenza, per ogni clausola C_j uno dei tre archi che collegano un letterale l_{ji} alla variabile corrispondente (x_h o $\bar{x_h}$) deve essere coperto da un nodo scelto fra quelli che corrispondono alle variabili: ossia, $x_h \in V'$ se $l_{ji} = x_h$ oppure $\bar{x_h} \in V'$ se $l_{ji} = \bar{x_h}$. Quindi ϕ soddisfa C_j. □

A questo punto possiamo dimostrare il risultato anticipato in precedenza.

Teorema 4.2: *Il problema* DOMINATING SET *è* ***NP****-Completo.*

Dimostrazione: Lasciamo come esercizio la prova dell'appartenenza del problema DS alla classe **NP**. Dimostriamone, invece, la completezza mediante una riduzione da VC. Osserviamo, innanzi tutto, che i due problemi sono simili, in quanto un insieme dominante copre i vertici di un grafo mentre un ricoprimento tramite vertici ne copre gli archi.

Prima di procedere con la riduzione, osserviamo che possiamo limitarci a considerare grafi privi di nodi isolati: infatti, poiché non è mai necessario includere un nodo isolato in un ricoprimento tramite nodi, un grafo $G = (V,E)$ contenente l'insieme $V' \subset V$ di nodi isolati ammette un ricoprimento con k nodi se e soltanto se il grafo $G' = (V - V', E)$ ammette un ricoprimento con k nodi.

Data un'istanza di VC $\langle G = (V,E), k\rangle$, definiamo la corrispondente istanza $\langle G_0 = (V_0, E_0), k_0\rangle$ di DS nel modo seguente:

- per ogni arco (v,w) del grafo G, V_0 contiene i vertici v, w ed il nuovo vertice vw: $V_0 = V \cup \{vw : (v,w) \in E\}$;
- per ogni arco (v,w) del grafo G, E_0 contiene gli archi (v,w), (vw,w) e (vw,v): $E_0 = E \cup \{(vw,w),(vw,v) : (v,w) \in E\}$;
- $k_0 = k$.

Questa trasformazione può essere calcolata in tempo in $O(|E| + |V|)$, ossia, in tempo polinomiale nella dimensione di G.

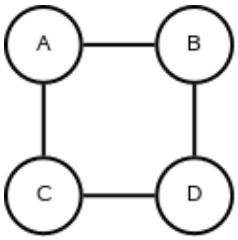

Figura 4.8: Grafo di cui vogliamo calcolare un MDS .

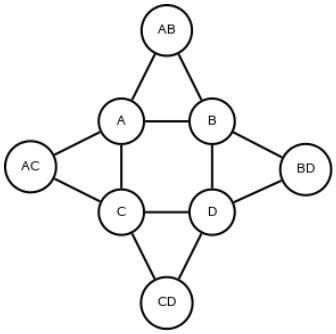

Figura 4.9: Grafo prodotto dalla riduzione.

Mostriamo ora che V_0 contiene un insieme dominante per G_0 di cardinalità k se e solo se V contiene un ricoprimento tramite vertici per G di cardinalità k.

($\Leftarrow$) Sia U un ricoprimento tramite vertici di G. Per ogni $v \in V_0 \setminus U$ sono possibili i due casi $v \in V$ oppure $v \in V_0 \setminus V$.

- Se $v \in V$ allora, non essendo v un vertice isolato, esiste in G un arco (v,w) appartenente a E. Poiché U è un ricoprimento tramite vertici e $v \notin U$, abbiamo $w \in U$, quindi v ha un vicino in U.
- Se v è un vertice generato a partire da un qualche arco $e = (u,w) \in E$ allora, poiché U è un ricoprimento tramite vertici, $u \in U$ o $w \in U$. Quindi v ha un vicino in U.

Abbiamo quindi che U è anche un insieme dominante per il grafo G_0.

($\Rightarrow$) Supponiamo che W sia un insieme dominante per G_0 di cardinalità $\leq k$. Se W contiene solo vertici appartenenti a V, allora tutti i vertici che abbiamo creato per rappresentare gli archi hanno un arco che li connette a un vertice in

1	$C := \emptyset$	
2	$F := \emptyset$	Usato in dimostrazioni successive
3	**while** $E \neq \emptyset$ **do begin**	
4	Scegli un arco $e = \{u, v\} \in E$	
5	**if** $u \notin C$ e $v \notin C$ **then begin**	Se e non è già coperto da un nodo in C
6	$C := C \cup \{u, v\}$	Aggiungi a C entrambi gli estremi di e
7	$F := F \cup \{e\}$	Aggiungi ad F l'arco e
8	**end**	
9	$E := E \setminus \{e\}$	Togli da E l'arco considerato
10	**end**	
11	**return** C	

Tabella 4.1: Algoritmo 2-approssimante per MVC.

W. Pertanto, ogni arco in E ha almeno un estremo in W che, dunque, è una valida copertura tramite vertici per G. Altrimenti, se W contiene un nodo di tipo vw, allora $(v,w) \in E$: in questo caso, rimuoviamo vw da W e inseriamo v al suo posto, se questo non già presente in W. L'insieme W così modificato è una valida copertura tramite vertici per G. □

È stato dimostrato che i problemi di trovare un insieme dominante o un insieme dominante connesso di data cardinalità sono **NP**-completi anche quando ci si limita a considerare grafi UDG [9]. Sarà quindi necessario utilizzare soluzioni alternative.

4.4 Algoritmi di approssimazione

In presenza di problemi **NP**-completi ci aspettiamo che non esista alcun algoritmo operante in tempo polinomiale che trovi una soluzione ottima. Quello che possiamo fare, allora, è cercare di progettare un algoritmo che operi in tempo polinomiale e che calcoli una soluzione *approssimata* del problema. Un algoritmo di questo tipo viene chiamato *algoritmo approssimante* (o *approssimato* o *di approssimazione*).

Definizione 4.3(Fattore di approssimazione): *Sia*

- *c il costo della soluzione fornita dall'algoritmo approssimato;*
- *c^* il costo della soluzione ottima.*

Chiamiamo fattore di approssimazione quella quantità $\rho(n)$ tale che

$$1 \leq \max\left\{\frac{c}{c^*}, \frac{c^*}{c}\right\} \leq \rho(n) \tag{4.1}$$

dove n è la dimensione dell'istanza del problema. Un algoritmo con fattore di approssimazione $\rho(n)$ è detto essere $\rho(n)$-approssimante.

Osserviamo meglio la disuguaglianza in 4.1. Per un problema di massimizzazione vale $0 < c \leq c^*$ e il rapporto c^*/c esprime il fattore entro il quale il costo di una soluzione ottima è maggiore del costo di una soluzione approssimata. Analogamente, per un problema di minimizzazione, $0 < c^* \leq c$ e il rapporto c/c^* esprime quanto il costo della soluzione approssimata sia più grande della soluzione ottima.

4.4.1 Algoritmo 2-approssimante per MVC

Presentiamo a titolo di esempio un algoritmo 2-approssimante per il problema MVC visto in precedenza. Consideriamo infatti l'algoritmo in Tabella 4.1 che ha in input un grafo $G = (V,E)$.

Vediamo come opera in dettaglio questo algoritmo. Innanzitutto notiamo come l'insieme C contenga il ricoprimento tramite vertici del grafo fornito in input, mentre F, che non svolge alcuna funzione all'interno dell'algoritmo, contiene tutti gli archi selezionati e servirà per dimostrare che l'algoritmo è 2-approssimante. L'algoritmo prende un insieme di

archi disgiunti tra loro (ovvero con nessun vertice in comune) ed inserisce i vertici di tali archi all'interno dell'insieme C; tale operazione è fatta nei punti da 5 a 7 ed infatti il controllo del punto 5 serve per scartare archi non disgiunti da quelli già inseriti in C.

Lemma 4.1: *Sia F l'insieme degli archi selezionati dall'algoritmo. Due archi qualsiasi di F non hanno estremi in comune, ovvero*

$$\forall e_1, e_2 \in F, e_1 \neq e_2 \Rightarrow e_1 \cap e_2 = \emptyset$$

Dimostrazione: Siano i due archi $e_1, e_2 \in F$ e supponiamo, senza perdere in generalità, che e_1 sia stato inserito prima di e_2. Al momento dell'inserimento di e_1 in F entrambi i suoi vertici sono stati inseriti in C (passo 6 dell'algoritmo). Se, per assurdo, e_1 ed e_2 avessero un estremo u in comune avremmo che $e_1 \cap e_2 = \{u\} \neq \emptyset$ e quindi, quando l'algoritmo analizza l'arco e_2, il controllo del passo 5 risulterebbe fallito in quanto $u \in C$ e l'arco e_2 non sarebbe aggiunto ad F. □

Teorema 4.3: *L'algoritmo considerato restituisce un ricoprimento tramite vertici che ha cardinalità al più due volte quella ottima.*

Dimostrazione: Ancora una volta sia C il ricoprimento tramite vertici restituito dall'algoritmo, mentre indichiamo con C^* un ricoprimento tramite vertici ottimo. Vogliamo dimostrare che $|C| \leq 2|C^*|$. Per arrivare a questo risultato consideriamo che ogni volta che l'algoritmo aggiunge un vertice ad F esso aggiunge anche i suoi due vertici a C, perciò dovrà essere

$$|C| = 2|F|$$

Dato che l'insieme degli archi selezionati è ovviamente un sottoinsieme degli archi del grafo abbiamo che $F \subseteq E$. Inoltre la soluzione ottima C^*, per definizione di ricoprimento tramite vertici, dovrà coprire tutti gli archi di F e, dato che per il lemma 4.1 ogni nodo $u \in C^*$ copre al più un arco di F deve valere

$$|C^*| \geq |F|$$

Unendo le due condizioni arriviamo a dimostrare

$$|C| = 2|F| \leq 2|C^*|$$

□

Esaminiamo un caso in cui l'algoritmo appena presentato restituisce una soluzione che è esattamente due volte quella ottima. In Figura 4.10 per ogni arco l'algoritmo selezionerà entrambi i suoi vertici mentre la soluzione ottima sarebbe prenderne soltanto uno. In un grafo bipartito completo la soluzione approssimata sarà sempre due volte quella ottima, includendo tutti i nodi del grafo.

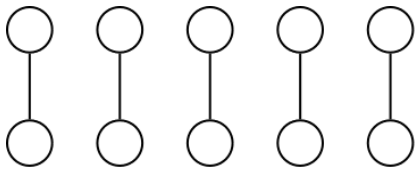

Figura 4.10: L'algoritmo seleziona tutti i nodi del grafo.

4.4.2 Il problema Minimum Set Cover e l'algoritmo di Chvatal

Definiamo, ora, un problema apparentemente molto diverso da DOMINATING SET che, tuttavia, potrà essere utilizzato per derivare per esso un agoritmo di approssimazione.
Iniziamo con la seguente definizione:

Definizione 4.4(SET COVER): *Sia X un insieme e sia $\mathscr{C}$ una collezione di sottoinsiemi di X. Un sottoinsieme S di $\mathscr{C}$ è un Set Cover per $\langle X, \mathscr{C} \rangle$ se gode delle seguenti proprietà:*

Input:	$X, \mathscr{C} \subseteq 2^X$.	
1	$S := \emptyset$;	
2	**while** $X \neq \emptyset$ **do begin**	
3	Seleziona un insieme $C_i \in \mathscr{C}$ di cardinalità massima;	
4	$S := S \cup \{C_i\}$	Aggiunge C_i ad S
5	$X := X \setminus C_i$	Elimina da X gli elementi presenti in C_i ovvero già coperti
6	$\mathscr{C} := \mathscr{C} \setminus \{C_i\}$	
7	$\forall C_j \in \mathscr{C}$: $C_j := C_j \setminus C_i$	Elimina gli elementi già coperti dai rimanenti sottoinsiemi.
8	**end**;	
9	**return** S	

Tabella 4.2: Algoritmo di Chvatal: un algoritmo approssimante greedy per Set Cover.

- $S \subseteq \mathscr{C}$;
- $\forall x_i \in X \; \exists C_i \in S$ *tale che* $x_i \in C_i$.

Dati un insieme X, una collezione $\mathscr{C}$ di suoi sottoinsiemi, ed un intero positivo k, il problema SET COVER (in breve, SC) consiste nel decidere se esiste un Set Cover per $\langle X, \mathscr{C} \rangle$ di cardinalità al più k. Analogamente, il problema MINIMUM SET COVER (in breve, MSC) consiste nell'individuare il Set Cover per $\langle X, \mathscr{C} \rangle$ di cardinalità minima.

Il teorema che segue prova che è poco probabile che sia possibile risolvere il problema MSC in tempo polinomiale.

Teorema 4.4: *Il problema SC è **NP**-Completo.*

Dimostrazione: Di nuovo, lasciamo come esercizio la prova di appartenenza alla classe **NP** del problema SC.
Per ciò che concerne la completezza, riduciamo polinomialmente il problema DS al problema SC. Consideriamo, allora, un'istanza $\langle G = (V,E), k \rangle$ di DS e deriviamo da essa un'istanza $\langle X, \mathscr{C}, k' \rangle$ di SC. Per ogni $v \in V$ definiamo l'insieme $C_v = \{v\} \cup \bigcup_{(v,u) \in E} u$, ovvero l'insieme che contiene il vertice v e tutti suoi vicini. Scegliamo, allora, $X = V$, $\mathscr{C} = \{C_v : v \in V\}$ e $k' = k$. Ovviamente, $\langle X, \mathscr{C}, k' \rangle$ può essere calcolata in tempo polinomiale nelle dimensioni di $\langle G = (V,E), k \rangle$.

Rimane da mostrare che esiste un ricoprimento $S \subseteq \mathscr{C}$ di cardinalità al più k per X se e solo se esiste un insieme dominante D di cardinalità k per G.

($\Rightarrow$) Assumiamo esista un ricoprimento $S \subseteq \mathscr{C}$ di X di cardinalità $\leq k$.
Poiché S ricopre X, allora l'insieme $D = \{u | C_u \in S\}$ è un insieme dominante per G perché, per ogni $v \notin D$, esiste $u \in D$ tale che $v \in C_u$, ossia, $(u,v) \in E$.
($\Leftarrow$) Assumiamo che esista un insieme dominante D di cardinalità $\leq k$ per G.
Consideriamo l'insieme $S = \{C_u | u \in D\}$. Poiché D è un insieme dominante per G, allora S è un ricoprimento per X in quanto, per ogni nodo v, $v \in D$ oppure esiste un arco $(u,v) \in E$ e $u \in D$: nel primo caso $C_v \in S$ e $v \in C_v$, nel secondo $C_u \in S$ e $v \in C_u$. □

Osserviamo che il precedente teorema dimostra qualcosa in più rispetto alla semplice **NP**-completezza del problema DOMINATING SET. In effetti, poiché la cardinalità dell'insieme dominante richiesto coincide con la cardinalità del sottoinsieme di $\mathscr{C}$ che copre X nell'istanza trasformata, il precedente teorema prova, sostanzialmente, che *qualunque algoritmo $\rho(n)$-approssimante per* MINIMUM SET COVER *è anche un algoritmo $\rho(n)$-approssimante per* MINIMUM DOMINATING SET.

In Tabella 4.2 riportiamo un algoritmo di approssimazione per MSC; tale algoritmo è conosciuto come *algoritmo di Chvatal* ed è di tipo greedy, ovvero, è un algoritmo che cerca una soluzione globale basandosi su scelte locali di volta in volta ottimali.

Vediamo adesso un esempio di funzionamento dell'algoritmo di Chvatal su input $X = \{a,b,c,d,e,f,g,h,i,j,k,l,m,n,o\}$ e $\mathscr{C} = \{C_1, \ldots, C_5\}$, dove:

- $C_1 = \{a,b,c,d,e,f\}$;

- $C_2 = \{g,h,i,j,k,l\}$;
- $C_3 = \{a,d,g,j,m\}$;
- $C_4 = \{b,e,h,k,n\}$;
- $C_5 = \{c,f,i,l,o\}$;

Una rappresentazione grafica della coppia $\langle X, \mathscr{C} \rangle$ è mostrata in Figura 4.11.

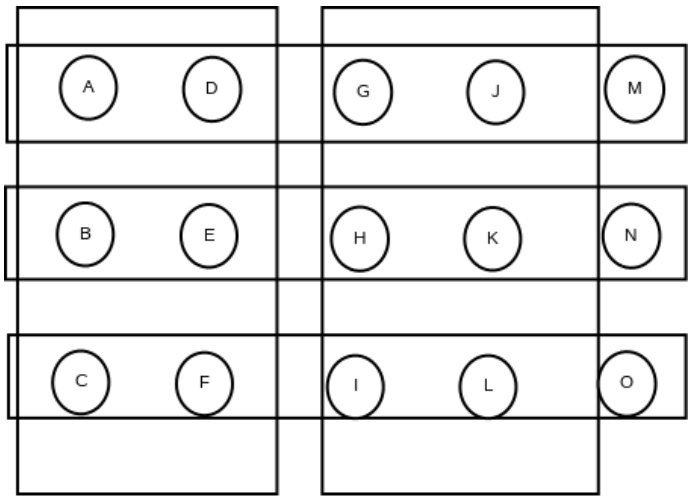

Figura 4.11: Esempio dell'algoritmo di Chvatal.

L'algoritmo di Chvatal seleziona come soluzione del problema MSC, nell'ordine, gli insiemi C_1, C_2, C_3, C_4 e C_5, ovvero calcola una soluzione S di cardinalità 5, mentre la soluzione ottima per questa istanza è $S^* = \{C_3, C_4, C_5\}$ e $|S^*| = 3$.

Vogliamo adesso dimostrare che l'algoritmo presentato calcola una soluzione S tale che $|S| \leq (\ln(n)+1)|S^*|$, dove S^* denota la soluzione ottima. Prima di procedere con la dimostrazione introduciamo alcuni concetti che ci saranno utili nel seguito.

Definizione 4.5(Numero Armonico): *Per ogni intero naturale $n > 0$ si definisce come n-esimo* numero armonico *la quantità*

$$H(n) = 1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n-1} + \frac{1}{n} = \sum_{k=1}^{n} \frac{1}{k} \tag{4.2}$$

Definiamo inoltre $H(0) = 0$.

Procediamo adesso con l'analisi dell'algoritmo. Il teorema che segue prova che esso non ha un fattore di approssimazione costante, bensì un fattore logaritmico nella dimensione del problema.

Teorema 4.5: *Sia S^* la soluzione ottima e sia S la soluzione calcolata dall'algoritmo di Chvatal. Allora vale*

$$|S| \leq (1 + \ln|X|)|S^*|$$

Dimostrazione: Assumiamo, senza perdita di generalità, che $C_1, \ldots, C_{|S|}$ sia l'ordine con cui i sottoinsiemi sono inseriti nella soluzione. Per ogni $i = 1, \ldots, |S|$ e per ogni $x \in C_i$ (ossia, per ogni $x \in X$ coperto per la prima volta alla i-esima iterazione) definiamo il costo c_x dell'elemento nella maniera seguente:

$$c_x = \frac{1}{|C_i \setminus (C_1 \cup C_2 \cup \cdots \cup C_{i-1})|} \tag{4.3}$$

Allora, il costo della soluzione S è pari a:

$$|S| = \sum_{x \in X} c_x \tag{4.4}$$

Definiamo ora la seguente quantità che mette in relazione la soluzione ottima S^* con la soluzione S mediante i valori c_x:

$$cost(S^*, S) = \sum_{C \in S^*} \sum_{x \in C} c_x \tag{4.5}$$

Per come abbiamo definito il valore $cost(S^*, S)$, se nelle intersezioni fra qualcuno degli insiemi scelti è presente qualche elemento, questo sarà contato due volte. Otteniamo perciò la disuguaglianza

$$cost(S^*, S) = \sum_{C \in S^*} \sum_{x \in C} c_x \geq \sum_{x \in X} c_x = |S| \tag{4.6}$$

Andiamo adesso a dimostrare che, per ogni $C \in S^*$,

$$\sum_{x \in C} c_x \leq H(|C|). \tag{4.7}$$

Sia allora C un elemento di S^* ed indichiamo con $u_i = |C \setminus (C_1 \cup \cdots \cup C_i)|$ la quantità di elementi scoperti in C dopo che gli insiemi $C_1, \ldots, C_i$ sono stati scelti dall'algoritmo. Il numero di elementi non ancora coperti al passo $i-1$ sarà, ovviamente, maggiore o uguale al numero degli elementi non ancora coperti al passo i, ed il valore $u_{i-1} - u_i$ rappresenta quindi il numero di elementi di C coperti nella i-esima iterazione. Abbiamo quindi che $u_0 = |C|$. Definiamo inoltre con k la prima iterazione per cui $u_k = 0$, ovvero quella in cui sono stati coperti gli ultimi elementi scoperti di C. Abbiamo così per l'equazione 4.3:

$$\sum_{x \in C} c_x = \sum_{i=1}^{k} \frac{u_{i-1} - u_i}{|C_i \setminus (C_1 \cup \cdots \cup C_{i-1})|} \tag{4.8}$$

Ma C_i è stato scelto dall'algoritmo in modo greedy, essendo quello che copriva più elementi rimasti scoperti di X. Pertanto dovrà essere:

$$|C_i \setminus (C_1 \cup \cdots \cup C_{i-1})| \geq |C \setminus (C_1 \cup \cdots \cup C_{i-1})| = u_{i-1} \tag{4.9}$$

Infatti, se la disuguaglianza in (4.9) non fosse valida, l'algoritmo avrebbe scelto C al posto di C_i alla i-esima iterazione.

Possiamo adesso dimostrare l'equazione (4.7) con una serie di manipolazioni algebriche. Infatti, dalle equazioni (4.8)

e (4.9), abbiamo:

$$
\begin{aligned}
\sum_{x\in C} c_x &= \sum_{i=1}^{k} \frac{u_{i-1}-u_i}{|C_i \setminus (C_1 \cup \cdots \cup C_{i-1})|} && (per\ la\ 4.8)\\
&\leq \sum_{i=1}^{k} \frac{u_{i-1}-u_i}{|C \setminus (C_1 \cup \cdots \cup C_{i-1})|} && (per\ la\ 4.9)\\
&= \sum_{i=1}^{k} \frac{u_{i-1}-u_i}{u_{i-1}} && (per\ la\ 4.9)\\
&= \sum_{i=1}^{k} \sum_{j=u_i+1}^{u_{i-1}} \frac{1}{u_{i-1}}\\
&\leq \sum_{i=1}^{k} \sum_{j=u_i+1}^{u_{i-1}} \frac{1}{j} && (j \leq u_{i-1})\\
&= \sum_{i=1}^{k} \left(\sum_{j=1}^{u_{i-1}} \frac{1}{j} - \sum_{j=1}^{u_i} \frac{1}{j} \right)\\
&= \sum_{i=1}^{k} (H(u_{i-i}) - H(u_i))\\
&= H(u_0) - H(u_k) && (serie\ telescopica)\\
&= H(|C|) - H(0) && (scelta\ di\ k\ tale\ che\ u_k = 0)\\
&= H(|C|)
\end{aligned}
$$

Dalle equazioni (4.6) e (4.7) possiamo quindi avvicinarci alla conclusione:

$$|S| = \sum_{x\in X} c_x \leq cost(S^*, S) = \sum_{C\in S^*} \sum_{x\in C} c_x \leq \sum_{C\in S^*} H(|C|) \leq \sum_{C\in S^*} H(|X|) = |S^*| H(|X|) \tag{4.10}$$

dove l'ultima disuguaglianza si ottiene in quanto $|C| \leq |X|$ da cui segue $H(|C|) \leq H(|X|)$. Essendo noto dall'analisi matematica che

$$H(n) \leq 1 + \ln n \tag{4.11}$$

otteniamo dalla 4.10 e 4.11

$$|S| \leq |S^*|(1 + \ln |X|) \tag{4.12}$$

come volevasi dimostrare. □

È inoltre stato dimostrato che non è possibile trovare un fattore di approssimazione inferiore a $ln(n)$ per gli algoritmi che approssimano MSC [14]. L'algoritmo presentato ha quindi prestazioni fra le migliori possibili per la risoluzione del problema.

Concludiamo questo paragrafo osservando come la combinazione della riduzione che prova la **NP**-completezza di SET COVER e l'algoritmo di Chvatal costituiscono un algoritmo $O(\ln n)$-approssimante per MINIMUM DOMINATING SET, illustrato in Tabella 4.3.

4.5 Approssimazioni distribuite del problema MDS

Nella sezione precedente abbiamo presentato un algoritmo approssimatnte (secondo un fattore *non costante*) per il problema MINIMUM DOMINATING SET. Tale algoritmo è di tipo *centralizzato*, ossia, presuppone l'esistenza di una autorità riconosciuta da tutti i nodi che prenda decisioni sulla conoscenza *globale* della rete. Tuttavia, siamo arrivati a studiare il problema MINIMUM DOMINATING SET per una sua applicazione nell'ambito delle reti wireless ad hoc. In

Input:	$G = (V,E)$.
1	in accordo alla riduzione mostrata nel Teorema 4.4 trasforma G in un insieme X ed una collezione $\mathscr{C}$ di suoi sottoinsiemi;
2	applica ad $\langle X, \mathscr{C} \rangle$ l'algoritmo di Chvatal: sia $S = \{C_{u_0}, \ldots, C_{u_k}\}$ il suo output;
3	**return** $D = \{u_0, \ldots, u_k\}$.

Tabella 4.3: Algoritmo approssimante per MDS derivato dall'algoritmo di Chvatal.

queto ambito è decisamente poco realistica l'assunzione di esistenza di una autorità centrale, in quanto caratteristica peculiare delle reti wireless ad hoc è proprio la loro capacità di essere operative in assenza di ogni tipo di infrastruttura. Per questa ragione, in questo paragrafo ci concentreremo sulla ricerca di algoritmi distribuiti per l'approssimazione del nostro problema.

4.5.1 Algoritmi randomizzati

Un *algoritmo randomizzato* (o *probabilistico*) è un algoritmo che utilizza un certo livello di randomizzazione come parte della sua logica. Un algoritmo randomizzato, generalmente, usa scelte probabilistiche per gestire il suo comportamento, cercando di ottenere buone prestazioni nel caso medio. Tipicamente, il tempo di esecuzione, l'output o entrambi saranno variabili casuali.

Tra le proprietà di un algoritmo randomizzato evidenziamo:

- lo stesso algoritmo randomizzato può fornire risultati diversi per il medesimo input;
- un algoritmo randomizzato può fornire un risultato errato. La probabilità di ottenere un risultato errato deve comunque essere ragionevolmente bassa;
- maggiore è il numero di volte che eseguo l'algoritmo, maggiore deve essere la *confidenza* della soluzione ottenuta (ossia, la probabilità che essa sia corretta);
- in generale un algoritmo randomizzato ha una complessità migliore (nel caso medio) dell'equivalente algoritmo deterministico.

Tipicamente, gli algoritmi randomizzati vengono classificati nelle due seguenti tipologie:

- algoritmi di tipo *Monte Carlo*: forniscono sempre una risposta, ma essa può anche essere errata. Ovviamente, la risposta di un algoritmo di questo tipo è esatta con alta probabilità mentre è errata con probabilità molto bassa. Più volte si esegue l'algoritmo maggiore sarà la probabilità che il risultato ottenuto sia corretto;
- algoritmi di tipo *Las Vegas*: non sempre forniscono una risposta, in quanto, con una certa probabilità, possono rispondere "non so". Comunque, quando forniscono una risposta, essa è corretta.

In Tabella 4.4 è presentato un algoritmo randomizzato per risolvere il problema MINIMUM DOMINATING SET. Questo algoritmo è molto rapido, ma il suo fattore di approssimazione è piuttosto elevato. L'algoritmo in questione è .

1	Ogni nodo sceglie un ID a caso;
2	Ogni nodo comunica il proprio ID ai vicini;
3	Ogni nodo elegge il vicino con ID massimo oppure sé stesso, se il proprio ID è maggiore di quello dei vicini;
4	I nodi eletti saranno i nodi dominanti.

Tabella 4.4: Un algoritmo distribuito randomizzato per MDS.

Vediamo in Figura 4.12 un esempio in cui questo algoritmo ottiene prestazioni scadenti. Supponiamo che in A ci siano $\sqrt{n}$ gruppi di $\sqrt{n}-1$ nodi che condividono la stessa posizione e che in B ci siano $\sqrt{n}$ nodi. Ogni nodo in B avrà quindi $2\sqrt{n}-1$ vicini (includendo se stesso): i $\sqrt{n}$ nodi di B e i $\sqrt{n}-1$ nodi in fronte ad esso presenti in A. La probabilità che un nodo in B elegga un nodo in A è quindi

$$\frac{\sqrt{n}-1}{2\sqrt{n}-1} \approx \frac{1}{2}.$$

Indichiamo, ora, con D i nodi eletti come dominanti dall'algoritmo e con X_A la variabile aleatoria che indica il numero di nodi contenuti in $D \cap A$ e, per ogni $b \in B$, con x_b la variabile aleatoria che vale 1 se b elegge un nodo in A, 0 altrimenti. Osserviamo che, poiché $X_A > \sum_{b \in B} x_b$, allora

$$VA(X_A) > \sum_{b \in B} VA(x_b) \approx \frac{1}{2}\sqrt{n}.$$

Pertanto, il numero atteso di nodi dominanti presenti in A dopo l'esecuzione dell'algoritmo è $\Omega(\sqrt{n})$, ma è evidente che il minimo insieme dominante ha cardinalità due: un nodo scelto in A e uno scelto in B sono infatti sufficienti a dominare l'intero grafo.

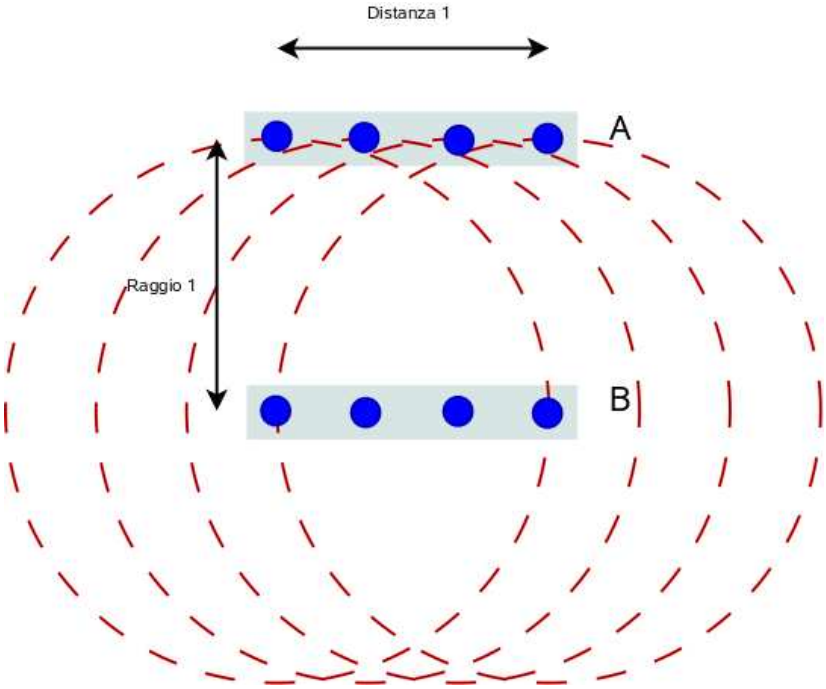

Figura 4.12: Un grafo UDG per il quale le prestazioni dell'algoritmo in Tabella 4.4 sono scadenti.

4.5.2 L'algoritmo di Luby

Per trovare un algoritmo approssimante più efficiente di quello proposto nel precedente paragrafo occorre, ancora una volta, ricorrere all'utilizzo di una soluzione per un differente problema. Introduciamo quindi il problema MINIMUM INDEPENDENT SET (in breve, MIS).

Definizione 4.6(Insieme indipendente): *Si dice insieme indipendente un sottoinsieme dei vertici di un grafo tali che nessuna coppia di elementi del sottoinsieme è connessa da un arco.*

Introduciamo, inoltre, i due concetti di insieme indipendente massimo e insieme indipendente massimale.

Definizione 4.7(Insieme indipendente massimo): *Un insieme indipendente costruito su un grafo $G = (V,E)$ si dice massimo se ha cardinalità maggiore o uguale a quella di ogni altro insieme indipendente costruibile su G.*

Definizione 4.8(Insieme indipendente massimale): *Un insieme indipendente massimale è un insieme indipendente che non è sottoinsieme di nessun altro insieme indipendente. In altre parole, è un insieme S tale che ogni arco del grafo ha almeno un'estremità non appartenente a S e ogni vertice non appartenente a S ha almeno un vicino in S. È quindi impossibile aggiungere un vertice a un insieme indipendente massimale mantenendo l'indipendenza dei vertici.*

È possibile osservare che, per definizione,

- un insieme indipendente massimo è anche massimale;
- un insieme indipendente massimale è un insieme dominante per G.

Relativamente al calcolo di un insieme indipendente massimo per un grafo, definiamo il problema decisionale soggiacente INDEPENDENT SET (in breve, IS): dati un grafo $G = (V,E)$ ed un intero positivo k, ci chiediamo se esiste $S \subseteq V$ tale che S è un insieme indipendente e $|S| \geq k$.

Il seguente teorema è un risultato ben noto in teoria della complessità computazionale.

Teorema 4.6: *Il problema IS è **NP**-Completo.*

Poiché l'ambito di applicazione dei nostri studi è quello delle reti wireless ad hoc, il modello di grafo cui siamo interessati è il modello *Unit Disk Graph* (in breve, UDG): l'insieme dei nodi di un grafo UDG è un insieme di punti del piano euclideo e due nodi sono connessi da un arco se e soltatnto se la loro distanza è minore o uguale ad 1. Relativamente a tale modello, valgono i seguenti risultati.

Lemma 4.2: *Sia $G = (V,E)$ un grafo UDG. Il vicinato di ogni vertice in V contiene al più 5 vertici indipendenti tra loro.*

Dimostrazione: Sia x un nodo di G e siano $u \in V$ e $v \in V$ due suoi vicini fra loro indipendenti: allora, $d(x,u) \leq 1$, $d(x,v) \leq 1$ e $d(u,v) > 1$. Allora, l'angolo $u\hat{x}v$ è tale che $u\hat{x}v > 60^\circ$.

Siano $x_1, \ldots, x_k$ punti nel vicinato di x, ordinati in senso antiorario a partire da una qualche direzione, fra loro indipendenti. Allora

$$x_1\hat{x}x_2 > 60^\circ, \ldots, x_{k-1}\hat{x}x_k > 60^\circ, x_k\hat{x}x_1 > 60^\circ,$$

ossia, $k60^\circ < 360^\circ$. Questo implica $k < 6$ e quindi $k \leq 5$ (si veda la Figura 4.15). □

Dal Lemma precedente discende quanto segue:

Teorema 4.7: *Siano $G = (V,E)$ un grafo UDG e D^* un insieme dominante minimo per G. Allora, ogni insieme indipendente massimale I per G è anche un insieme dominante per G tale che*

$$|I| \leq 5|D^*|.$$

Dimostrazione: Dal lemma 4.2 sappiamo che ogni nodo può dominare al più 5 vertici appartenenti a un insieme indipendente. Pertanto, la cardinalità di un qualsiasi insieme indipendente I sarà al più 5 volte quella del minimo insieme dominante. □

Come già osservato, un insieme indipendente massimale è anche un insieme dominante. Questo fatto, unito al risultato del Teorema 4.7 prova che, se disponessimo di un algoritmo distribuito polinomiale capace di calcolare un insieme indipendente massimale di un grafo UDG, allora avremmo anche un algoritmo 5-approssimante per il problema MINIMUM DOMINATING SET.

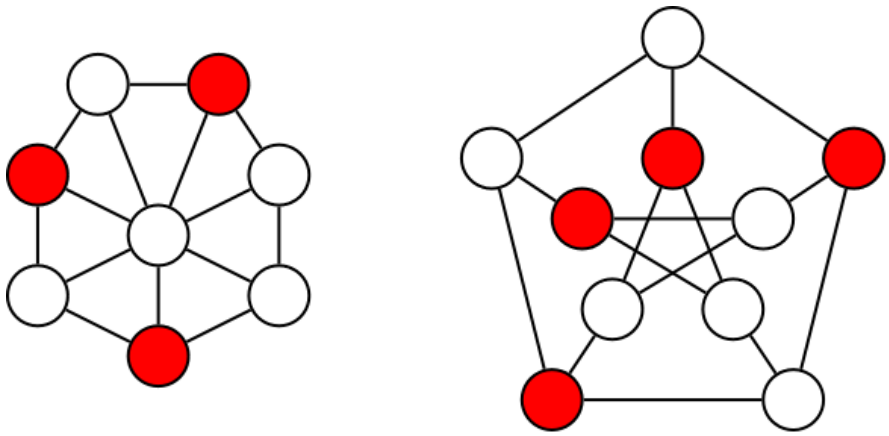

Figura 4.13: Esempi di insiemi indipendenti (i nodi colorati).

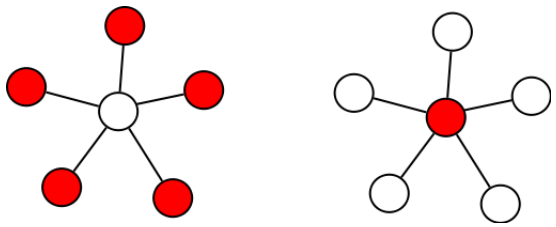

Figura 4.14: A sinistra, l'insieme indipendente massimo. A destra, un insieme indipendente massimale.

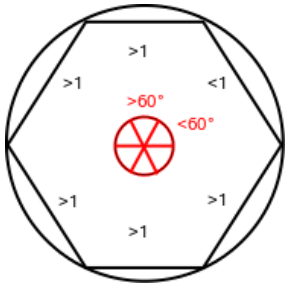

Figura 4.15: Non è possibile inserire 6 vertici a distanza > 1 fra essi.

Il resto di questo paragrafo è pertanto dedicato allo studio di un algoritmo distribuito in grado di individuare un insieme indipendente massimale in tempo polinomiale. L'algoritmo in questione, detto *algoritmo di Luby* (dal nome del suo ideatore), è descritto in Tabella 4.5. Esso è stato sviluppato per un'architettura chiamata PRAM (Parallel Random Access Machine): una macchina di tipo PRAM consiste di p processori identici $M_1, \ldots, M_p$ (RAM) che lavorano in maniera sincrona e possono compiere operazioni di lettura e scrittura su una memoria condivisa globale (Figura 4.16).

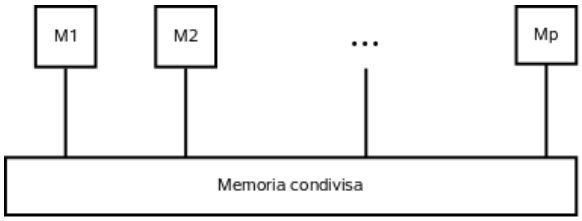

Figura 4.16: Schematizzazione di una PRAM.

Come osservabile in Tabella 4.5, l'algoritmo opera eseguendo una serie di *round* (ciascun round corrispondente ad una iterazione del ciclo **while**) che vengono eseguiti fino a quando l'insieme V' non è vuoto. In ogni round, viene calcolato un insieme indipendente I' sul grafo G' indotto in G dall'insieme dei nodi V'; al termine del round viene rimosso da V' l'insieme $I' \cup N(I')$, composto dai vertici in I' e dai vertici a questi vicini, e l'insieme I' viene aggiunto ad I per comporre l'insieme indipendente finale. Dunque, l'insieme indipendente restituito dall'algoritmo sarà dato dall'unione degli insiemi indipendenti calcolati in ogni round.
Vediamo, in particolare, come viene calcolato in ciascun round il contributo I' all'insieme indipendente I. Sia X l'insieme dei nodi candidati ad entrare in I'. In parallelo, ogni nodo $v \in V'$ si candida ad essere inserito in X con probabilità $\frac{1}{2d(v)}$, dove $d(v)$ indica il grado di v, ovvero il numero di vicini di v, favorendo quindi l'inserimento di nodi con grado minore. È qui che nell'algoritmo viene utilizzata la randomizzazione; si osservi come l'inserimento di un nodo in X avvenga indipendentemente da quello degli altri nodi. Osserviamo anche che, detto E' l'insieme degli archi di G indotti da V', i nodi isolati di $G' = (V', E')$ sono sempre aggiunti ad X, in quanto essi sono parte di ogni insieme indipendente massimale di G. Invece, l'inserimento in X dei nodi v_i con vicini in V' è deciso dall'estrazione di un numero casuale y: se $y < \frac{1}{2d(v_i)}$ allora v_i è effettivamente inserito in X. Successivamente, per ogni arco in E' si controlla che almeno uno dei suoi estremi non sia inserito in X, ovvero, che due candidati non siano vicini fra loro; in caso affermativo rimuoviamo da X il nodo con grado minore, scegliendo in modo arbitrario nel caso di nodi con grado uguale. Questo ci fa capire perché durante l'inserimento si siano favoriti i nodi di grado minore: in questo modo si cerca di vincolare X a contenere poche coppie di nodi nodi adiacenti. Gli elementi rimasti in X costituiscono il

Input:	grafo $G = (V,E)$.	
1	$I := \emptyset$	
2	$V' := V$	
3	**while** $V' \neq \emptyset$ **do begin**	fintanto che il grafo non è vuoto
4	$X := \emptyset$	
5	In parallelo per $\forall v \in V'$	
6	**if** $d(v) = 0$ aggiungi sempre v ad X	
7	**else** aggiungi v ad X con probabilità $\frac{1}{2d(v)}$	$d(v)$ = grado di v
8	$I' = X$	
9	In parallelo $\forall v \in X$ $w \in X$	
10	**if** $(v,w) \in E$ **then**	Per ogni arco con entrambi gli estremi in X
11	**if** $d(v) \leq d(w)$ **then** $I' := I' \setminus \{v\}$	togli quello di grado minore
12	**else** $I' := I' \setminus \{w\}$	
		I' è un insieme indipendente per G'
13	$Y := I' \cup N(I')$	$N(I')$ è il vicinato di I'
14	$V' := V' \setminus Y$	Elimina da V' l'insieme I' ed il suo vicinato $N(I')$
15	$I := I \cup I'$	Aggiungi la soluzione ad I
14	**end**	

Tabella 4.5: Algoritmo di Luby.

contributo del round corrente all'insieme indipendente I.
L'insieme indipendente I calcolato in tale modo è massimale: infatti, in ogni round vengono eliminati da V' (e quindi non saranno ulteriormente considerati come candidati a far parte di I) soltanto nodi che fanno parte di I' (e, quindi, sono già stati inseriti in I) oppure che sono vicini a nodi in I' (e, quindi, non sono indipendenti rispetto ai nodi già inseriti in I).
Osserviamo, infine che ogni round può essere eseguito in tempo costante utilizzando $O(|V|^2)$ processori.

Notiamo come sia possibile passare dall'implementazione parallela qui mostrata ad un'implementazione distribuita. Ciò è possibile mediante lo scambio di messaggi fra nodi vicini (ossia, collegati da un arco): in particolare, ciascun nodo invierà messaggi ai propri vicini al momento della sua candidatura, del suo inserimento nell'insieme indipendente o della sua sicura esclusione da esso.

Dimostriamo adesso che il tempo impiegato dall'algoritmo di Luby è effettivamente polinomiale. Più precisamente:

Teorema 4.8: *Il tempo medio impiegato dall'algoritmo di Luby è* $O(\log m)$*, dove* $m = |E|$.

La dimostrazione del precedente teorema procede attraverso l'utilizzo di una serie di risultati intermedi.
Nella loro prova, indicheremo con V'_i il valore della variabile V' all'inizio del round i (ossia, all'ingresso della i-esima iterazione del ciclo **while**) dell'algoritmo di Luby, e con $G'_i = (V'_i, E'_i)$ il grafo indotto in G da V'_i. Inoltre, X_i e I'_i denoteranno, rispettivamente, il valore della variabile X all'istruzione 8 e il valore della variabile I' al termine dellistruzione **if** (istruzioni 10-12) del round i dell'algoritmo di Luby.

Il primo dei risultati intermedi che concorreranno alla prova del Teorema 4.8 è la dimostrazione che, ad ogni iterazione dell'algoritmo, il numero di archi del grafo indotto dall'insieme V' è ridotto di almeno $\frac{1}{72}$ del numero di archi del grafo indotto dall'insieme V' calcolato nell'iterazione precedente. A questo scopo, dimostriamo prima il seguente lemma.

Lemma 4.3: *Per ogni round i dell'algoritmo di Luby e per ogni nodo v*

$$Pr(v \in I'_i) \geq \frac{1}{4d(v)} \tag{4.13}$$

Dimostrazione: Definiamo $L(v) = \{u \in N(v) | d(u) \geq d(v)\}$, ovvero, $L(v)$ è l'insieme dei vicini di v con grado maggiore o uguale al grado di v. Abbiamo allora che:

$$Pr(v \notin I'_i | v \in X_i) \leq Pr(\exists u \in L(v) \cap X_i | v \in X_i),$$

dove la disuguaglianza è dovuta al caso in cui u e v abbiano lo stesso grado e l'algoritmo scelga il nodo v. Utilizzando

lo Union Bound [1], segue che

$$\begin{aligned}
Pr(v \notin I'_i | v \in X_i) &\leq Pr(\exists u \in L(v) \cap X_i | v \in X_i) \\
&\leq \sum_{u \in L(v)} Pr(u \in X_i | v \in X_i) && (\textit{union bound}) \\
&= \sum_{u \in L(v)} Pr(u \in X_i) && (\textit{indipendenza a due a due}) \\
&= \sum_{u \in L(v)} \frac{1}{2d(u)} \\
&\leq \sum_{u \in L(v)} \frac{1}{2d(v)} && (d(u) \geq d(v)) \\
&\leq \frac{d(v)}{2d(v)} = \frac{1}{2} && (\textit{poiché } L(v) \subseteq N(v))
\end{aligned}$$

Otteniamo quindi la tesi. Infatti, per ogni round i,

$$\begin{aligned}
Pr(v \in I'_i) &= Pr((v \in I'_i \wedge v \in X_i) \vee (v \in I'_i \wedge v \notin X_i)) \\
&= Pr(v \in I'_i \wedge v \in X_i) + Pr(v \in I'_i \wedge v \notin X_i) \\
&= Pr(v \in I'_i | v \in X_i) \cdot Pr(v \in X_i) \geq \frac{1}{2} Pr(v \in X_i) = \frac{1}{4d(v)}
\end{aligned}$$

□

Notiamo come all'interno della dimostrazione non sia stata usata l'indipendenza totale, ma sia stata sufficiente la più debole indipendenza a due a due. Considerato un insieme di eventi $A_1 \ldots A_n$ possiamo definire gli n eventi *totalmente indipendenti* se vale $Pr(A_1 \cap A_2 \cap \ldots \cap A_n) = Pr(A_1) \cdot Pr(A_2) \cdot \ldots \cdot Pr(A_n)$, mentre definiamo gli eventi *a due a due indipendenti* se vale $Pr(A_i \cap A_j) = Pr(A_i) \cdot Pr(A_j)\ \forall i, j$.

Introduciamo adesso i seguenti concetti.

Definizione 4.9(Nodo buono): *Un nodo $v \in V$ è detto* buono *se*

$$\sum_{u \in N(v)} \frac{1}{2d(u)} \geq \frac{1}{6} \tag{4.14}$$

Altrimenti v è detto cattivo.

Intuitivamente un nodo sarà buono se ha molti vicini con grado piccolo. La nozione di "bontà" viene ora estesa agli archi:

Definizione 4.10(Arco buono): *Un arco $(u, v) \in E$ è detto* buono *se almeno un vertice fra u e v è buono. Altrimenti, l'arco è detto* cattivo.

Dimostriamo ora che, nell'istruzione 15 di un qualsiasi round i, un arco buono sarà rimosso da E'_i con probabilità pari almeno a $\frac{1}{36}$. Questa affermazione è conseguenza diretta del lemma seguente:

Lemma 4.4: *Per ogni nodo buono $v \in V$ e per ogni round i dell'algoritmo di Luby,*

$$Pr(v \in N(I'_i)) \geq \frac{1}{36}$$

.

Dimostrazione: Distinguiamo fra due casi:

[1] Per ogni insieme di eventi finito o numerabile la probabilità che almeno un evento accada non è maggiore della somma delle probabilità dei singoli eventi, ovvero $P\left(\bigcup_i A_i\right) \leq \sum_i P(A_i)$.

1. v ha un vicino u di grado al più 2. Formalmente $\exists u \in N(v) : \frac{1}{2d(u)} > \frac{1}{6}$;
2. ogni vicino u di v ha grado almeno 3. Formalmente $\forall u \in N(v) : \frac{1}{2d(u)} \leq \frac{1}{6}$.

Nel primo caso, per il lemma 4.3, abbiamo:

$$Pr(v \in N(I_i')) \geq Pr(u \in I_i') \geq \frac{1}{4d(u)} > \frac{1}{12} > \frac{1}{36}$$

Nel secondo caso, invece, notiamo come esista $M(v) \subseteq N(v)$ tale che

$$\frac{1}{6} \leq \sum_{u \in M(v)} \frac{1}{2d(u)} \leq \frac{1}{3} \tag{4.15}$$

Infatti, la disuguaglianza sinistra segue dal fatto che v è buono, mentre la disuguaglianza destra è conseguenza dell'ipotesi valida in questo caso (ogni vicino u di v ha grado almeno 3). Da questo otteniamo

$$\begin{aligned}
Pr(v \in N(I_i')) &\geq Pr(\exists u \in (M(v) \cap I_i')) \\
&\geq \sum_{u \in M(v)} Pr(u \in I_i') - \sum_{u,w \in M(v), u \neq w} Pr(u \in I_i' \wedge w \in I_i') \\
&\geq \sum_{u \in M(v)} \frac{1}{4d(u)} - \sum_{u,w \in M(v), u \neq w} Pr(u \in I_i' \wedge w \in I_i') && \textit{(Lemma 4.3)} \\
&= \sum_{u \in M(v)} \frac{1}{4d(u)} - \sum_{u,w \in M(v)} Pr(u \in I_i') Pr(w \in I_i') && \textit{(indipendenza a due a due)} \\
&\geq \sum_{u \in M(v)} \frac{1}{4d(u)} - \sum_{u \in M(v)} \sum_{w \in M(v)} \frac{1}{2d(u)} \frac{1}{2d(w)} && \textit{(}u, w \in X_i \textit{ non implica } u, w \in I_i', \\
& && \textit{ossia, } P(u \in I_i') \leq P(u \in X_i)\textit{)} \\
&= \sum_{u \in M(v)} \frac{1}{2d(u)} \left(\frac{1}{2} - \sum_{w \in M(v)} \frac{1}{2d(w)} \right) \\
&\geq \sum_{u \in M(v)} \frac{1}{2d(u)} \left(\frac{1}{2} - \frac{1}{3} \right) && \textit{(disuguaglianza 4.15 destra)} \\
&\geq \frac{1}{6}\frac{1}{6} = \frac{1}{36} && \textit{(disuguaglianza 4.15 sinistra)}
\end{aligned}$$

□

Abbiamo, dunque, delimitato inferiormente la probabilità che, se un arco è buono, esso venga eliminato durante una iterazione. Il prossimo lemma dimostra che il numero di archi buoni è, ad ogni iterazione, una frazione significativa del numero totale di archi residui in quella iterazione.

Lemma 4.5: *In ogni grafo, almeno la metà degli archi è buona.*

Dimostrazione: Per prima cosa orientiamo ogni arco verso l'estremo di grado maggiore, con una scelta arbitraria in caso di parità. Per ogni nodo v, denotiamo con $N_p(v)$ e con $N_s(v)$, rispettivamente, i vicini di v che precedono v ed i vicini di v che seguono v in questo orientamento. Osserviamo che $|N_p(v)| + |N_s(v)| = d(v)$.

Dimostriamo ora che ogni nodo cattivo $v \in V$ ha almeno il doppio di archi in uscita rispetto a quelli in entrata. Infatti, se fosse $|N_p(v)| \geq 2|N_s(v)|$, cioè, $\frac{|N_p(v)|}{d(v)} \geq \frac{1}{3}$, allora, poiché i nodi $u \in N_p(v)$ hanno grado più basso rispetto a v,

$$\sum_{u \in N(v)} \frac{1}{2d(u)} \geq \sum_{u \in N_p(v)} \frac{1}{2d(u)} \geq \sum_{u \in N_p(v)} \frac{1}{2d(v)} = \frac{1}{2} \frac{|N_p(v)|}{d(v)} \geq \frac{1}{6}. \tag{4.16}$$

Ma questo equivale a dire che v è buono, ottenendo così una contraddizione.

Perciò, deve essere $\frac{|N_p(v)|}{d(v)} < \frac{1}{3}$ (Figura 4.17).

Denotiamo, ora, con E_C l'insieme degli archi cattivi e dimostriamo l'esistenza di una funzione $f : E_C \to E \times E$ che associa ad ogni arco cattivo una coppia di archi *distinti* e tale che, per ogni $e, e' \in E_C$, $f(e) \cap f(e') = \emptyset$ se $e \neq e'$.
Sia allora $e = (u,v)$ un arco cattivo entrante in v (ricordiamo che u e v sono entrambi cattivi). Poiché v è cattivo ed (u,v) è un arco entrante in v, allora, in virtù di quanto provato all'inizio di questo lemma, esistono due archi distinti $e_1 = (v,v_1)$ e $e_2 = (v,v_2)$ $(v_1 \neq v_2)$ uscenti da v. Definiamo allora $f(e) = \{e_1, e_2\}$: osserviamo ora che $e_1 \neq e_2$ e che, poiché ciascun arco cattivo è associato mediante f a due archi uscenti dal suo estremo di grado maggiore, $e \neq e' \Rightarrow f(e) \cap f(e') = \emptyset$.

L'esistenza di f è sufficiente a provare l'asserto. Infatti, poiché, per definizione, $\cup_{e \in E_c} f(e) \subseteq E$ e $f(e_1) \cap f(e_2) = \emptyset$ per ogni $e_1 \neq e_2$, allora

$$|E| \geq |\cup_{e \in E_c} f(e)| = \sum_{e \in E_c} |f(e)| = \sum_{e \in E_c} 2 = 2|E_c|.$$

□

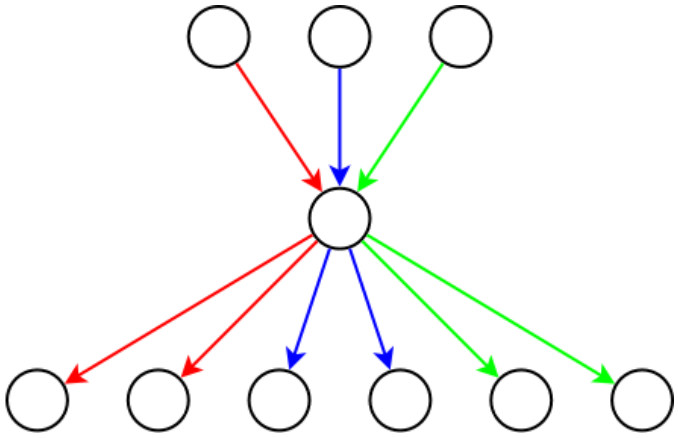

Figura 4.17: Per ogni arco in entrata ve ne sono almeno due in uscita.

Possiamo finalmente dimostrare l'affermazione fatta inizialmente circa il rapporto fra la dimensione del grafo G'_i e la dimensione di G'_{i-1}.

Teorema 4.9: *Sia $R_i = |E'_{i-1}| - |E'_i|$ il numero di archi rimossi durante l'iterazione $i-1$. Allora, per ogni i,*

$$VA(R_i) \geq \frac{1}{72}|E'_{i-1}|,$$

dove $VA(R_i)$ denota il valore atteso di R_i.

Dimostrazione: Per ogni arco $e \in E$, sia $R_i(e)$ la variabile aleatoria tale che $R_i(e) = 1$ se $e \in E'_{i-1} - E'_i$, $R_i(e) = 0$ altrimenti. Allora,

$$\begin{aligned} VA(R_i) &= \sum_{e \in E} VA(R_i(e)) \geq \sum_{e \text{ buono}} VA(R_i(e)) \\ &= \sum_{e \text{ buono}} 1 \cdot Pr(e \text{ è rimosso da } E'_{i-1}) \geq \sum_{e \text{ buono}} \frac{1}{36} && \textit{(Lemma 4.4)} \\ &\geq \frac{|E'_{i-1}|}{2}\frac{1}{36} && \textit{(Lemma 4.5)} \end{aligned}$$

□

Utilizziamo da qui in avanti le seguenti quantità:

- m: numero totale di archi del grafo;

- S_i: numero totale di archi rimossi nel corso delle iterazioni $1,\ldots,i$, ossia, $S_i = \sum_{j=1}^{i} R_j$.

Avremo perciò che $S_0 = 0$, $S_i \leq m$ e $S_{i+1} = S_i + R_{i+1}$. Possiamo riformulare il teorema 4.9 nel modo seguente:

$$VA(R_{i+1}|S_i = l) \geq \frac{1}{72}(m-l). \tag{4.17}$$

Osserviamo, inoltre, che

$$VA(R_{i+1}|S_i = l) = \sum_{k\in\mathbb{N}} k \cdot Pr(R_{i+1} = k|S_i = l). \tag{4.18}$$

Per concludere è necessario dimostrare tre ulteriori lemmi.

Lemma 4.6: *Per ogni i,*

$$VA(R_{i+1}) \geq \frac{1}{72}m - \frac{1}{72}VA(S_i) \tag{4.19}$$

Dimostrazione:

$$\begin{aligned}
VA(R_{i+1}) &= \sum_{k\in\mathbb{N}} k \cdot Pr(R_{i+1} = k) \\
&= \sum_{k\in\mathbb{N}} k \cdot Pr\left(R_{i+1} = k \wedge (\vee_{l\in\mathbb{N}} S_i = l)\right) \\
&= \sum_{k\in\mathbb{N}} \sum_{l\in\mathbb{N}} k \cdot Pr(R_{i+1} = k \wedge S_i = l) \\
&= \sum_{k\in\mathbb{N}} \sum_{l\in\mathbb{N}} k \cdot Pr(R_{i+1} = k|S_i = l) \cdot Pr(S_i = l) && (\textit{probabilità condizionata}) \\
&= \sum_{l\in\mathbb{N}} Pr(S_i = l) \sum_{k\in\mathbb{N}} k \cdot Pr(R_{i+1} = k|S_i = l) \\
&= \sum_{l\in\mathbb{N}} Pr(S_i = l) \cdot VA(R_{i+1}|S_i = l) && (\textit{equazione 4.18}) \\
&\geq \sum_{l\in\mathbb{N}} Pr(S_i = l) \cdot \frac{1}{72}(m-l) && (\textit{equazione 4.17}) \\
&= \frac{1}{72}m \sum_{l\in\mathbb{N}} Pr(S_i = l) - \frac{1}{72} \sum_{l\in\mathbb{N}} l \cdot Pr(S_i = l) \\
&= \frac{1}{72}m - \frac{1}{72}VA(S_i)
\end{aligned}$$

□

Lemma 4.7: *Per ogni i,*

$$VA(S_i) \geq m\left[1 - (1 - \frac{1}{72})^i\right]. \tag{4.20}$$

Dimostrazione: Dimostriamo il lemma per induzione su *i*.

Il caso $i = 0$ è evidentemente verificato.
Supposto l'asserto vero per ogni $j \leq i$, Dimostriamolo vero per $j = i+1$. Per $j = i+1$ abbiamo

$$\begin{aligned} VA(S_{i+1}) &= VA(S_i) + VA(R_{i+1}) \\ &\geq VA(S_i) + \frac{1}{72}m - \frac{1}{72}VA(S_i) && (\textit{lemma 4.6}) \\ &= \frac{1}{72}m + \left(1 - \frac{1}{72}\right) VA(S_i) \\ &\geq \frac{1}{72}m + m\left(1 - \frac{1}{72}\right)\left[1 - \left(1 - \frac{1}{72}\right)^i\right] && (\textit{passo induttivo}) \\ &= m\left[1 - \left(1 - \frac{1}{72}\right)^{i+1}\right]. \end{aligned}$$

□

Lemma 4.8: *Per ogni i,*

$$VA(S_i) \leq m - 1 + Pr(S_i = m) \tag{4.21}$$

Dimostrazione:

$$\begin{aligned} VA(S_i) &= \sum_{j=0}^{m} j \cdot Pr(S_i = j) \\ &\leq \sum_{j=0}^{m-1} (m-1) \cdot Pr(S_i = j) + m \cdot Pr(S_i = m) && (j \leq m-1) \\ &= (m-1)(1 - Pr(S_i = m)) + m \cdot Pr(S_i = m) \\ &= m - 1 + Pr(S_i = m) \end{aligned}$$

□

Siamo, finalmente, in grado di dimostrare il teorema 4.8 che, per comodità, enunciamo nuovamente.

Teorema 5.8: *Il tempo medio impiegato dall'algoritmo di Luby è $O(\log m)$, dove $m = |E|$.*

Dimostrazione: Dai due lemmi 4.7 e 4.8 si ha

$$\begin{aligned} Pr(S_i = m) &\geq VA(S_i) - m + 1 \\ &\geq m\left[1 - \left(1 - \frac{1}{72}\right)^i\right] - m + 1 \\ &= 1 - m\left(1 - \frac{1}{72}\right)^i. \end{aligned}$$

Quindi,

$$Pr(S_i < m) \leq m\left(1 - \frac{1}{72}\right)^i.$$

Scegliamo un valore k tale che $m(1 - \frac{1}{72})^k \leq 1$. Perciò, per un numero di iterazioni $i \geq k$ abbiamo

$$Pr(S_i < m) \leq m\left(1 - \frac{1}{72}\right)^i \leq \left(1 - \frac{1}{72}\right)^{i-k}$$

Definiamo ora una funzione $f : \mathbb{N} \to \{0, 1\}$ tale che

$$f(x) = \begin{cases} 1 & \text{se } x < m \\ 0 & \text{altrimenti,} \end{cases} \tag{4.22}$$

ed introduciamo la variabile casuale $L = f(S_0) + f(S_1) + f(S_2) + \dots$ che conta il numero di iterazioni dell'algoritmo di Luby. Segue da quanto appena dimostrato che

$$VA(f(S_i)) = Pr(S_i < m) \leq (1 - \frac{1}{72})^{i-k} \qquad \text{per } i \geq k, \tag{4.23}$$

da cui si ottiene

$$\begin{aligned} VA(L) &= \sum_{i \geq 0} VA(f(S_i)) \\ &\leq k + \sum_{i \geq k} VA(f(S_i)) \\ &\leq k + \sum_{i \geq k} \left(1 - \frac{1}{72}\right)^{i-k} && (\textit{da 4.23}) \\ &= k + 72 && (\textit{serie geometrica di ragione } |x| < 1.) \end{aligned}$$

Osserviamo, infine, che abbiamo scelto k in modo da avere $m(1 - \frac{1}{72})^k \leq 1$; questo significa che $k \in O(\log m)$ ed il teorema è così dimostrato. □

Riferimenti bibliografici

[1] Posenato R., *Insegnamento di algoritmi avanzati: Algoritmi Probabilistici*, appunti del corso di algoritmi avanzati del corso di laurea specialistica in informatica presso l'università di Verona.

[2] Pasquale F., *Un algoritmo 2-approssimante per Min Vertex Cover*, 2007.

[3] Malucelli F., *Algoritmi approssimati*, 2005, appunti del corso di progetto e analisi di algoritmi del corso di laurea specialistica in ingegneria informatica presso il politecnico di Milano.

[4] Luby M., *A simple parallel algorithm for the maximal independent set problem*, 1985, Department of Computer Science, University of Toronto.

[5] Wu W., Du H., Jia X., Li Y., Huang S.C.H., *Minimum connected dominating sets and maximal independent sets in unit disk graphs*, 2005.

[6] Crescenzi P., *Algoritmi per Reti di Calcolatori: Clusterizzazione*, 2009, appunti del corso di algoritmi per reti di calcolatori del corso di laurea specialistica in informatica presso l'università di Firenze.

[7] Kleinberg J, Tardos E., *Algorithm Design*, 2005, Addison Wesley.

[8] Nowotka D., *Algorithmik*, 2009, appunti del corso di Algoritmica presso l'università di Stoccarda.

[9] Lichtenstein D., *Planar formulae and their uses*, 1982, SIAM Journal on Computing Vol. 11.

[10] Safra M., Appunti del corso di complessità computazionale presso l'università di Tel Aviv.

[11] Evans P., Appunti del corso di algoritmi presso l'università del New Brunswick.

[12] Luby M., Widgerson A., *Pairwise Independence and Derandomization*, 2005, Foundations and Trends in Theoretical Computer Science.

[13] Dey T.K., Prabhavalkar R., *The Set-Covering Problem*, 2009, appunti del corso di algoritmi avanzati presso Ohio State University.

[14] Feige U., *A treshold of ln n for approximating set cover*, 1998, Journal of the ACM.

[15] Wan V.K., Ba K.D., *Set Cover and Application to Shortest Superstring*, 2005, appunti del corso di algoritmi presso Dartmouth College.

[16] Garey M.R., Johnson D.S., *Computers and Intractability*, 1979, W. H. Freeman.

[17] Cormen T.H., Leiserson C.E., Rivest R.L., *Introduction to Algorithms*, 2001, McGraw-Hill.

[18] Parthasarathy S., Gandhi R., *Distributed Algorithms for Coloring and Domination in Wireless Ad Hoc Networks*, 2004.