Introduzione alla teoria della **NP**-completezza

2.1 Introduzione informale alla teoria della complessità

Obiettivo di queste pagine è quello di fornire una introduzione informale alla teoria della complessità.

La teoria della complessità si propone di studiare la complessità *intrinseca* dei problemi: dato un problema, la sua complessità coincide con la complessità dell'algoritmo più efficiente che lo risolve. Per chiarire tale concetto, è necessario specificare in maggiore dettaglio alcuni dettagli.
La complessità di un algoritmo è definita essere come la quantità di una certa risorsa (che in questa dispensa sarà sempre il tempo di calcolo) necessaria affinché l'esecuzione dell'algoritmo con un dato input termini. Naturalmente, il tempo necessario ad un algoritmo per portare a termine il proprio compito deipende dalla dimensione dell'input sul quale deve operare. Se abbiamo scritto un algoritmo in grado di decidere se un dato numero intero è composto (cioè, non è primo), possiamo ragionevolmete aspettarci che l'esecuzione dell'algoritmo sia molto più breve quando il suo input è 123 che non quando è dell'ordine di 10^7. Questo significa che intendiamo esprimere la complessità di un algoritmo (e, dunque, di un problema) come una *funzione della dimensione dell'input*. Informalmente, la dimensione del generico input x di un algoritmo è il numero di bit necessario a rappresentarlo, e viene indicata con $|x|$.
Nell'esempio di problema che abbiamo appena considerato veniva richiesto all'algoritmo di *prendere una decisione circa una proprietà che l'input doveva soddisfare*. Questo non è casuale: la teoria della complessità si occupa di *problemi decisionali*, ossia, problemi le cui uniche risposte possono essere *sì* oppure *no*. Più in dettaglio, un problema decisionale è caratterizzato da un insieme di *istanze*, ove ciascuna istanza corrisponde ad un insieme di dati per il problema, e da una serie di condizioni che devono essere soddisfatte da una istanza: se una istanza soddisfa dette condizioni essa è detta *istanza sì* per il problema, altrimenti è detta *istanza no*.

Esempio. Nel caso del problema NUMERO COMPOSTO, una istanza è costituita da un numero intero; una istanza è una istanza sì se non è un numero primo (ossia, ha almeno un divisore diverso da sé stesso e da 1). □

I problemi decisionali *trattabili computazionalmente* sono quei problemi che sono decisi da un algoritmo la cui esecuzione su input x richiede tempo polinomiale in $|x|$. La classe dei problemi trattabili computazionalmente è denotata con **P**.

La classe **NP** è stata introdotta considerando un classe di algoritmi più potenti di quelli che possono essere eseguiti sui normali calcolatori, e, dunque, un modello di calcolo più potente del modello di Von Neumann. Il modello di calcolo in questione ha la possibilità di valutare in parallelo un numero di possibilità non costante, ma dipendente dalle dimensioni dell'input. Più in particolare, un algoritmo non deterministico utilizza la seguente istruzione

scegli $y \in S(x)$;

in cui x è l'input e $S(x)$ è l'insieme delle *soluzioni possibili* per l'istanza x del problema, ossia, di tutti e soli gli elementi che potrebbero indurre l'algoritmo a rispondere sì. Nel caso in cui vogliamo decidere se x è un numero composto, ad esempio, $S(x) = \{y \in \mathbb{N} : 0 < y \leq x/2\}$: infatti x è composto se e soltanto se esiste $y \in S(x)$ tale che x è divisibile per y. Una soluzione possibile che induce l'algoritmo a rispondere sì è detta *soluzione effettiva*.
L'istruzione non deterministica può essere vista come una sorta di oracolo che, all'interno dell'insieme $S(x)$, è capace, in tempo in $O(\log|S(x)|)$, di scegliere proprio un elemento y che permette all'algoritmo di rispondere sì, se un tale y esiste[1]. Più in dettaglio, se esiste $y_e \in S(x)$ che è soluzione effettiva per il problema allora l'oracolo sceglie y_e, altrimenti sceglie un qualunque $y \in S(x)$: successivamente alla risposta dell'oracolo, l'algoritmo esegue una serie di controlli (deterministici) in cui *verifica* se la risposta dell'oracolo è una soluzione effettiva oppure no, decidendo, conseguentemente la risposta.
Tuttavia, ai fini della teoria della complessità, è più conveniente vedere l'istruzione non deterministica come una istruzione che genera $|S(x)|$ computazioni parallele: il blocco di istruzioni che segue l'istruzione **scegli** viene eseguito, in parallelo, per ogni $y \in S(x)$. L'algoritmo risponde sì non appena una delle computazioni (deterministiche) parallele risponde sì; se, invece, nessuna delle computazioni deterministiche risponde sì, allora l'algoritmo risponde no quando tutte le computazioni paralele hanno risposto no.

La classe **NP** è la classe dei problemi decisionali che sono decisi da un algoritmo non deterministico in tempo polinomiale nella dimensione dell'input.

Esempio. Sia $G = (V,E)$ un grafo. Un sottoinsieme V' dell'insieme dei nodi V è un *ricoprimento tramite nodi* (o *Vertex Cover*) per G se ogni arco in E ha almeno un estremo in V':

$$\forall (u,v) \in E : \; u \in V' \; \vee \; v \in V'.$$

Il problema VERTEX COVER (in breve, VC) consiste nel decidere, dati un grafo $G = (V,E)$ ed un intero k, se esiste per G un ricoprimento tramite nodi di cardinalità k.
Osserviamo che, in questo caso, dati un grafo $G = (V,E)$ ed un intero positivo k, $S(G,k) = \{V' \subseteq V : |V'| = k\}$. Un algoritmo non deterministico che decide VC è allora il seguente:

1) **scegli** $V' \subseteq V$ tale che $|V'| = k$;
2) esito $\rightarrow$ vero;
3) **for** $((u,v) \in E)$ **do**
4) **if** $(u \notin V' \; wedge \; v \notin V')$ **then** esito $\rightarrow$ falso;
5) output: esito;

Poiché $|S(G,k)| \leq 2^{|V|}$ (in quanto V contiene $2^{|V|}$ sottoinsiemi), allora l'istruzione 1) richiede tempo in $O(|V|)$; inoltre, il ciclo **for** alla linea 3) viene iterato $|E|$ volte e, in ciascuna iterazione, viene eseguito l'**if** alla linea 4) che richiede tempo in $O(k)$. Poiché $k \leq |V|$, questo prova che VC è un problema in **NP**. □

Da quanto detto sino ad ora, e dall'esempio sopra riportato, possiamo affermare che, affinché un problema sia in **NP**, è necessario che, per ogni input x, la cardinalità dell'insieme delle soluzioni possibili sia in $O(2^{p(|x|)})$, per qualche polinomio p e che, inoltre, sia possibile verificare deterministicamente che una soluzione possibile è anche soluzione effettiva in tempo polinomiale.

Osserviamo, a questo punto, che un algoritmo (deterministico) è un particolare algoritmo non deterministico in cui l'istruzione non deterministica **scegli** non viene mai utilizzata. Dunque, $\mathbf{P} \subseteq \mathbf{NP}$. Sfortunatamente, anche se sembra ragionevole che l'utilizzo dell'istruzione non deterministica porti alla definizione di algoritmi strettamente più efficienti di quelli che non la utilizzano, non si è, sino ad ora, riusciti a dimostrare né l'inclusione stretta fra le due classi **P**e **NP** né la loro coincidenza. In effetti, tale questione è uno dei problemi aperti del millennio ed è stato anche offerto un premio cospicuo per la sua soluzione. È, comunque, generalmente ritenuta valida la seguente congettura:

[1]Osserviamo che $\log|S(x)|$ bit sono richiesti per rappresentare gli elementi di $S(x)$. Il tempo $O(\log|S(x)|)$ è, dunque, il tempo necessario a scegliere tutti i bit di $y \in S(x)$.

Congettura. $\mathbf{P} \neq \mathbf{NP}$.

Assunta la congettura $\mathbf{P} \neq \mathbf{NP}$, è naturale chiedersi quali sono quei problemi che possono essere candidati ad essere collocati in $\mathbf{NP} - \mathbf{P}$. A questo scopo, è stata introdotta la teoria della **NP**-completezza. Intuitivamente, un problema in **NP** è *completo* se l'esistenza di un algoritmo polinomiale per la sua soluzione implica l'esistenza di un agoritmo polinomiale per la soluzione di *ogni* problema in **NP**: dunque, se un problema **NP**-completo fosse contenuto in **P**allora ogni problema in **NP** sarebbe contenuto in **P**, ossia, si avrebbe $\mathbf{P} = \mathbf{NP}$.
Ma cosa significa che "l'esistenza di un algoritmo polinomiale per la soluzione di un problema A in **NP** implica l'esistenza di un agoritmo polinomiale per la soluzione di ogni problema in **NP**"? Per chiarire questo concetto, è necessaria la seguente definizione.

Definizione 2.1 (Riduzione polinomiale): *Un problema decisionale A è riducibile polinomialmente ad un problema decisionale B se esiste una funzione $f : I_A \to I_B$, dove I_A è l'insieme delle istanze di A e I_B è l'insieme delle istanze di B tale che:*

- *$x \in I_A$ è una istanza sì per il problema A se e soltanto se $F(x) \in I_B$ è una istanza sì per il problema B;*
- *esiste un algoritmo che, per ogni x, calcola $f(x)$ in tempo polinomiale in $|x|$.*

Se A è riducibile polinomialmente a B, scriviamo $A \leq B$. Osserviamo che, se un problema decisionale A è riducibile polinomialmente ad un problema decisionale B ed esiste un algoritmo polinomiale per decidere B, allora tale algoritmo può essere combinato con la funzione che riduce A a B per ottenere un algoritmo polinomiale per A. Infatti, sia $A \leq B$, sia f la funzione che testimonia la riducibilità, e sia T_B l'algoritmo polinomiale che decide B; allora, dalla definizione di riducibilità, segue che il seguente è un algoritmo polinomiale che decide A.

Input: $x \in I_A$.

1. calcola $f(x)$;
2. esegui $T_B(f(x)$;
3. se $T_B(f(x)$ ha risposto sì allora rispondi sì, altrimenti rispondi no.

Definizione 2.2 (NP-completezza): *Un problema $A \in$ **NP** è **NP**-completo se ogni altro problema $B \in$ **NP** è riducibile polinomialmente ad A:*

$$\forall B \in \mathbf{NP} \ : \ B \leq A.$$

Da quanto detto sino ad ora, è chiaro che, se A è **NP**-completo, allora l'esistenza di un algoritmo polinomiale per A implicherebbe l'esistenza di un algoritmo polinomiale per ogni problema in **NP**: ossia, l'esistenza di un algoritmo polinomiale per A implicherebbe $\mathbf{P} = \mathbf{NP}$. Dunque, i problemi **NP**-completi sono i candidati ad appartenere a $\mathbf{NP} - \mathbf{P}$. Ma esistono problemi **NP**-completi? La risposta a tale domanda è l'oggetto della prossima sezione.

2.2 Problemi NP-completi

L'esistenza di un problema **NP**-completo è stata dimostrata nel 1971 da Stephen Cook. La dimostrazione di Cook mostra come è possibile trasformare l'esecuzione di un qualunque algoritmo non deterministico su un qualsiasi input x, che avviene in tempo polinomiale in $|x|$, in una funzione booleana φ in modo tale che l'algoritmo risponde s` se e soltanto se esiste una assegnazione di verità per le variabili che compaiono in φ che rende vera φ. Vediamo in maggior dettaglio quanto dimostrato da Cook.
Sia $X = \{x_1, x_2, \ldots, x_n\}$ un insieme di variabili booleane;una funzione φ nelle variabili in X è detta essere in *forma congiuntiva normale* se ha la seguente forma:

- φ è una *congiunzione di clausole*, ossia, $\varphi = c_1 \wedge c_2 \wedge c_m$,
- ciascuna clausola c_j è della forma $l_{j_1} \vee l_{j_2} \vee \ldots \vee l_{j_{h_j}}$ dove, per $i = 1,2,\ldots,h_j$, l_{j_i} o la sua negazione appartiene ad X (ciascun l_{j_i} è chiamato *letterale*).

Nel problema SODDISFACIBILITÀ (in breve, SAT) sono dati un insieme di variabili booleane $X = \{x_1, x_2, \ldots, x_n\}$ ed una funzione booleana φ nelle variabili in X in forma congiuntiva normale e ci si domanda se esiste una assegnazione di verità per le variabili in X che renda vera φ.

Teorema 2.1: **(Cook, 1971)** *Il problema SAT è **NP**-completo.*

Una volta trovato un capostipite nella famiglia dei problemi **NP**-completi (quale è SAT), per dimostrare la **NP**-completezza di un problema è sufficiente

- dimostrarne l'appartenenza alla classe **NP** e
- ridurre polinomialmente ad esso un altro problema **NP**-completo.

Infatti, se sappiamo che un problema A è in **NP** e se un problema **NP**-completo B è tale che $B \leq A$, allora ogni altro problema in **NP** sarà riducibile polimialmente ad A. Proviamo ora questa affermazione. Sia C un qualunque problema in **NP**; poiché B è **NP**-completo, allora $C \leq B$, ossia, esiste una funzione $f_{CB} : I_C \to I_B$, calcolabile in tempo polinomiale, che trasforma istanze sì di C in istanze sì di B e viceversa. D'altra parte, $B \leq A$: allora esiste una funzione $f_{BA} : I_B \to I_A$, calcolabile in tempo polinomiale, che trasforma istanze sì di B in istanze sì di A e viceversa. Ma allora la funzione $f_{BA} \circ f_{CB} : I_C \to I_A$, composizione di f_{CB} con f_{BA}, è calcolabile in tempo polinomiale e trasforma istanze sì di C in istanze sì di A e viceversa. Questo prova che A è **NP**-completo.

Dedichiamo il resto di questa sezione a mostrare due esempi di dimostrazioni di **NP**-completezza.

Sia 3SAT il problema decisionale che ha la stessa definizione del problema SAT con il vincolo ulteriore che la funzione booleana nell'istanza, in forma congiuntiva normale, abbia clausole costituite da esattamente 3 letterali (ossia, sia in forma *3-congiuntiva normale*).

Teorema 2.2: *Il problema* 3SAT *è **NP**-completo.*

Dimostrazione: Poiché una funzione booleana in forma 3-congiuntiva normale è anche in forma congiuntiva normale, e in entrambi i casi ci chiediamo se esiste una assegnazione di verità che soddisfa la funzione, allora una istanza del problema 3SAT è anche istanza del problema SAT; questo prova che decidere 3SAT non può essere più difficile che decidere SAT e, poiché SAT$\in$ **NP** (Teorema 2.1), allora anche il problema 3SAT è in **NP**.

Per quanto riguarda la completezza, utilizziamo una riduzione dal problema SAT. Siano dati un insieme $X = \{x_1, x_2, \ldots, x_n\}$ ed una funzione $f(X) = c_1 \wedge c_2 \wedge \ldots \wedge c_m$, dove, per $j = 1, \ldots, m$, $c_j = l_{j_1} \vee l_{j_2} \vee \ldots \vee l_{j_{h_j}}$ con $l_{j_i} \in X$ oppure $\neg l_{j_i} \in X$, per $i = 1,2,\ldots,h_j$. Una riduzione da SAT a 3SAT deve trasformare l'istanza $\langle X, f \rangle$ di SAT in una istanza $\langle X', f' \rangle$ di 3SAT in modo tale che f è soddisfacibile se e soltanto se f' è soddisfacibile. Osservando la similitudine fra i due problemi, dunque, non si deve fare altro che trasformare ciascuna clausola di f in una o più clausole di f' (ossia, costituite da esattamente 3 letterali) in modo tale che esse siano soddisfatte da tutte e sole le assegnazioni di verità che soddisfano la clausola originaria di f. Descriviamo, allora, come opera la riduzione su una generica clausola $c_j = l_{j_1} \vee l_{j_2} \vee \ldots \vee l_{j_{h_j}}$ di f, inizializzando, momentaneamente X' uguale ad X:

a) se $h_j = 3$ (ossia, se c_j è costituita da esattamente 3 letterali) allora c_j viene inserita, invariata, in f';

b) se $h_j = 2$, ossia $c_j = l_{j_1} \vee l_{j_2}$, allora consideriamo una nuova variabile $y^{(j)} \notin X'$ e, dopo averla inserita in X', inseriamo in f' la coppia di clausole $c_{j_1} = l_{j_1} \vee l_{j_2} \vee y^{(j)}$ e $c_{j_2} = l_{j_1} \vee l_{j_2} \vee \neg y^{(j)}$: osserviamo che una assegnazione di verità per $\{l_{j_1}, l_{j_2}\}$ soddisfa c_j se e soltanto se la stessa assegnazione soddisfa anche sia c_{j_1} che c_{j_2};

c) se $h_j \geq 4$, ossia $c_j = l_{j_1} \vee l_{j_2} \vee \ldots \vee l_{j_{h_j}}$, allora consideriamo $h_j - 2$ nuove variabili $y_1^{(j)}, y_2^{(j)}, \ldots, y_{h_j-2}^{(j)} \notin X'$ e, dopo averle inserite in X', inseriamo in f' le $h_j - 2$ clausole $c_{j_1} = l_{j_1} \vee l_{j_2} \vee y_1^{(j)}$, $c_{j_2} = \neg y_1^{(j)} \vee l_{j_3} \vee y_2^{(j)}$, ..., $c_{j_{h_j-3}} = \neg y_{h_j-4}^{(j)} \vee l_{j_{h_j-2}} \vee y_{h_j-3}^{(j)}$, $c_{j_{h_j-2}} = y_{h_j-3}^{(j)} \vee l_{j_{h_j-1}} \vee l_{j_{h_j}}$: osserviamo che c_j una assegnazione di verità per $\{l_{j_1}, \ldots, l_{j_{h_j}}\}$ soddisfa c_j se e soltanto se la stessa assegnazione soddisfa anche tutte le clausole $c_{j_1}, \ldots, c_{j_{h_j}}$;

d) se $h_j = 1$, ossia $c_j = l_{j_1}$, allora consideriamo due nuove variabili $y_j, z_j \notin X'$ e, dopo averle inserite in X', inseriamo in f' le clausole $c_{j_1} = l_{j_1} \vee y_j \vee z_j$, $c_{j_2} = l_{j_1} \vee \neg y_j \vee z_j$, $c_{j_3} = l_{j_1} \vee y_j \vee \neg z_j$, $c_{j_4} = l_{j_1} \vee \neg y_j \vee \neg z_j$: osserviamo che c_j una assegnazione di verità per $\{l_{j_1}\}$ soddisfa c_j se e soltanto se la stessa assegnazione soddisfa anche sia c_{j_1} che c_{j_2} che c_{j_3} che c_{j_4}.

Dunque, costruita $\langle X', f' \rangle$ in accordo a quanto sopra descritto, una assegnazione di verità per le variabili in X soddisfa f se e soltanto se soddisfa anche f'. Poiché è possibile calcolare $\langle X', f' \rangle$ da $\langle X, f \rangle$ in tempo polinomiale in $|X|$ e $|f|$, segue che 3SAT è completo per la classe **NP**. □

Teorema 2.3: *Il problema* VERTEX COVER *è* ***NP****-completo.*

Dimostrazione: Abbiamo già visto nella precedente sezione che il problema è in **NP**.

Per quanto riguarda la completezza, utilizziamo una riduzione dal problema 3SAT.
Sia allora $X = \{x_1, x_2, x_n\}$ e $\varphi = c_1 \wedge c_2 \wedge \ldots \wedge c_m$ in cui $c_j = l_{j1} \vee l_{j2} \vee l_{j3}$ e $l_{ji} \in X \vee \neg l_{ji} \in X$, per $j = 1, \ldots, m$ e $i = 1, 2, 3$. La riduzione che associa a φ 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 ciascuna variabile in X e la sua negazione. 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 2.1.

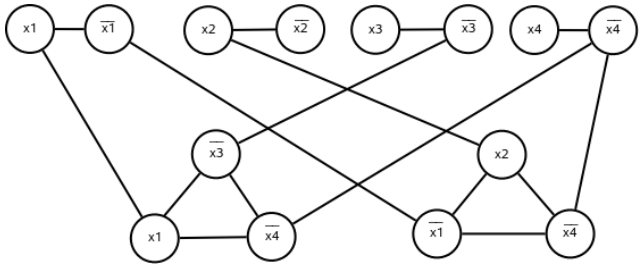

Figura 2.1: Il grafo G corrispondente alla formula $(x_1 \vee \neg x_3 \vee \neg x_4) \wedge (\neg x_1 \vee x_2 \vee \neg x_4)$.

A questo punto, possiamo affermare che φ è soddisfacibile se e solo se esiste un ricoprimento tramite vertici del grafo ottenuto di cardinalità pari a

$$k = n + 2m$$

dove, ricordiamo, m è il numero di clausole ed n il numero di variabili booleane.

Dimostriamo l'affermazione precedente mostrando, per prima cosa, come un assegnazione di verità a che soddisfa φ corrisponde all'esistenza di un ricoprimento del grafo di cardinalità k. Sia, allora, $\varphi(a(X)) = vero$; pertanto, dovrà essere $a(c_j) = a(l_{j1}) \vee a(l_{j2}) \vee a(l_{j3}) = vero$ per ogni clausola c_j, ossia, almeno un letterale l_{ji} all'interno della clausola risulta soddisfatto dall'assegnazione a.
Per ogni elemento $x_i \in X$, inseriamo in V' il vertice x_i se $a(x_i) = vero$, mentre inseriamo in V' il vertice $\bar{x}_i$ se $a(x_i) = falso$. In tal modo, abbiamo inserito in V' esattamente n vertici. Inoltre, per ogni $j = 1, \ldots, m$, scegliamo un letterale l_{ji} tale che $a(l_{ji}) = vero$ (tale letterale esista poiché a soddisfa φ) ed inseriamo in V' i nodi corrispondenti ai rimanenti due letterali di c_j. Ora, V' ha cardinalità $n + 2m = k$.
In questo modo:

- ogni arco del tipo $(x_i, \bar{x}_i)$ di G ha esattamente un estremo in V';
- poiché per ogni clausola c_j abbiamo inserito in V' due nodi corrispondenti ai suoi letterali, ciascuno dei tre archi corrispondenti ad essa è coperto da almeno un nodo in V';
- consideriamo ora un arco (l_{ji}, x_h) (un ragionamento analogo è valido per un arco $(l_{ji}, \bar{x_h})$): se $a(x_h) = vero$ allora $x_h \in V'$ e l'arco risulta coperto, mentre, se $a(x_h) = falso$ allora, per come abbiamo costruito V', $l_{ji} \in V'$ e l'arco risulta, ugualmente, coperto.

Questo dimostra che V' è un vertex cover per G.

Supponiamo, ora, che il grafo G ammetta un vertex cover $V' \subseteq V$ di cardinalità pari a $n+2m$. Osserviamo che, per ogni $x_i \in X$, V' deve contenere il nodo x_i oppure il nodo $\bar{x}_i$ (altrimenti l'arco $(x_i, \bar{x}_i)$ non sarebbe coperto): poniamo, allora $V' = Y \cup W$, dove l'insieme Y vontiene gli elementi di V' del tipo corrispondenti alle variabili in X o alle loro negazioni. Allora, $|Y| = n$, da cui segue che deve essere $|W| = 2m$.
Osserviamo ora che, per $j = 1, \ldots, m$, al fine di coprire i tre archi costituenti il grafo completo corrispondente alla clausola c_j, almeno due nodi corrispondenti a suoi letterali devono essere contenuti in W; d'altra parte, poiché $|W| = 2m$, per ogni $j = 1, \ldots, m$, esattamente due letterali corrispondenti alla clausola c_j sono contenuti in W.
Vediamo, infine, come è possibile ottenere da V' un assegnazione a che soddisfa φ. Se $x_i \in Y$ allora poniamo $a(x_i) =$ *vero*, altimenti, se $\bar{x}_i \in Y$, poniamo $a(x_i) = falso$. Questa assegnazione soddisfa la formula perché tutti e tre gli archi che collegano le clausole ai letterali sono coperti (essendo V' un ricoprimento per G), ma solo due vertici corrispondenti a letterali in ciascuna clausola possono appartenere a V' (altrimenti sarebbe$|W| > 2m$). 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 a soddisfa c_j, per ogni $j = 1, \ldots, n$. □

2.3 Problemi di ottimizzazione e algoritmi di approssimazione

In un problema decisionale l'obiettivo è *decidere se esiste* una soluzione possibile che sia anche soluzione effettiva. Ad esempio, nel problema VERTEX COVER, dati un grafo $G = (V, E)$ ed un intero positivo k, si cerca un sottoinsieme V' di V (la soluzione possibile) *tale che* la sua cardinalità sia al più k ed ogni nodo in E abbia almeno un estremo in esso.
In un *problema di ottimizzazione*, invece, ad ogni soluzione effettiva è associata una misura e l'obiettivo è *calcolare* una soluzione effettiva la cui misura sia *ottima* (ossia, dipendentemente se si tratta di un problema di minimizzazione o di massimizzazione, di misura minima o massima).

Esempio. Il prolema MINIMUM VERTEX COVER (in breve, minVC) è il problema di minimizzazione in cui è dato un grafo $G = (V, E)$ e viene richiesto di calcolare un sottoinsieme V' di V di cardinalità minima tale che ogni arco in E abbia almeno un estremo in V'. □

Osserviamo come al problema di ottimizzazione minVC appena definito sia possibile far corrispondere naturalmente il problema decisionale VC. Questo accade per tutti i problemi di ottimizzazione: chiamiamo *problema decisionale soggiacente* al problema di ottimizzazione P_O quel problema decisionale il cui insieme delle istanze è il prodotto cartesiano $I_{P_O} \times \mathbb{R}$ ed in cui ci si interroga circa l'esistenza di una soluzione effettiva di misura limitata (inferiormente se il problema è di minimizzazione, superiormente se il problema è di massimizzazione) dal valore numerico.

In presenza di problemi **NP**-completi ci aspettiamo che non esista alcun algoritmo operante in tempo polinomiale che trovi una soluzione ottima per il problema di ottimizzazione corrispondente (altrimenti, tale algoritmo permetterebbe di decidere in tempo polinomiale il problema decisionale soggiacente).
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 2.3 (Fattore di approssimazione): *Sia*

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

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

$$1 \leq \max\left\{\frac{c}{c^*}, \frac{c^*}{c}\right\} \leq \rho(n) \tag{2.1}$$

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

Input: $G = (V,E)$.		
1	$C := \emptyset$	il ricoprimento, inizialmente vuoto
2	$F := \emptyset$	utilizzato ai soli fini della prova del Lemma **??**
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	$E := E - \{e\}$	elimina da E l'arco considerato
10	**end**	
11	**Output**: C	

Tabella 2.1: Algoritmo 2-approssimante per MVC.

Osserviamo meglio la disuguaglianza in 2.1. 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.

Presentiamo, a titolo di esempio, un algoritmo 2-approssimante per il problema MVC visto in precedenza. Consideriamo l'algoritmo in Tabella 2.1 che ha in input un grafo $G = (V,E)$.

Vediamo in dettaglio come opera questo algoritmo. Innanzitutto, il ricoprimento tramite nodi del grafo fornito in input G è memorizzato nell'insieme C, 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 considera 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 viene eseguita alle linee 5-7: in particolare, il controllo alla linea 5 serve per scartare archi non disgiunti da quelli già inseriti in C.

Teorema 2.4: *L'algoritmo in Tabella 2.1 calcola un ricoprimento tramite nodi del grafo input $G = (V,E)$ che ha cardinalità al più due volte quella ottima.*

Dimostrazione: Ricordiamo che C è il ricoprimento tramite nodi di $G = (V,E)$ calcolato dall'algoritmo ed indichiamo con C^* un ricoprimento tramite nodi di G ottimo. Vogliamo dimostrare che $|C| \leq 2|C^*|$. A questo scopo, osserviamo che, ogni volta che l'algoritmo aggiunge un arco ad F, esso aggiunge anche i suoi due estremi a C. Perciò, deve essere

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

Osserviamo, inoltre, che l'insieme degli archi selezionati è un sottoinsieme degli archi del grafo,ossia, $F \subseteq E$. Infine, la soluzione ottima C^*, per definizione di ricoprimento tramite nodi, deve coprire tutti gli archi di F e, dato che, per costruzione (linee 5-7 dell'algoritmo), nessuna coppia di archi in F ha estremi in comune, allora ogni nodo $u \in C^*$ copre al più un arco di F. Pertanto, 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 **??** per ogni arco l'algoritmo selezionerà entrambi i suoi vertici mentre la soluzione ottima consiste nel prenderne soltanto uno.