Laurea Magistrale in Informatica A.A. 2020/2021
Università degli Studi di Milano-Bicocca
Appunti Teoria della Computazione

Marta Pelusi
@Marta629

Indice

1 Macchine di Turing

Sia L_π un linguaggio di una **Macchina di Turing** che è un linguaggio di tutte le istanze per cui la TM risponde YES, cioè è l'insieme degli input del problema per cui la risposta è sempre 1.

1.1 Problemi P e problemi NP

Definizione 1. *i **linguaggi P** sono definiti come una classe di linguaggi di decisione che sono accettati da un algoritmo A in tempo polinomiale da una Macchina di Turing TM.*

Definizione 2. *un L_π è accettato da una **Macchina di Turing deterministica** in tempo $T(n)$ polinomiale se $\exists\, T : N \to N$ calcolabile da TM e $\forall x \in L_\pi$ con $|x| = n$ la macchina TM risponde 1/YES in $T(n)$ mosse di calcolo/configurazioni.*

I **linguaggi P** possono essere, quindi, anche definiti come quella classe di linguaggi L_π di decisione che, considerato un algoritmo A che accetta L_π in tempo $T(n)$ polinomiale, questo algoritmo su input x, con $|x| = n$, termina dopo $T(n)$ passi di calcolo.

Definizione 3. *i **linguaggi NP** sono definiti come una classe di linguaggi di decisione accettati da un algoritmo A in tempo $T(n) = cn^p$ polinomiale da una Macchina di Turing non deterministica NDTM.*

Definizione 4. *sia $T : N \to N$ funzione calcolabile da una Macchina di Turing TM e L_π un linguaggio di decisione, allora una **Macchina di Turing non deterministica** NDTM accetta L_π in tempo $T(n)$ polinomiale se $\forall x \in L_\pi$ con $|x| = n$, NDTM accetta x in $T(n)$ mosse di calcolo/configurazioni.*

I **linguaggi NP** possono essere, quindi, anche definiti come quella classe di linguaggi L_π di decisione, tale che presa $T : N \to N$ una funzione calcolabile e y una stringa di lunghezza polinomiale nell'input x, allora un algoritmo A con certificato y accetta L_π in tempo $T(n)$ polinomiale se $\forall x \in L_\pi$ con $|x| = n$, A termina su input (x, y) dopo $T(|x|)$ passi di calcolo producendo 1/YES in output. Questo significa che i **linguaggi NP** sono la classe di linguaggi di decisione accettati in tempo polinomiale da un algoritmo A con certificato.
Un **certificato** è una dimostrazione che prova che x può essere accettato. Spesso il certificato è la soluzione ammissibile al mio problema che mi consente di rispondere 1/YES alla domanda posta dal problema.

Dalla definizione di linguaggi P e linguaggi NP, si può concludere che c'è la seguente relazione tra problemi in P e problemi in NP:

$$P \subseteq NP$$

1.2 Problemi NP-difficili e problemi NP-completi

Definizione 5. *Un problema π si dice **problema NP-difficile** o NP-hard se:*

- $\pi \notin NP$
- $\forall \pi' \in NP,\ \pi' \leqslant_p \pi$, *ovvero se ogni problema* $\pi' \in NP$ *può essere ridotto polinomialmente ad un problema* $\pi \in NP-$difficile

Definizione 6. *Un problema* π *si dice* ***NP-completo*** *se è sia in NP che NP-difficile, ovvero se:*

- $\pi \in NP$
- $\forall \pi' \in NP,\ \pi' \leqslant_p \pi$

Tutti i problemi NP-completi sono tra loro **NP-equivalenti**.

2 Riduzione

Procedimento

- bisogna trasformare l'input w di A in un input $f(w)$ per B in tempo polinomiale
- la risposta di B con input $f(w)$ è la stessa che dà A su input w

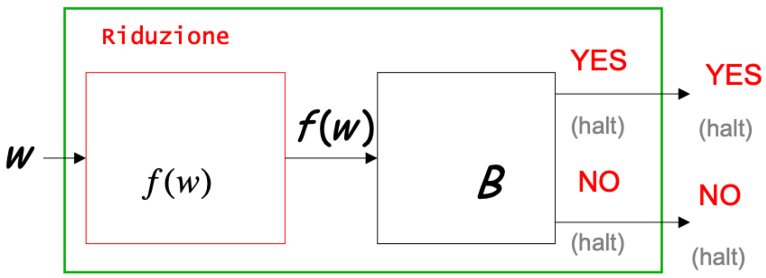

Figura 1: Riduzione polinomiale se f è una funzione polinomiale.

Definizione 7. ***A si riduce polinomialmente a B*** *se esiste una funzione* f *tale che* $w \in L_A \Leftrightarrow f(w) \in L_B$ *con* f *calcolabile in tempo polinomiale.*

Osservazione 1. *la riduzione non è una relazione simmetrica, cioè se* $A \leqslant_p B$ *non è detto che sia vero anche* $B \leqslant_p A$.

Teorema 1. *se* $\pi \in$ *NP-completo e* $\pi \in P$ *allora P=NP.*

Dim. *per definizione di NP-difficile sappiamo che* $A \in$ *NP-difficile se* $A \in NP$ *e se* $A \leqslant_p \pi$, *cioè* π *è una procedura che risolve A dopo aver trasformato l'input* x *di A in un input* $f(x)$ *per* π. $f(x)$ *è calcolabile in tempo polinomiale, quindi la trasformazione dell'input è polinomiale. Questo fatto, combinato con l'assunzione che* $\pi \in P$, *implica che ogni problema* $A \in P$, *cioè P=NP.*
Il costo sarà la composizione di costi polinomiali: $O(|x| \cdot c_1) + O(|f(x)| \cdot c_2)$.

Teorema 2. *la riduzione polinomiale è transitiva.*

Dim. *se* $A \leqslant_p B \ \wedge \ B \leqslant_p C \ \Rightarrow A \leqslant_p C$

- $A \leqslant_p B \Rightarrow \exists \ f : \ x \in L_A \Leftrightarrow f(x) \in L_B$
- $B \leqslant_p C \Rightarrow \exists \ g : \ x \in L_B \Leftrightarrow g(x) \in L_C$
- $A \leqslant_p C \Rightarrow \exists \ f' : \ x \in L_A \Leftrightarrow f'(x) \in L_C$

Assumo che $f'(x) = g(f(x))$. *L'idea è di prendere* $x \in$ *input per* A, $f(x) \in$ *input per* B, $g(f(x)) \in$ *input per* C, *quindi* $x \in L_A \Leftrightarrow f(x) \in L_B \Leftrightarrow g(f(x)) \in L_C$. *Poichè* f *è calcolabile in tempo polinomiale rispetto a* $|x|$ *e anche* g *è calcolabile in tempo polinomiale rispetto a* $|f(x)|$, *deduco che* $f'(x) = g(f(x))$ *è calcolabile in tempo polinomiale rispetto a* $|x|$ *in quanto* $x \longrightarrow f'(x)$.

3 Vertex cover - VC

Definizione 8. *Un insieme* $V' \subseteq V$ *è **copertura** di vertici del grafo* $G = (V, E)$ *se* $\forall e \in E$ *almeno un estremo di* $e = (u, v)$ *arco sta in* V', *quindi* $u \in V' \ \vee \ v \in V'$.

Definizione 9 (vertex-cover versione di ottimo)**.** *il **problema del VC** ha come input un grafo* $G = (V, E)$ *e come output si vuole trovare un sottoinsieme* $V' \subseteq V$ *tale che* V' *è una copertura minima per* G.

Definizione 10 (vertex-cover versione di decisione)**.** *il **problema del VC** ha come input un grafo* $G = (V, E)$ k *intero, e come output 0/1, ovvero si vuole decidere se* $\exists \ V' \subseteq V$ *tale che* V' *è una copertura di* G *di dimensione* k.

Il problema vertex-cover è un problema di minimo ed è **NP-completo**.

Algoritmo

```
pseudocode: "vertex-cover"
for each e in E do
  e=(u,v) almeno u o v in y
```

L'algoritmo ha tempo polinomiale $O(|E|)$.

4 Problema del commesso viaggiatore - TSP

Definizione 11 (versione di ottimo)**.** *il **problema TSP (travelling salesman problem)** ha come input un grafo* $G = (V, E)$ *completo e pesato e come output si vuole trovare il cammino di peso minimo che parte da un vertice* $u \in E$ *e torna nel vertice* u *visitando una ed una sola volta tutti i vertici del grafo.*

Definizione 12 (versione di decisione)**.** *il **problema TSP (travelling salesman problem)** ha come input un grafo* $G = (V, E)$ *completo e pesato e un parametro intero* k, *e come output 0/1, ovvero si vuole decidere se esiste il cammino che parte da un vertice* $u \in E$ *e torna nel vertice* u *visitando una ed una sola volta tutti i vertici del grafo di costo al massimo* k.

TSP è un problema di minimo ed è **NP-difficile**, mentre la sua versione di decisione TSP_d è sempre un problema di minimo ed è **NP-completo**.

$\text{TSP}_d \in NP$ perchè esiste un algoritmo A polinomiale con input (x, y) che verifica che y su x è una soluzione che consente ad A di rispondere 1/YES in tempo polinomiale se $x \in L_\pi = L_{TSP_d}$. Il tempo d'esecuzione è $|y| = O(|x|)$, con input $x = (G, k)$.

5 Indipendent-set - IND-SET

Definizione 13. *l'**insieme indipendente** di un grafo $G = (V, E)$ non orientato è un sottoinsieme $I \subseteq V$ di vertici tale che $\forall u, v \in I,\ (u, v) \notin E$.*

Definizione 14 (versione di ottimo)**.** *il **problema indipendent-set** ha come input un grafo $G = (V, E)$ non orientato e come output si vuole trovare l'insieme $I \subseteq V$ tale che I è un insieme indipendente con cardinalità massima.*

Definizione 15 (versione di decisione)**.** *il **problema indipendent-set** ha come input un grafo $G = (V, E)$ e un parametro intero k tale che $k \leqslant |V|$ e come output 0/1, ovvero si vuole decidere se G ha un insieme indipendente di dimensione di dimensione almeno k.*

Osservazione 2. *il complemento di una copertura è sempre un insieme indipendente, quindi il complemento di una copertura minima è sempre il massimo insieme indipendente. Questo significa che, mentre il vertex-cover è un problema di minimo, indipendent-set è un problema di massimo.*

Fatto 1. *se V è l'insieme dei vertici di un grafo G e V' è una copertura minima di G allora l'insieme indipendente di cardinalità massima sarà definito come $I = V \setminus V'$.*

Dim. *per definizione V' è tale che $\forall (u, v) \in E$, cioè per ogni arco c'è almeno un estremo di E in V'. Per definizione tutti gli archi stanno in V'. Se I non fosse un insieme indipendente, quindi, ci sarebbe un arco che non V' non copre perchè sarebbe in I, ma qui si contraddirebbe il fatto che V' è una copertura.*

Teorema 3. *il problema $\text{IND-SET}_d \in NP$.*

Dim. *esiste un algoritmo A che ha costo polinomiale e che dato in input $x = (G, k)$ e un certificato y (=vertici di un insieme indipendente di dimensione k) verifica y su x è insieme indipendente.*

Il **costo computazionale** dell'algoritmo dell'indipendent-set si ottiene sapendo che $\forall u, v \in y$ verifica che $(u, v) \notin E \Rightarrow$ il costo di verifica è quadratico in $|y|$ perchè si avranno $|y| \cdot |y| = |y|^2$ coppie da esaminare. La carnalità $|y|^2 = O(|E| + |V|) = O(|x|)$ e si sa che $|y| \leqslant |V|$, quindi nel caso peggiore si avrà $|V|^2 = |E|$, quindi $|y| = O(|V|)$.

Fatto 2. *IND-SET$\leqslant_p$VC e VC$\leqslant_p$IND-SET*

Dim. *la dimostrazione si basa sul fatto che l'insieme indipendente è sempre il complemento di un vertex-cover, quindi dato un grafo G in input e avendo il suo insieme indipendente (o avendo il suo vertex-cover) è sempre possibile, facendone il complemento, trovare quindi in tempo polinomiale anche il suo vertex-cover (o trovare anche il suo insieme indipendente). Quindi dato un grafo $G = (V, E)$ in input prendo un suo insieme indipendente I. Considerando $V \backslash I = V'$ so che V' è una copertura di vertici per G.*
Vale anche il viceversa.

6 Problema di soddisfacibilità - SAT

Definizione 16. *una **formula normale congiunta** CNF ha diversi livelli di composizione:*

- *CNF è un $\wedge$ di clausole (congiunzione)*
- *una clausola è formata da $\vee$ di letterali*
- *un letterale è una variabile booleana x_i oppure $\neg x_i$*

Affinchè sia vera una clausola è sufficiente che sia vero un letterale.

Definizione 17. *il **problema di soddisfacibilità** ha come input una formula ϕ formula booleana in forma normale congiunta (CNF) e come output 0/1, ovvero si vuole decidere se ϕ è soddisfacibile o meno.*

SAT è un problema **NP-completo**. Il problema k-SAT è stato il primo problema della storia ad essere definito NP-completo. Introduciamo il problema 3-SAT: ciò che differenzia da SAT è che, per ogni clausola, presenta solamente 3 letterali. Sapendo ciò è possibile dimostrare il seguente fatto trasformando una clausola ϕ in un grafo G_ϕ. La conclusione sarà che anche IND-SET è un problema NP-completo.

Fatto 3. *3-SAT$\leqslant_p$ IND-SET*

Dim. *è necessario procedere per step:*

1. *bisogna far vedere che $\exists\ f$ che trasforma l'input ϕ di 3-SAT nell'input (G, k) di IND-SET e bisogna far vedere che f è polinomiale*
2. *bisogna far vedere che $w \in L_A \Longleftrightarrow f(w) \in L_B$*

ovvero bisogna far vedere che:
($\Rightarrow$) se ϕ è vera allora esiste un insieme indipendente di dimensione k per (G, k)
($\Leftarrow$) se esiste un insieme indipendente di dimensione k per (G, k) allora ϕ è vera

Si vuole costruire G_ϕ partendo da ϕ:

- *G contiene 3 vertici per ogni clausola, precisamente uno per ogni letterale della clausola*

- *si collegano i 3 letterali di una singola clausola come fossero un triangolo (gadget)*
- *si collegano tra loro ogni gadget collegando ogni letterale al suo negato*

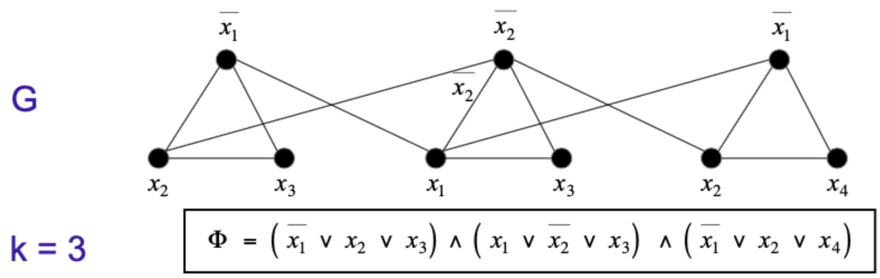

Figura 2: Costruzione dei gadget a partire da una formula normale congiunta.

($\Rightarrow$) sia dato un assegnamento di verità e seleziono un vertice per ogni triangolo, cioè un solo letterale vero. Si ottiene un insieme indipendente di dimensione k.
Se ϕ è vera allora per ogni clausola c_i esiste il letterale l_{ij} che la rende vera. Tale letterale è un vertice del triangolo gadget G_{c_i} e poichè ogni clausola è vera allora si ha la scelta di k vertici, se k sono le clausole.

($\Leftarrow$) se esiste un insieme indipendente di dimensione k allora trovo un assegnamento di verità alle variabili $< x_1, x_2, ..., x_m >$ che rende vera ϕ.
Se esiste l'insieme indipendente I di dimensione k significa che per ogni gadget G_{c_i} esiste un vertice in I, quindi esiste un letterale per ogni clausola che non è collegato ad un letterale di un altro gadget. Ovvero data la clausola c_s esiste il letterale l_s di c_s che non è collegato al letterale l_t della clausola c_t.
Ad ogni letterale l_i di ϕ corrisponde una variabile x_i del gadget. Col valore dato alla variabile, quindi, si costruisce l'assegnamento di verità, cioè:

- *se* $l_i = x_i \rightarrow x_i = 1$
- *se* $l_i = \neg x_i \rightarrow x_i = 0$

In questo modo trovo un assegnamento di verità per le variabili che è di verità anche per ϕ, dimostrando che ϕ è vera.

7 Set-cover - SC

Definizione 18. *dato un universo U di n elementi, una collezione $S = \{S_1, S_2, ..., S_m\}$ di sottoinsiemi di U e una funzione di costo $c : S \rightarrow Q^+$, un **set-cover** è una collezione C di sottoinsiemi di S di minimo costo che copra tutti gli elementi di U. In altre parole il set-cover è una sottocollezione che copra tutti gli elementi di U e che abbia minima dimensione.*

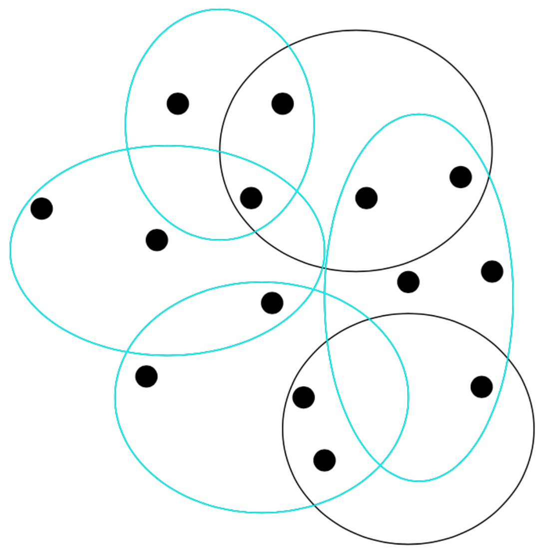

Figura 3: Esempio di set-cover.

Definizione 19 (versione di ottimo)**.** *il **problema del set-cover** ha come input un insieme* $U = \{e_1, e_2, ..., e_n\}$ *universo di città,* $S_1 \subseteq U$*,* $S_2 \subseteq U, ...,$ $S_l \subseteq U$*,* $R_1 \rightarrow \{e_{11}, e_{12}, ..., e_{1m}\}$*,* $R_2 \rightarrow \{e_{21}, e_{22}, ..., e_{2r}\}$ *... e come output si vuole trovare la collezione di minima cardinali di sottoinsiemi la cui unioni è* U*, ovvero si vuole trovare*

$$S_{i1}, S_{i2}, ..., S_{ik} : \bigcup_{1 \leqslant j \leqslant k} S_{ij} = U$$

tale che k *sia minimo.*

Definizione 20 (versione di decisione)**.** *il **problema del set-cover** ha come input un insieme* $U = \{e_1, e_2, ..., e_n\}$ *universo di città, e un parametro intero* j *e come output 0/1, ovvero si vuole trovare la collezione di minima cardinali di sottoinsiemi la cui unioni è* U*, ovvero si vuole stabilire se esiste una collezione* C *di insiemi di* S *la cui unione sia* U *e tale che* C *sia di cardinalità* $\leqslant j$ *(numero minimo di sottoinsiemi che coprono tutto l'universo* U*).*

Esempio 1. *Questa è la trasformazione dell'input di VC in un input di SC.*
$U = \{(1,2), (2,3), (3,4), (2,5), (4,5)\}$, $\mathcal{C} = \{S_{v_1}, ..., S_{v_i}\}$ *e:*

- $S_{v1} = \{(1,2)\}$
- $S_{v2} = \{(1,3), (2,3), (2,5)\}$
- $S_{v3} = \{(2,3), (3,4)\}$
- $S_{v4} = \{(3,4), (4,5)\}$
- $S_{v5} = \{(2,5), (4,5)\}$

Fatto 4. *set-cover è NP-completo.*

Dim. *dimostro che $SC \in NP$: data una sottocollezione C si verifica in tempo polinomiale che $|C| \leqslant k$ e che l'unione degli insiemi di C inclde tutti gli elementi di U. Per farlo utilizzo un algoritmo che lavora in tempo polinomiale e che funziona prendendo U e sottraendogli i suoi sottoinsiemi. Se non avanzano elementi allora l'algoritmo funziona.*
Questo algoritmo è polinomiale con certificato y (=collezione di insiemi che dimostra che si può coprire tutto U).

Dimostro che $SC \in$ NP-difficile: data un'istanza/input C di VC (cioè (G,k), j intero positivo) costruiamo un'istanza/input C' di SC in tempo polinomiale tale che C è soddisfacibile (=ammette risposta 1/YES) se e solo se C' è soddsifacibile.

- *$U = \{e_1, e_2, ..., e_m\}$ l'universo sono gli archi perchè l'obiettivo di VC è coprire tutti gli archi, con $|E| = m$, $E = \{e_1, ..., e_m\}$ archi di E*
- *$S_1, S_2, ..., S_?$ codificano i vertici, ovvero tutti gli archi che un vertice copre*

Per dimostrare che SC è NP-difficile devo dimostrare che $VC \leqslant_p SC$.
Mostriamo che vertex-cover ha risposta 1/YES per j se e solo se set-cover ha risposta 1/YES per $k = j$.

($\Rightarrow$) Assumiamo che VC ha una copertura $C = \{v_{i1}, ..., v_{ik}\}$ di k vertici. Data una collezione di sottoinsiemi $\mathcal{C} = \{S_{v_{i1}}, ..., S_{v_{ik}}\}$ e riesco a dimostrare che

$$\bigcup_{1 \leqslant j \leqslant k} S_{v_{ik}} = U$$

perchè per ogni arco e di U sappiamo che e ha un estremo in C, e quindi e appartiene ad un insieme della collezione $\mathcal{C}$.

($\Leftarrow$) Data una collezione di sottoinsiemi $\mathcal{C} = \{S_{v_{i1}}, ..., S_{v_{ik}}\}$ che copre U costruiamo $C = \{v_{i1}, ..., v_{ik}\}$ e dimostriamo che C copre il grafo G. Per ogni arco $e \in U$ so che e appartiene ad un insieme $\mathcal{C}$, quindi se $e \in S_{v_{il}}$ allora significa che v_{il} è un estremo dell'arco e. Quindi $\forall e \in E$, e ha un estremo in C. Questo implica che C è copertura di G.

8 Clique problem - CL

Definizione 21. *la **clique** di un grafo $G = (V, E)$ non orientato è un sottoinsieme $V' \subseteq V$ di vertici di G tale che per ogni coppia di vertici $u, v \in V'$, $(u, v) \in E$.*

Definizione 22 (versione di ottimo)**.** *il **problema della cricca** ha come input un grafo $G = (V, E)$ e come output si vuole trovare $V' \subseteq V$ che è una clique di dimensione massima, cioè $|V'|$ è massimo.*

Definizione 23 (versione di decisione)**.** *il **problema della cricca** ha come input $G = (V, E)$ e un parametro intero k e come output 0/1, ovvero si vuole stabilire se esiste un sottoinsieme $V' \subseteq V$ che è una clique di dimensione $\geqslant k$, cioè $|V'| = k$.*

Fatto 5. *il clique problem è NP-completo.*

Dim. *dimostro che $CL \in NP$: prendo $V' \subseteq V$ di vertici come certificato y per l'input G. Per verificare che V' è una clique controllo che $\forall u, v \in V'$, $(u, v) \in E$ in tempo polinomiale $\rightarrow$ $O(|V'| \cdot |V'|) = O(|V'|^2)$ è quadratico nella dimensione dell'input G.*

Dimostro che $CL \in$ NP-difficile: bisogna trovare un problema $A \in$ NP-completo (mi interessa che $A \in$ NP-difficile) tale che $A \leqslant_p$ CL, cioè CL è una sottoprocedura per A che risolve A.

La clique è il complementare dell'insieme indipendente, quindi se per trasformare una clausola in un grafo nel caso di 3-SAT$\leqslant_p$IND-SET collegavo tra loro i vertici associati ad una stessa clausola, nel caso di 3-SAT$\leqslant_p$CL non li collego più. Collego tutto il collegabile tranne vertici corrispondenti a letterali della stessa clausola e tranne vertici inconsistenti (es x_1 e $\neg x_1$).

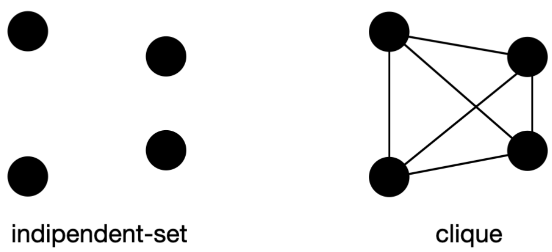

Figura 4: Clique è il complemento di indipendent-set.

Siccome 3-SAT$\leqslant_p$IND-SET e IND-SET$\leqslant_p$CL, posso considerare A=3-SAT e ottenere A$\leqslant_p$CL Dato SAT$\leqslant_p$CL vale anche il viceversa, ovvero CL$\leqslant_p$SAT.

Esempio: *avendo* $\phi = c_1 \wedge c_2 \wedge c_3$
$c_1 = x_1 \vee \neg x_2 \vee \neg x_3$ $\quad l_1^1,\ l_2^1,\ l_3^1$
$c_2 = \neg x_1 \vee x_2 \vee x_3$ $\quad l_1^2,\ l_2^2,\ l_3^2$
$c_3 = x_1 \vee x_2 \vee x_3$ $\quad l_1^3,\ l_2^3,\ l_3^3$

Si vuole costruire G_ϕ a partire da ϕ:

- *per ogni clausola $c_r = l_1^r \vee l_2^r \vee l_3^r$ in ϕ definiamo un gadget G_{c_r} che consiste di 3 vertici $v_1^r,\ v_2^r,\ v_3^r$ in G_ϕ tale che $V = \bigcup_{1 \leqslant i \leqslant k}(v_1^i,\ v_2^i,\ v_3^i)$*
- *per ogni coppia di vertici $v_i^s,\ v_j^r$ esiste l'arco in E, cioè $(v_i^s,\ v_j^r) \in E$ sse $l_i^s\ s_j^r$ sono consistenti, cioè $l_i^s \neq \neg l_j^r$, cioè l'uno non esclude l'altro (non sono uno il negato dell'altro)*

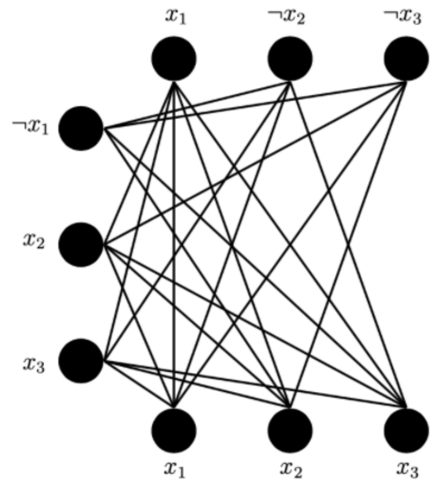

Figura 5: Costruzione della clique del grafo a partire da una formula normale congiunta.

($\Rightarrow$) Se ϕ ha un assegnamento che la rende vera, allora esiste una clique per il grafo G_ϕ di dimensione k.
Se ϕ è vera allora bisogna costruire una clique per il grafo G che abbia k vertici. Per ogni clausola esiste almeno un letterale che è vero e assumiamo che sia $l_i^r = 1$ (associato al vertice v_i^r del grafo). Associo ad ogni clausola un letterale vero, il quale corrisponde ad un vertice:

$$c_1 \longrightarrow l_i^1 \longrightarrow v_i^1 \ \ldots \ \ldots \ c_k \longrightarrow l_z^k \longrightarrow v_z^k$$

Costruisco l'insieme dei vertici $V' \subseteq V$ dove $V' = \{v_i^1,\ v_j^2, ..., v_z^k\}$.
Ora dobbiamo dimostrare che V' è una clique di dimensione k (un vertice per ogni clausola), cioè $\forall v_i^r, v_j^s \in V'$, $(v_i^r, v_j^s) \in E$ con $r \neq s$.
Per ogni coppia di letterali l^s, l^r questi sono consistenti, e per costruzione del grafo G_ϕ i letterali consistenti sono collegati da un arco, quindi $(u, v) \in E,\ \forall u, v \in V'$.

($\Leftarrow$) Se esiste una clique $V' = \{v_{i1},\ v_{i2}, ..., v_{ik}\} \subseteq V$ di dimensione k (pari al numero di clausole) per il grafo G allora ϕ è vera, cioè le k clausole sono vere.
Sappiamo per costruzione che V' ha un solo vertice per ogni clausola, ad es v_i^r per la clausola c_r a cui è associato al letterale $l_i^r = 1$ (so che il letterale negato di l_i^r non verrà usato per rendere vera una clausola). Riesco quindi a costruire un assegnamento di verità consistente per ϕ. In altre parole per ogni vertice di $v_{it} \in V'$, se $v_{it} \in$ gadget della clausola c_t, v_{it} è associato ad un letterale che rendo vero.

- *se $l_{it} \longrightarrow x_j \Rightarrow x_j = 1$*
- *se $l_{it} \longrightarrow \neg x_j \Rightarrow x_j = 0$*

Rendo vera ogni clausola rendendo vero il letterale associato al vertice della clausola, quindi so di ottenere un assegnamento di verità ϕ in quanto i letterali sono consistenti.

Fatto 6. *$CL \leqslant_p VC$*

9 Problemi di ottimizzazione

Dato un problema π e un'istanza x, l'ottimo su x di π è indicato come $opt(x)$. Il costo di una soluzione ammissibile su x è calcolato da un algoritmo A per il problema π ed è indicato come $A(x)$.

Definizione 24. *un **algoritmo ϵ-approssimato** per un problema π di ottimizzazione è un algoritmo A polinomiale che restituisce una soluzione ammissibile che "dista" un fattore costante ϵ da quella ottima.*

- *se π è un problema di minimo allora $A(x) \leqslant \epsilon \cdot opt(x),\ \epsilon > 1$*
- *se π è un problema di massimo allora $A(x) \geqslant \epsilon \cdot opt(x),\ \epsilon < 1$*

Esempio 2. *costo della soluzione per vertex-cover. Dato un grafo $G = (V, E)$ come input, si vuole ottenere come output una copertura di vertici per G (che non è necessariamente la minima).*
Il costo di una soluzione è $|V'|$ se V' è la copertura di vertici per G.
Il costo ottimo è $|V'|$ minima.

Il **matching perfetto** di un grafo per vertex-cover sono tutti gli archi che non condividono tra loro estremi.

9.1 Idea della 2-approssimazione considerando vertex-cover

L'idea è di cercare un limite inferiore all'ottimo per il problema $opt(x) \geqslant k$ sapendo che:

- $opt(x)$ è la dimensione dell'ottimo su input $x : G = (V, E)$
- k è il limite inferiore dato da un matching

Si sviluppa, poi, un algoritmo polinomiale che produce una soluzione ammissibile per il problema tale che il costo di tale soluzione sia $\epsilon > 1$ limite inferiore sapendo che:

- $A(x)$ è la dimensione di un matching del grafo (vertici)
- $opt(x) \geqslant k$
- $A(x) \leqslant \epsilon \cdot k,\ \epsilon > 1$

Se si ha una ϵ-approssimazione vale:

$$\frac{A(x)}{opt(x)} \leqslant \frac{\epsilon \cdot k}{k} \leqslant \epsilon$$

Considerando vertex-cover si costruisce un matching perfetto del grafo in input utilizzando tutti i vertici. Il limite inferiore per $opt(x)$ è il numero di archi del matching:

$$opt(x) \geqslant k = \text{numero archi del matching}$$

L'algoritmo A costruisce incrementalmente un matching per G come segue:

- prende un arco e_i e rimuove tutti gli archi incidenti agli estremi di e_i e per fare inserisce nella soluzione ammissibile i due estremi di e_i
- $A(x) = 2$ volte il numero di archi nel matching
- $opt(x) \geqslant$ numero di archi nel matching

9.2 Tasso di approssimazione

$$\max\left\{\frac{A(x)}{opt(x)}, \frac{opt(x)}{A(x)}\right\} \leqslant r$$

Il massimo sta nel primo fattore per problemi di minimo, con $\epsilon > 1$. Il massimo sta nel secondo fattore per problemi di massimo, con $0 < \epsilon < 1$.
Non tutti i problemi NP-completi ammettono una r-rappresentazione per r costante:

$$\max\left\{\frac{A(x)}{opt(x)}, \frac{opt(x)}{A(x)}\right\} \leqslant \rho(x)$$

- x input del problema π, con $|x| = n$
- $\rho(x)$ è una funzione che dipende dalla dimensione dell'input

Algoritmo approx per vertex-cover

```
pseudocode: "approx code for vertx-cover"
C:={}
E:= insieme degli archi di G
while E != {} do
  let(u,v) in E %prendo (u,v) in E
  C:= {C, {u,v}} %C unito {u,v}
  forall (u,z) in E and (v,w) in E
    rimuovi da E (u,z) and (v,w)
```

Questo algoritmo ha costo polinomiale rispetto alla dimensione del grafo G in input: $O(|V|, |E|)$.

10 Complessità parametrica

L'idea è quella di prendere un problema NP-completo e, dato che si sa che alcune istanze sono "risolvibili" in tempo polinomiale, fissare dei parametri che caratterizzano queste istanze.
Prendendo come esempio vertex-cover ci chiediamo se esiste un vertex-cover di dimensione k (es $k = 5$) prendendo come input un grafo con *tantisssssssssssssimi* archi e nodi (es 10^8 nodi e 10^{20} archi). Per definizione di vertex-cover si dovrebbero avere tutti gli archi incidenti nei k vertici.

Definizione 25 (algoritmo parametrico)**.** *un problema π è **trattabile fissato un parametro** k se e solo se $\exists$ A algoritmo, $\exists$ c costante e $\exists$ f funzione computabile tale che per tutti gli input $\langle x, k\rangle$ A risolve π con tempo di calcolo $f(k) \cdot |x|^c$.*

- *$f(k)$ è la complessità del problema ed è esponenziale in k (es 2^k)*
- *$|x|^c$ è un tempo polinomiale*

Differenze:

- Se do in input x, con $|x| = n$, ad un algoritmo A di un problema NP-completo, avrò soluzione in tempo esponenziale 2^n.
- Se do in input x, k, con $|x| = n$, ad un algoritmo A di un problema NP-completo, avrò soluzione in tempo $f(k)n^c$ che è polinomiale in $|x| = n$ ed esponenziale in k, che abbiamo detto essere piccolo, quindi prende una parte molto ridotta dell'input.

11 Bounded Search Trees - alberi di ricerca limitati

Dato un grafo $G = (V, E)$, considero tutti i posibili sottoinsiemi di V che sono $2^{|V|}$, quindi costruisco dei sottoinsiemi di k vertici partendo da V.

Teorema 4. *il problema VC_d (G, k) è risolvibile in tempo $O(2^k \cdot |V|)$, dove V è l'insieme dei vertici di G; la costante non dipende nè da k nè da V. Si ha*

$$O * (f(k)) = O(2^k \cdot |V|)$$

dove 2^k è l'altezza dell'albero.

L'algoritmo, che ha costo $O(2^k \cdot |V|)$, è descritto come segue:

1. ho un input (G, k) e, partendo dalla radice, considero un arco di G a caso, ad esempio (u, v). Questo arco posso coprirlo o con u o con v, non con tutti e due i vertici insieme.
2. considero un altro arco di G, ad esempio (x, v). Questo arco non posso più coprirlo con v, quindi aggiungo questi due vertici alla copertura che NON contiene v e vado avanti così.

Esempio 3. *se dovessi considerare ad esempio l'arco (v, z) posso considerare v nella copertura che non lo contiene. Vedi figura seguente:*

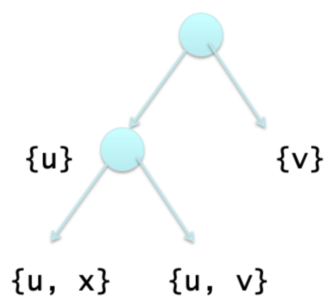

Figura 6: Esempio di albero per vertex-cover.

Dopo aver esplorato una profondità k, ho esaminato 2^k insiemi e ho trovato una copertura del grafo.
Se ho un percorso di lunghezza k allora non ho bisogno di esplorare oltre l'albero, cioè limito l'esplorazione dell'albero perchè ho già trovato la copertura.
L'algoritmo esplora un albero di dimensione 2^k e ad ogni livello si sceglie un arco. Nel caso peggiore si esaminano $|E|$ archi. La complessità di questo algoritmo sarà nel caso peggiore $O(2^k \cdot |E|)$ che può scendere a $O(2^k \cdot |V|)$ tramite delle ottimizzazioni.

12 TSP metrico

Traveling Salesman Problem. si vuole trovare il cammino di peso minimo che parte dal vertice v e torna nel vertice v (ciclo semplice) visitando una ed una sola volta tutti i vertici del grafo.
Il problema del commesso viaggiatore è un problema **NP-completo**, quindi è anche NP-difficile.

Hemiltonian Cycle. voglio trovare il cammino che parte dal vertice v e torna nel vertice v (ciclo semplice) visitando una ed una sola volta tutti i vertici del grafo.
Il ciclo Hamiltoniano è un problema **NP-completo**.

Cammino di Eulero. voglio trovare il cammino che parte dal vertice v e torna nel vertice v (ciclo) visitando una ed una sola volta tutti gli archi del grafo. I suoi vertici devono avere grado in-degree=out-degree.
Il cammino di Eulero è un problema **polinomiale**.

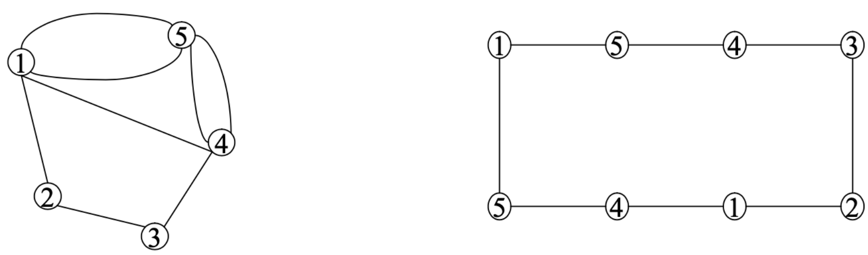

Figura 7: Ciclo Euleriano: sono coperti tutti i vertici e tutti gli archi sono attraversati una ed una sola volta.

Nel TSP metrico vale la **proprietà della distanza**/metrica:

- $c_{uv} \geqslant 0 \quad \forall (u, v) \in E$
- $c_{uv} = 0 \Leftrightarrow u = v$
- dev'essere soddisfatta la disuguaglianza triangolare: $c_{uv} \leqslant c_{ui} + c_{iv}$

Teorema 5 (di Eulero). *un grafo $G = (V, E)$ ammette un Cammino di Eulero (CE) se e solo se è connesso e ogni nodo ha grado (somma numero archi uscenti ed entrati) pari.*

Definizione 26. *dato un grafo $G = (N, A)$ non orientato, connesso e pesato e assumendo che $w(u, v) \in \mathbb{R}^+$ sia una funzione peso per G si ha che un **albero di copertura minimo** MST perè u G è un insieme di archi tale che:*

$$w(MST) = \min \sum_{(u,v)\in MST} w(u, v)$$

12.1 Algoritmo di Christofides

L'**algoritmo di Christofides** è diviso in 6 passi:

1. trova il minimo albero di copertura T di G
2. raddoppia ogni arco di T
3. costruisco il ciclo euleriano $\mathcal{E}$ di G (ha costo polinomiale!!)
4. scelgo un nodo iniziale $u \in \mathcal{E}$ e visito $\mathcal{E}$ da u
5. costruisco una permutazione π dei nodi V sequenziando i nodi nell'ordine della loro prima apparizione in $\mathcal{E}$ nella visita (tolgo i nodi doppioni)
6. restituisco il ciclo Hemiltoniano H associato a π

Vediamo meglio, passo passo, ogni step.

Step1: trova il minimo albero ricoprente T di G

Dato un grafo completo G, l'algoritmo greedy garantisce l'ottimo: basta scegliere gli archi in ordine di costo crescente ma saltando quelli che chiudono cicli.

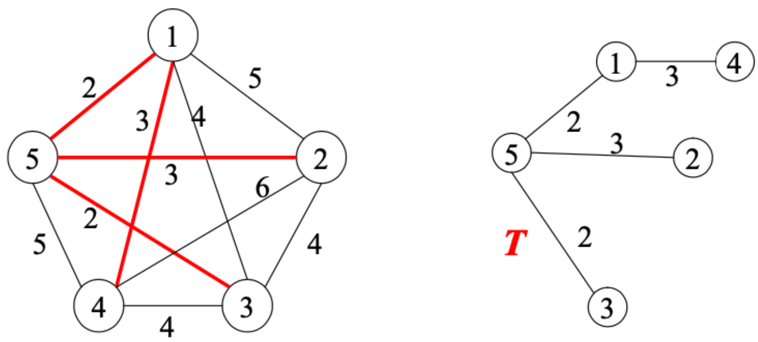

Figura 8: Dato un grafo completo e pesato G costruisco il suo albero di copertura T.

Step 2: raddoppia ogni arco di T ottenendo un grafo euleriano

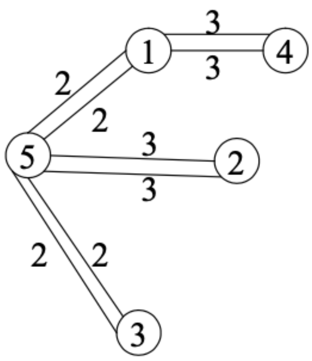

Figura 9: Raddoppio ogni arco di T.

Step 3: costruisci un ciclo euleriano E del grafo euleriano

Basta percorrere uno dopo l'altro tutti i cicli ottenuti dai rami dell'albero allo step 2.

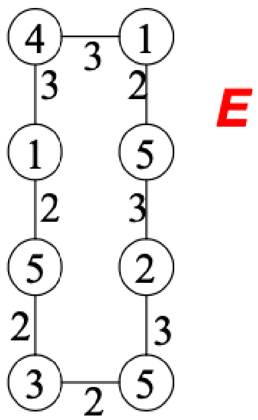

Figura 10: Costruzione del ciclo euleriano.

Step 4: scegli il nodo iniziale u in ε e visita ε a partire da u

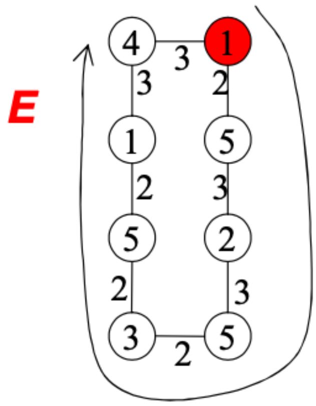

Figura 11: Scelgo 1 come nodo iniziale e visita di ε.

Step 5: costruisci una permutazione π dei nodi V sequenziando i nodi nell'ordine della loro prima apparizione in ε nella visita

Sequenza di visita di $\varepsilon :\ (1, 5, 2, 5, 3, 5, 1, 4)$

Permutazione associata $\pi :\ (1, 5, 2, 3, 4)$

Step 6: ciclo Hamiltoniano H associato a $\pi = (1, 5, 2, 3, 4)$

H è ottenuto da ε sostituendo cammini con archi.

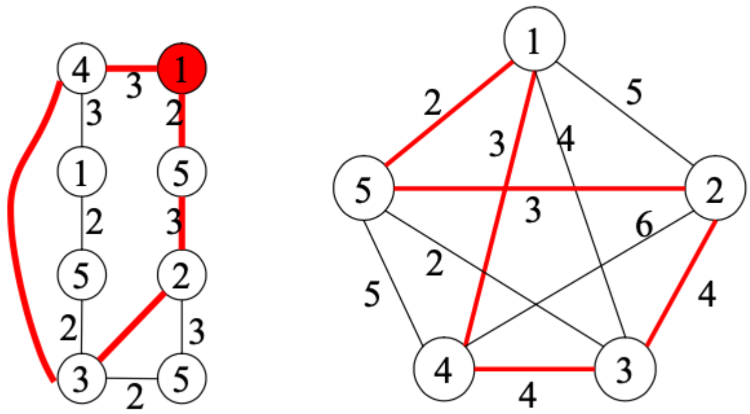

Figura 12: Cammino Hamiltoniamo ottenuto sostituendo i cammini ε con archi del grafo G.

Poichè c costo soddisfa la disuguaglianza triangolare, saltando dei nodi e seguendo delle "scorciatoie" si ha

$$c(H) \leqslant c(\varepsilon)$$

Siccome da E ad H ho vertici in meno posso dire che $costo(H) \leqslant costo(E)$, ovvero $c(H) \leqslant c(E)$.

Siccome E è ottenuto raddoppiando gli archi di T, si può concludere che $c(E) = 2 \cdot c(T)$.

Sia H^* un ciclo Hemiltoniano. Un albero di copertura T' (non necessariamente quello di copertura minimo) lo si ottiene prendendo H^* e togliendoci un arco e. Di conseguenza si ha $H^* = T' \cup \{e\}$, quindi $c(H^*) \geqslant c(T')$, perchè
$c(H^*) = c(T') + c(e)$.
Come già detto non è detto che T' sia l'albero di copertura minimo, quindi
$c(T) \leqslant c(T')$, dove T è l'albero di copertura minimo di H^*.
Mettendo in fila le disuguaglianze si ottiene $c(T) \leqslant c(T') \leqslant c(H^*)$, quindi si ha $c(T) \leqslant c(H^*)$.

In conclusione si ha:

- $c(H) \leqslant c(E)$
- $c(E) = 2 \cdot c(T)$
- $c(T) \leqslant c(H^*) \Rightarrow 2 \cdot c(T) \leqslant 2 \cdot c(H^*) \Rightarrow c(E) \leqslant 2 \cdot c(H^*)$

$\Longrightarrow c(H) \leqslant 2 \cdot c(H^*)$ e questo vuol dire che c'è una 2-approssimazione (si utilizza un *lower-bound*).