

Teoria della computazione

1. Problemi di decisione e macchine di Turing

Nella teoria della computazione si è interessati a classificare i problemi sulla base della loro difficoltà di risoluzione mediante strumenti o macchine di calcolo, dove per difficoltà di risoluzione si intende la difficoltà stimata rispetto all'uso di risorse di calcolo quali tempo e spazio.

I problemi classificati d'interesse sono quelli cosiddetti di **decisione** definiti come:

Data una funzione booleana $f : \{0, 1\}^* \rightarrow \{0, 1\}$, l'insieme dei linguaggi o problemi di decisione associati alla funzione f sono dati dall'insieme $L_f = \{x \in \{0, 1\}^* \mid f(x) = 1\}$

Si identifica inoltre il problema di **calcolare** f , ovvero dato $x \in \{0, 1\}^*$ calcolare $f(x)$, con il problema di **decidere** il linguaggio L_f , ovvero dato $x \in \{0, 1\}^*$ decidere se $x \in L_f$

Breve recap sulla definizione di difficoltà di risoluzione di un problema da parte di un algoritmo:

Date $f, g : \mathbb{N} \rightarrow \mathbb{N}$ allora diciamo che:

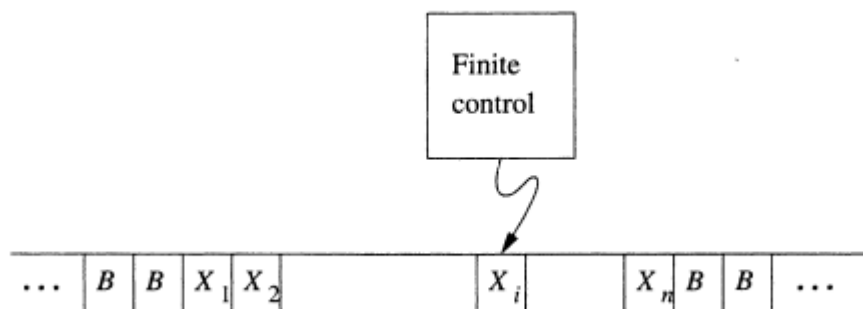
1. $f = O(g)$ se $\exists c, n_0 \in \mathbb{N} : f(n) \leq c \cdot g(n) \forall n \geq n_0$ ovvero se g limita "da sopra" f ($f(n) \leq c \cdot g(n)$) da un certo punto in avanti ($\forall n \geq n_0$)
2. $f = \Omega(g)$ se $g = O(f)$, ovvero se g limita "da sotto" f
3. $f = \Theta(g)$ se $f = O(g) \wedge g = O(f)$, ovvero se f è limitata "da sopra" e "da sotto" da g
4. $f = o(g)$ se $\forall \epsilon \in \mathbb{R}^+, f(n) \leq \epsilon \cdot g(n) \forall n \geq n_0$
5. $f = \omega(g)$ se $g = o(f)$

Come strumento o macchina di calcolo vengono utilizzate le **Macchine di Turing**.

Una macchina di Turing consiste in un *controllo finito*, un *nastro* diviso in *celle* ognuna delle quali può contenere un solo simbolo appartenente all'insieme dei simboli di nastro.

Inizialmente *l'input*, rappresentato da una stringa di lunghezza finita, viene posto sul nastro, un simbolo per cella. Tutte le altre celle contengono un simbolo detto *blank*.

La macchina è dotata di una *testina* che, posizionata su di una cella del nastro, legge o scrive un simbolo e può muoversi di una posizione a destra o a sinistra del simbolo letto/scritto.



Una TM M è definita come:

$M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$ dove:

- Q insieme finito degli *stati*
- Σ insieme finito dei simboli *in input*
- Γ insieme finito dei simboli di *nastro*
- $\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$ funzione di transizione che, presa in input una coppia composta dallo stato attuale e dal simbolo del nastro letto dalla testina, restituisce una tripla composta dallo stato successivo, il simbolo da scrivere sul nastro e il movimento che la testina deve eseguire (Left, Right)
- $q_0 \in Q$ è lo stato iniziale
- $B \in \Gamma$ è il simbolo di *Blank*
- F insieme di stati *finali* o *d'accettazione*

Sopra si è data la definizione di macchina di Turing *deterministica*, esiste anche una definizione di macchina di Turing *non deterministica* in cui la funzione di transizione δ ritorna un'insieme *finito* di triple del tipo $Q \times \Gamma \times \{L, R\}$

Lo stato complessivo di una TM si può dunque così rappresentare:

$$X_1 X_2 \dots X_{i-1} q X_i X_{i+1} \dots X_n$$

dove:

- $X = X_1 \dots X_n \in \Sigma^*$ è la stringa di input attualmente sul nastro
- $q \in Q$ è lo stato attuale del controllo finito
- X_i è il simbolo dell'input attualmente letto dalla testina da sinistra

Una mossa (descritta dalla funzione di transizione δ) è indicata dal simbolo \vdash , ad esempio se $\delta(q, X_i) = (p, Y, L)$ allora si avrà

$$X_1 X_2 \dots X_{i-1} q X_i X_{i+1} \dots X_n \vdash X_1 X_2 \dots X_{i-2} p X_{i-1} Y X_{i+1} \dots X_n$$

Una o più mosse di una TM sono indicate con \vdash^*

Una TM M va nello stato **halt** se non esistono transizioni da applicare, ovvero se $\delta(q, X_i) = \emptyset$ per un qualche $q \in Q, X_i \in \Sigma$.

Una TM M dunque:

- **Accetta** una stringa $w \in \Sigma^*$ se va in *halt* in uno stato finale, ovvero se $\delta(q, X_i) = \emptyset, q \in F, X_i \in \Sigma$
- **Rifiuta** una stringa $w \in \Sigma^*$ se va in *halt* in uno stato non finale, ovvero se $\delta(q, X_i) = \emptyset$ per un qualche $q \notin F, X_i \in \Sigma$ o se entra in un *loop infinito*

Definiamo dunque il **linguaggio accettato** da una TM M come:

Data una TM M

$$L(M) = \{w \in \Sigma^* \mid q_0 w \vdash^* \alpha p \beta, \text{ con } p \in F \text{ e } \alpha, \beta \in \Gamma\}$$

Essendo di **decisione** i problemi d'interesse per la teoria della computazione, si può dimostrare che ogni stringa $w \in \Sigma^*$ può essere tradotta in una stringa binaria $x_w \in \{0, 1\}^*$ e che per ogni macchina di Turing M che accetta stringhe $w \in \Sigma^*$ ne esiste un'altra M' che accetta le corrispondenti traduzioni binarie $x_w \in \{0, 1\}^*$.

Avevndo ora definito formalmente lo strumento/macchina di calcolo possiamo dare la definizione di **funzione calcolabile/computabile in tempo** $T(n)$, ovvero:

Siano $f : \{0, 1\}^* \rightarrow \{0, 1\}^*, T : \mathbb{N} \rightarrow \mathbb{N}$ e sia M una macchina di Turing. Allora diciamo che **M calcola/computa f in tempo $T(n)$** se $\forall x \in \{0, 1\}^*, q_0 x \vdash^* p f(x)$ con $p \in F$ in un numero di mosse al più pari a $T(|x| = n)$

Diciamo che **M calcola/computa f** se M calcola f in tempo $T(n)$ per qualche funzione $T : \mathbb{N} \rightarrow \mathbb{N}$

Dunque diciamo che

- un linguaggio L è **deciso (ricorsivo)** se esiste una macchina di Turing M che calcola la funzione $f_L : \{0, 1\}^* \rightarrow \{0, 1\}$ definita come $\forall x \in L, x \in L \implies f_L(x) = 1 \wedge x \notin L \implies f_L(x) = 0$
- un linguaggio L è **accettato (ricorsivamente enumerabile)** se esiste una macchina di Turing M che calcola la funzione $f_L : \{0, 1\}^* \rightarrow \{0, 1\}$ definita come $\forall x \in L, f_L(x) = 1 \iff x \in L$

Ovviamente se la funzione f_L è una funzione calcolabile in tempo $T(n)$, allora il linguaggio L diventa **deciso/accettato in tempo** $T(n)$

2. Classi di problemi

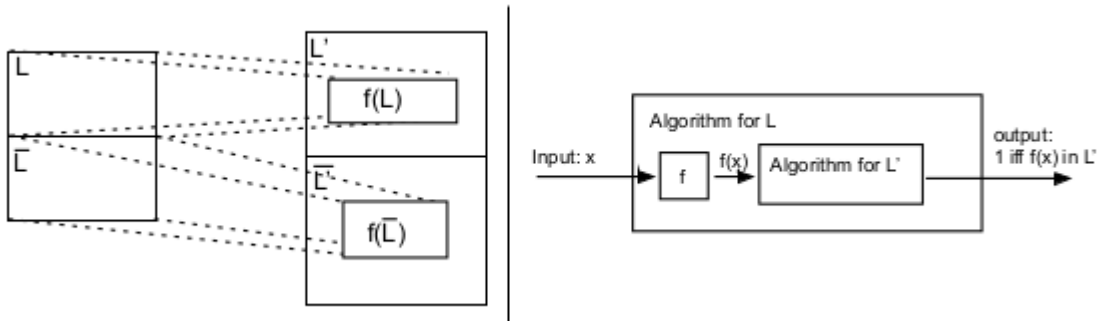
2.1. Definizioni

P = classe di problemi o linguaggi *accettati* in tempo $T(n) = c \cdot n^p$ da una TM M deterministica

NP = classe di problemi o linguaggi *accettati* in tempo $T(n) = c \cdot n^p$ da una TM M non deterministica, oppure più formalmente

NP = classe di linguaggi o problemi tali per cui esistono un polinomio $p : \mathbb{N} \rightarrow \mathbb{N}$ e una TM M deterministica tale che $\forall x \in \{0, 1\}^*, x \in L \subseteq \{0, 1\}^* \iff \exists u \in \{0, 1\}^{p(|x|)} : M(x, u) = 1$, ovvero si riesce a verificare in tempo polinomiale che un input x è *accettato* (o è un'istanza SI del problema) se viene presentata una prova u di questo fatto.

Un linguaggio $A \in \{0, 1\}^*$ si **riduce polinomialmente** ad un linguaggio $B \in \{0, 1\}^*$ ($A \leq_p B$) se esiste una *funzione computabile in tempo polinomiale* f tale che $\forall x \in \{0, 1\}^*, x \in A \iff f(x) \in B$



B è **NP-hard** se $\forall A \in \mathbf{NP}, A \leq_p B$

B è **NP-completo** se B è **NP-hard** e $B \in \mathbf{NP}$

Vale inoltre il seguente teorema:

1. $A \leq_p B \wedge B \leq_p C \implies A \leq_p C$
2. $A \in \mathbf{NP-hard} \wedge A \in \mathbf{P} \implies \mathbf{P} = \mathbf{NP}$
3. $A \in \mathbf{NP-completo} \implies (A \in \mathbf{P} \iff \mathbf{P} = \mathbf{NP})$

2.2. Problemi NP-completi

Alcuni esempi di problemi di decisione da noi trattati sono:

- **3-SAT**: data una formula booleana ϕ in *CNF form* (*Conjunctive Normal Form*), ovvero del tipo $\phi = c_1 \wedge c_2 \wedge \dots \wedge c_n$ dove ogni clausola c_i è la disgiunzione \vee di al più tre letterali (variabili logiche o

le loro negazioni \neg), stabilire se esiste un assegnamento alle variabili z tale che $\phi(z) = 1$, ovvero stabilire se ϕ è soddisfacibile

- **INDipendence-SET**: dato un grafo $G = (V, E)$ e un intero $k \in \mathbb{N}$, stabilire se esiste un sottoinsieme $I \subseteq V$ tale che $|I| \geq k \wedge \forall u, v \in I$ vale che $u, v \in I \implies (u, v) \notin E$, ovvero ci si chiede se esiste un sottoinsieme di almeno k vertici che presi a due a due non sono collegati da nessun arco in E
- **Vertex-Cover**: dato un grafo $G = (V, E)$ e un intero $k \in \mathbb{N}$, stabilire se esiste un sottoinsieme $V' \subseteq V$ tale che $|V'| \leq k \wedge \forall (u, v) \in E$ vale che $(u, v) \in E \implies u \in V' \vee v \in V'$, ovvero ci si chiede se esiste un sottoinsieme di al più k vertici che toccano tutti gli archi di G , formando appunto una copertura
- **Set-Cover**: dato un insieme universo U di n elementi, una collezione $S = \{S_1, S_2, \dots, S_m\}$ tale che $S_i \subseteq U \wedge \bigcup_{i=1}^m S_i = U$ e un intero $k \in \mathbb{N}$, stabilire se esiste una collezione $C \subseteq S$ tale che $|C| \leq k \wedge \bigcup_{j=1}^k C_j = U$
- **HAMILtonian-cycle**: dato un grafo $G = (V, E)$, stabilire se esiste un cammino che visita, partendo da un nodo $v \in V$ e tornando in v , ogni nodo di G esattamente una sola volta
- **TSP (Travelling Salesman Problem)**: dato un grafo $G = (V, E, w)$ completo e pesato, con pesi sugli archi dati da w , tali che $\forall e \in E, w(e) > 0$, e un intero $k \in \mathbb{N}$, stabilire se esiste, preso un vertice $v \in V$, un cammino da v a v che visita ogni nodo di G esattamente una sola volta e di costo $c \leq k$

Andremo ora a dimostrare che:

- $3\text{-SAT} \leq_p \text{IND-SET} \leq_p \text{VC} \leq_p \text{SC}$ (3-SAT è **NP**-completo per il [teorema di Cook-Levin](#))
- $\text{HAMIL} \leq_p \text{TSP}$ (non so chi, ma qualcuno ha sicuramente dimostrato che HAMIL è **NP**-completo)

Per i problemi di cui sopra è stata fornita la variante decisionale, in cui viene posto il problema di "stabilire se esiste...". Per ognuno di essi si può enunciare il problema di ottimo associato, che mira a trovare la soluzione più generale possibile. Ad esempio VC sarà così formulato: "Qual'è il più piccolo insieme tale che...", mentre per IND-SET avremo: "Qual'è il più grande insieme tale che..."

2.2.1. $3\text{-SAT} \leq_p \text{IND-SET}$

Data un'istanza I di 3-SAT, un grafo $G = (V, E)$ e un intero $k \in \mathbb{N}$, I è soddisfacibile $\iff G$ ha un independent set di cardinalità k .

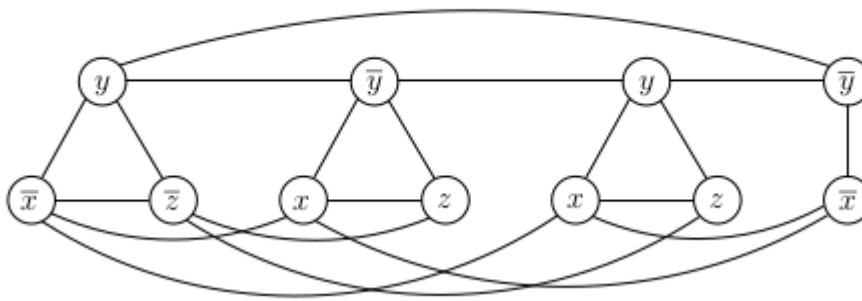
2.2.1.1. Parte 1

Data un'istanza $I \in 3\text{-SAT}$ creiamo un'istanza $(G, k) \in \text{IND-SET}$ in questo modo:

- il grafo G ha un triangolo per ogni clausola, con i tre vertici etichettati con i letterali di quest'ultima e collegati tra loro da un arco. Si collega inoltre ogni letterale con il suo negato (se esiste)

- Poniamo k uguale al numero di clausole in I

Figure 8.8 The graph corresponding to $(\bar{x} \vee y \vee \bar{z}) (x \vee \bar{y} \vee z) (x \vee y \vee z) (\bar{x} \vee \bar{y})$.



2.2.1.2. Parte 2

Ora dobbiamo dimostrare che:

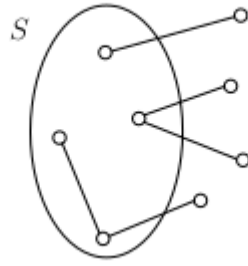
1. $\text{IND-SET} \in \mathbf{NP}$, ma data una soluzione è facile verificare in tempo polinomiale che questa soddisfa IND-SET
2. Se $S \subseteq V$ è un independence set di k vertici allora I è soddisfacibile. Per ogni letterale x , S non può contenere sia x che $\neg x$, poichè tali vertici sono collegati da un arco per costruzione di G , inoltre essendo che S ha cardinalità k , sempre per costruzione di G conterrà un solo letterale per clausola. Dunque ponendo $x = 1$ se S contiene un vertice etichettato con x o $x = 0$ se S contiene un vertice etichettato con $\neg x$, si ottiene un assegnamento che rende veri tutti i letterali contenuti in S , rendendo dunque vera I
3. Se I ha un assegnamento che la rende soddisfacibile allora G ha un independence set di cardinalità k . Per ogni clausola prendo un letterale x tale che $x = 1$ (ne esiste almeno uno per clausola) e lo aggiungo a S . Per le stesse considerazioni fatte al punto 1. S è un independence set

2.2.2. $\text{IND-SET} \leq_p \text{VC}$

Avendo dimostrato che IND-SET è un problema \mathbf{NP} -completo e notando che se un insieme $S \subseteq V$ è un insieme indipendente per un qualche grafo $G = (V, E)$, allora l'insieme $V - S$ è una copertura per G , la dimostrazione che $\text{IND-SET} \leq_p \text{VC}$ risulta [triviale](#) (vale anche il viceversa).

Infatti basta dimostrare che, dato un grafo $G = (V, E)$, un insieme $S \subseteq V$ è un insieme indipendente di $G \iff V - S$ è una copertura per G .

Figure 8.9 S is a vertex cover if and only if $V - S$ is an independent set.



1. Se un insieme $S \subseteq V$ è un insieme indipendente di G allora $V - S$ è una copertura per G . Per definizione di insieme indipendente vale che non esiste nessun arco $(u, v) \in E$ tale che $u \in S \wedge v \in S$. Allora deve valere che $u \in V - S \vee v \in V - S$, allora $V - S$ è una copertura.
2. Se $V - S$ è una copertura per G allora S è un insieme indipendente. Per definizione di copertura, $\forall (u, v) \in E, u \in V - S \vee v \in V - S$, dunque non può mai succedere che entrambi u e v appartengano a S

2.2.3. $VC \leq_p SC$

Dobbiamo dimostrare che esiste una funzione f che mappi ogni istanza di VC in un'istanza di SC, ovvero che esiste una funzione $f(G, k) = (U, S, l)$ per cui G ha vertex cover di k elementi $\iff U$ ha un set cover di l elementi.

2.2.3.1. Parte 1

Avendo numerato i vertici di V da 1 a n , costruiamo f in questo modo:

- $U = E$
- $S = \{S_1, S_2, \dots, S_n\}$ è tale che $S_i = \{\text{archi incidenti al vertice } i\}, \forall i = 1, \dots, n$
- $k = l$

2.2.3.2. Parte 2

Ora dobbiamo dimostrare che:

1. $SC \in \mathbf{NP}$. Avendo una collezione C' è banale provare in tempo polinomiale che questa sia una soluzione per SC.
2. Se G ha una copertura di k vertici allora U ha una copertura di $l = k$ vertici. Supponiamo C sia una copertura per G di cardinalità k , allora per costruzione, ad essa corrisponde un insieme $C' \subseteq S$ di pari cardinalità. Essendo $U = E$ ed essendo C una copertura per $G, \forall u \in U$ vale che uno dei suoi estremi appartiene a C , dunque C' contiene almeno un insieme associato agli estremi di u , e per definizione, entrambi contengono u . (Oppure, essendo C una copertura di G , per definizione essa forma un insieme i cui elementi toccano tutti gli archi di E . Sia $C' \subseteq U$ l'insieme

formato dalle collezioni associate ai vertici di C ($|C'| = |C| = k$). Ogni $C_j \in C'$ contiene gli archi incidenti al vertice j , la loro unione è dunque U)

3. Dimostriamo ora il viceversa. Sia C' una copertura di U . Allora $\forall u \in U$ sicuramente C' contiene almeno un insieme che include u . Tale insieme corrisponde ad un nodo che è estremo di u (poichè contiene i suoi incidenti), quindi C deve contenere almeno un estremo di u

2.2.4. $\text{HAMIL} \leq_p \text{TSP}$

Dati due grafi $G = (V, E)$ e $G' = (V', E', w)$ e due interi $k, c \in \mathbb{N}$ con $c > k$, G ha un ciclo hamiltoniano di lunghezza $k \iff G'$ ha un ciclo che visita tutti i nodi di G' una sola volta e di costo $k|V|$

2.2.4.1. Parte 1

Dobbiamo prima di tutto costruire una funzione che mappi ogni istanza di HAMIL in un'istanza di TSP in modo che HAMIL risponde 1 \iff TSP risponde 1.

Dunque:

- $V' = V$
- $E' = \{(u, v) \mid u, v \in V \wedge u \neq v\}$
- $w : E' \rightarrow \{k, c\}$ è tale che $\forall e \in E'$ vale che $e \in E \implies w(e) = k \wedge e \notin E \implies w(e) = c$

2.2.4.2. Parte 2

Si dimostra dunque che:

1. $\text{TSP} \in \mathbf{NP}$, ma avendo un cammino e un intero $h \in \mathbb{N}$ è facile verificare in tempo polinomiale che questo è soluzione di TSP
2. Se esiste un ciclo che visita tutti i nodi di G' una e una sola volta e di costo $k|V|$ ciò significa che deve per forza essere costituito da archi di peso k , infatti se ci fosse anche solo un arco di peso c allora il costo complessivo supererebbe $k|V|$. Tali archi sono anche in G e quindi esiste il ciclo hamiltoniano.
3. Se esiste un ciclo hamiltoniano in G , questo è sicuramente costituito da $|V|$ archi. Percorrendo gli stessi archi in G' otteniamo un cammino lungo $|V|$ archi tutti di peso k , quindi di costo $k|V|$

2.3. Algoritmi ϵ -approssimanti

I problemi \mathbf{NP} -hard sono i più difficili problemi di ottimizzazione, dunque, a meno che $\mathbf{P} = \mathbf{NP}$ non esistono algoritmi efficienti che li risolvano.

Entrano dunque in gioco i cosiddetti algoritmi ϵ -approssimanti, che trovano una soluzione ammissibile

del problema in tempo polinomiale che dista dalla soluzione ottima per una fattore ϵ .

Dunque:

Dato un problema π e un'istanza x , indichiamo con $\mathbf{opt}(x)$ il costo della *soluzione ottima* di π sull'istanza x , mentre con $\mathcal{A}(x)$ il costo della *soluzione ammissibile* su x calcolato da un algoritmo \mathcal{A}

Un **algoritmo ϵ -approssimato** per un problema π di ottimizzazione è un algoritmo \mathcal{A} *polinomiale* tale che restituisce una soluzione ammissibile che dista dalla soluzione ottima di un fattore costante ϵ , ovvero:

- $\mathbf{opt}(x) < \mathcal{A}(x) \leq \epsilon \cdot \mathbf{opt}(x)$, per $\epsilon > 1$ se π è un *problema di minimo*
- $\mathbf{opt}(x) > \mathcal{A}(x) \geq \epsilon \cdot \mathbf{opt}(x)$, per $0 < \epsilon < 1$ se π è un *problema di massimo*

Un *polynomial-time approximation scheme (PTAS)* è una famiglia di algoritmi $\{A_\epsilon\}$ in cui esiste un algoritmo $\forall \epsilon > 0$, tale che A_ϵ è un algoritmo $(1 + \epsilon)$ -approssimato (per un problema di minimo) o un algoritmo $(1 - \epsilon)$ -approssimato (per un problema di massimo)

Esistono problemi di ottimizzazione che **non ammettono** un algoritmo ϵ -approssimato

2.3.1. 2-approssimazione per VC

Dato un grafo $G = (V, E)$, definiamo un algoritmo 2-approssimato per VC in questo modo:

1. $C = \emptyset$
2. Scelto un arco $e = (u, v) \in E$, $C = C \cup \{u, v\}$
3. Rimuovi da E gli archi incidenti a u e v
4. Torna al punto 2. se $E \neq \emptyset$

Questo algoritmo restituisce una copertura di dimensione $2|M|$, ovvero $\mathcal{A}(x) = 2|M|$, dove M è l'insieme degli archi scelti dall'algoritmo.

OSSERVAZIONE: L'insieme M degli archi scelti forma un cosiddetto *matching*, ovvero un insieme di archi che non condividono vertici.

In questo caso M è un *maximal matching*, ovvero è tale per cui $\forall e \notin M, M \cup \{e\} \implies M$ non è più un *matching*.

Dunque l'algoritmo di cui sopra può essere così riformulato:

1. Trova un *maximal matching* M di G
2. $C = \{u, v \in V \mid (u, v) \in M\}$

Dimostriamo prima di tutto che una minima copertura di vertici $V' \subseteq V$ ha dimensione maggiore o uguale a quella del *matching* M .

Una copertura ottima di vertici deve coprire ogni arco in M , dunque vale che $\forall (u, v) \in M, u \in V' \vee v \in V'$, quindi varrà che $\mathbf{opt}(x) \geq |M|$.

in definitiva si avrà che $2 \cdot \mathbf{opt}(x) \geq 2|M| = \mathcal{A}(x)$

2.3.2. TSP non ha un' ϵ -approssimazione

Nel problema del TSP, nella sua versione di ottimizzazione, viene fornito un grafo $G = (V, E, w)$, con $w(e) > 0 \forall e \in E$, e si chiede di trovare un ciclo hamiltoniano di costo minimo (un ciclo è detto hamiltoniano se visita ogni vertice in V esattamente una sola volta).

TSP è un problema **NP**-hard, dunque non possiamo trovare un algoritmo che approssima TSP in tempo polinomiale a meno che **P** = **NP**, questo perchè se un tale algoritmo esistesse allora potremmo risolvere in tempo polinomiale HAMIL, che è **NP**-completo, il che non è possibile a meno che **P** = **NP**.

Vale dunque il seguente teorema:

■ TSP non ha un'approssimazione costante a meno che **P** = **NP**

Sia $G = (V, E)$ un grafo di cui vogliamo determinare se esiste un ciclo hamiltoniano. Costruiamo l'input $G' = (V, E', w)$ per TSP in questo modo:

- $E' = \{(u, v) \mid u, v \in V \wedge u \neq v\}$
- $\forall e \in E', w(e)$ è tale che $e \in E \implies w(e) = 1 \wedge e \notin E \implies w(e) = |V|/(1 - 1/\epsilon)$

Assumiamo esista un algoritmo \mathcal{A} polinomiale che sia ϵ -approssimante per TSP.

Si avranno allora due casi:

1. \mathcal{A} restituisce cammino di costo pari a $|V|$, ciò significa che sono stati scelti solo archi di peso 1 e quindi il ciclo hamiltoniano esiste
2. \mathcal{A} restituisce un cammino di costo $> |V|$, ciò significa che il cammino include almeno uno degli archi di peso $|V|/(1 - 1/\epsilon)$, dunque il costo totale sarà almeno $|V| - 1 + |V|/(1 - 1/\epsilon)$.

Semplificando si ottiene $\mathcal{A}(x) > \epsilon|V|$, ma per definizione di ϵ -approssimazione si ha che $\epsilon \cdot$

$\mathbf{opt}(x) \geq \mathcal{A}(x) > \epsilon|V|$, ovvero $\mathbf{opt}(x) \geq \mathcal{A}(x)/\epsilon > |V|$, ovvero il cammino ottimo ha costo strettamente maggiore di $|V|$, quindi il ciclo hamiltoniano non esiste poichè non c'è modo di ottenere un cammino di costo $|V|$

Pertanto \mathcal{A} diventa un algoritmo che decide in tempo polinomiale se, dato un grafo $G = (V, E)$, esiste un ciclo hamiltoniano, ed essendo HAMIL un problema **NP**-completo, allora **P** = **NP**, poichè potrei ridurre ogni problema in **NP** ad HAMIL e risolverlo polinomialmente.

2.3.3. TSP metrico

Abbiamo appena dimostrato che TSP, nella sua versione di ottimizzazione, non ha un' ϵ -approssimazione.

Viene ora fornita una variante di TSP chiamata TSP metrico, che si basa appunto sul concetto di *metrica*:

una **distanza** (o **metrica**) $d : X \times X \rightarrow \mathbb{R}$ è una funzione che soddisfa le seguenti proprietà $\forall x, y, z \in X$:

1. $d(x, y) \geq 0$
2. $d(x, y) = 0 \iff x = y$
3. $d(x, y) = d(y, x)$
4. $d(x, y) \leq d(x, z) + d(z, y)$

Dunque TSP metrico è tale per cui, dato un grafo $G = (V, E, w)$ tale che w è una metrica, si chiede di trovare un ciclo hamiltoniano di costo minimo.

Si dimostra che tale variante di TSP ha un algoritmo 2-approssimante.

Diamo innanzitutto dunque la seguente definizione:

un **cammino euleriano** è un cammino che utilizza ogni arco esattamente una volta

Vale inoltre il seguente teorema:

Ogni grafo non orientato ha un *ciclo euleriano* se e solo se ogni vertice ha grado pari e tutti i vertici di grado > 0 appartengono ad una singola componente connessa

Inoltre:

Ogni grafo connesso e con ogni nodo di grado pari è detto *grafo euleriano*

2.3.3.1. Algoritmo 1

Dunque, un possibile algoritmo 2-approssimante per TSP metrico è il seguente:

1. Calcola il Minimum Spanning Tree T^* di G (polinomiale)
2. Raddoppia ogni arco di T^* ottenendo un grafo euleriano
3. Trova un ciclo euleriano \mathcal{E} (polinomiale)
4. Costruisci un ciclo hamiltoniano \mathcal{H} da \mathcal{E} , partendo da un vertice arbitrario $u \in \mathcal{E}$, procedendo su \mathcal{E} e saltando i nodi già presenti in \mathcal{H} (ciò è lecito poichè w è una metrica, e in particolare vale la disuguaglianza triangolare)

Tale algoritmo è una 2-approssimazione per TSP metrico, infatti vale che:

1. $c(\mathcal{H}) \leq c(\mathcal{E})$, per la disuguaglianza triangolare
2. $c(\mathcal{E}) = 2c(T^*)$, avendo raddoppiato gli archi di T^* per ottenere \mathcal{E}

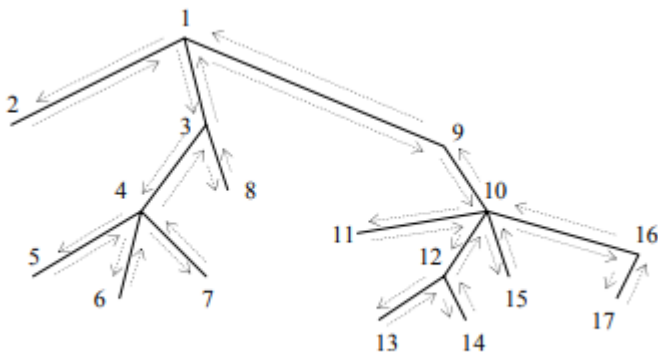
3. $c(T^*) \leq c(\mathcal{H}^*)$, infatti, preso il ciclo hamiltoniano di costo ottimo e rimuovendo un arco, otteniamo l'MST ottimo, che non può costare più del ciclo

Dunque si ha: $c(\mathcal{H}) \leq c(\mathcal{E}) = 2c(T^*) \leq 2c(\mathcal{H}^*)$, ovvero il costo del ciclo hamiltoniano trovato dall'algoritmo non supera due volte il costo del ciclo ottimo.

2.3.3.2. Algoritmo 2

Un'altro algoritmo 2-approssimante per TSP metrico è il seguente:

1. Calcola il Minimum Spanning Tree T^* di G
2. Scegli un nodo root $r \in T^*$ e visita in *pre-order* l'albero: v_1, v_2, \dots, v_n
3. Ritorna il tour: $v_1 \rightarrow v_2 \rightarrow \dots \rightarrow v_n \rightarrow v_1$



Dimostriamo che l'algoritmo è una 2-approssimazione per TSP metrico.

Sia σ una visita completa in pre-order di T^* , ovvero rivisitiamo i vertici ogni volta che risaliamo all'indietro l'albero.

Come si evince dalla figura di cui sopra, $c(\sigma) = 2c(T^*)$ ed essendo il tour ritornato da \mathcal{A} una sottosequenza di σ , per la disuguaglianza triangolare si avrà che $\mathcal{A}(x) \leq 2c(T^*)$.

Inoltre, supponiamo σ^* sia un tour ottimo, dunque $\mathbf{opt}(x) = c(\sigma^*)$. Cancellando da σ^* un arco otteniamo uno spanning tree T tale che $c(T) \geq c(T^*)$. Varrà quindi che $\mathbf{opt}(x) = c(\sigma^*) \geq c(T) \geq c(T^*)$

In definitiva, combinando i risultati si avrà $\mathcal{A}(x) \leq 2c(T^*) \leq 2 \cdot \mathbf{opt}(x)$