"Sapienza" Università di Roma - Dipartimento di Ingegneria Informatica, Automatica e Gestionale

Euristiche per il Problema del Commesso Viaggiatore

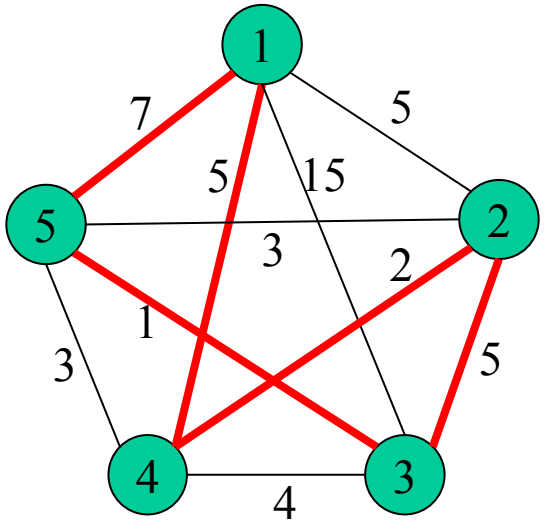
Renato Bruni
bruni@dis.uniroma1.it

Il materiale presentato è derivato da quello dei proff. A. Sassano e C. Mannino

Il Problema del Commesso Viaggiatore

- Problema del **Commesso Viaggiatore** (Traveling Salesman Problem, **TSP**): bisogna visitare una serie di clienti e tornare al punto di partenza seguendo il percorso meno costoso
- Molti altri problemi pratici hanno questa struttura (es. spostare un macchinario che deve lavorare in tanti punti di un oggetto, passare un cavo che deve collegare vari punti, etc.)

- Abbiamo un **costo** per ogni possibile "collegamento"
- Il costo del percorso è la somma dei costi dei collegamenti percorsi (nell'esempio 7+5+2+5+1 = 20)

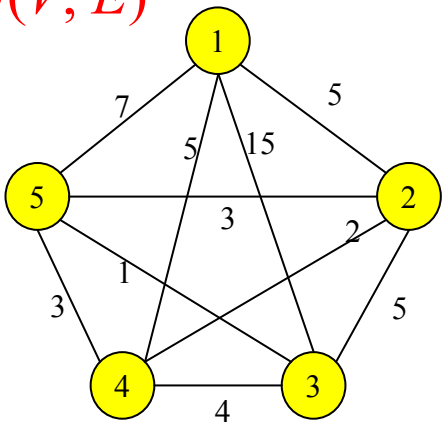


Ciclo Hamiltoniano di peso minimo

- Per modellarlo, consideriamo un grafo $G(N,E)$ completo (se non orientato ha $\binom{n}{2}$ cioè $n(n\text{-}1)/2$ archi, se orientato ne ha il doppio $n(n\text{-}1)$), con costi sugli archi $c \in R^E$
- Un ciclo **hamiltoniano** è un ciclo semplice che passa per ogni nodo di G. Il costo del ciclo è pari alla somma dei costi dei suoi archi
- Il problema del commesso viaggiatore (Travelling Salesman Problem, **TSP**) consiste nel trovare un **ciclo hamiltoniano di costo minimo**
- Se il grafo è non orientato, si parla di **TSP simmetrico** (quello che vedremo in dettaglio); se invece il grafo è orientato si parla di TSP asimmetrico
- Il problema appartiene alla classe di problemi combinatori con funzione obiettivo lineare ed alla casse dei problemi difficili (**NP-hard**)
- In questa lezione ci concentriamo su algoritmi euristici per il TSP simmetrico (l'estensione al caso asimmetrico è in molti casi ovvia)

Il TSP è un problema di OC

$G(V, E)$

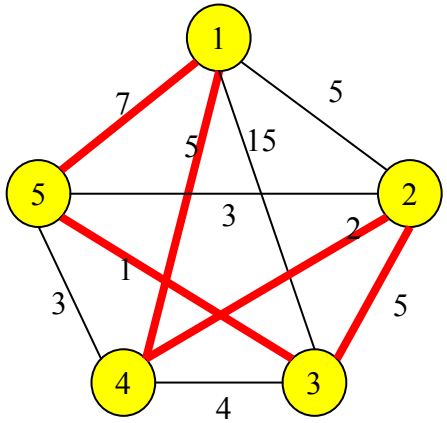


- Grafo $G(V, E)$ non orientato, completo
- c_e costo arco $e \in E$

L'insieme base è l'insieme degli archi E

Una soluzione ammissibile è $T \subseteq E$

Costo di $T = c(T) = \sum_{e \in T} c_e$



$T = \{(1,4), (4,2), (2,3), (3,5), (5,1)\}$

$c(T) = 5+2+5+1+7 = 20$

Algoritmo Greedy (= Avido)

- Conosciamo l'insieme base $E = \{1,2,\ldots,n\}$
- Sappiamo verificare se una soluzione appartiene all'insieme delle soluzioni ammissibili $\mathcal{S} = \{T_1, T_2, \ldots, T_m\}$ $(T \subseteq E)$

Se $H \subseteq T$ → H è soluzione parziale (può diventare ammissibile aggiungendo elementi)

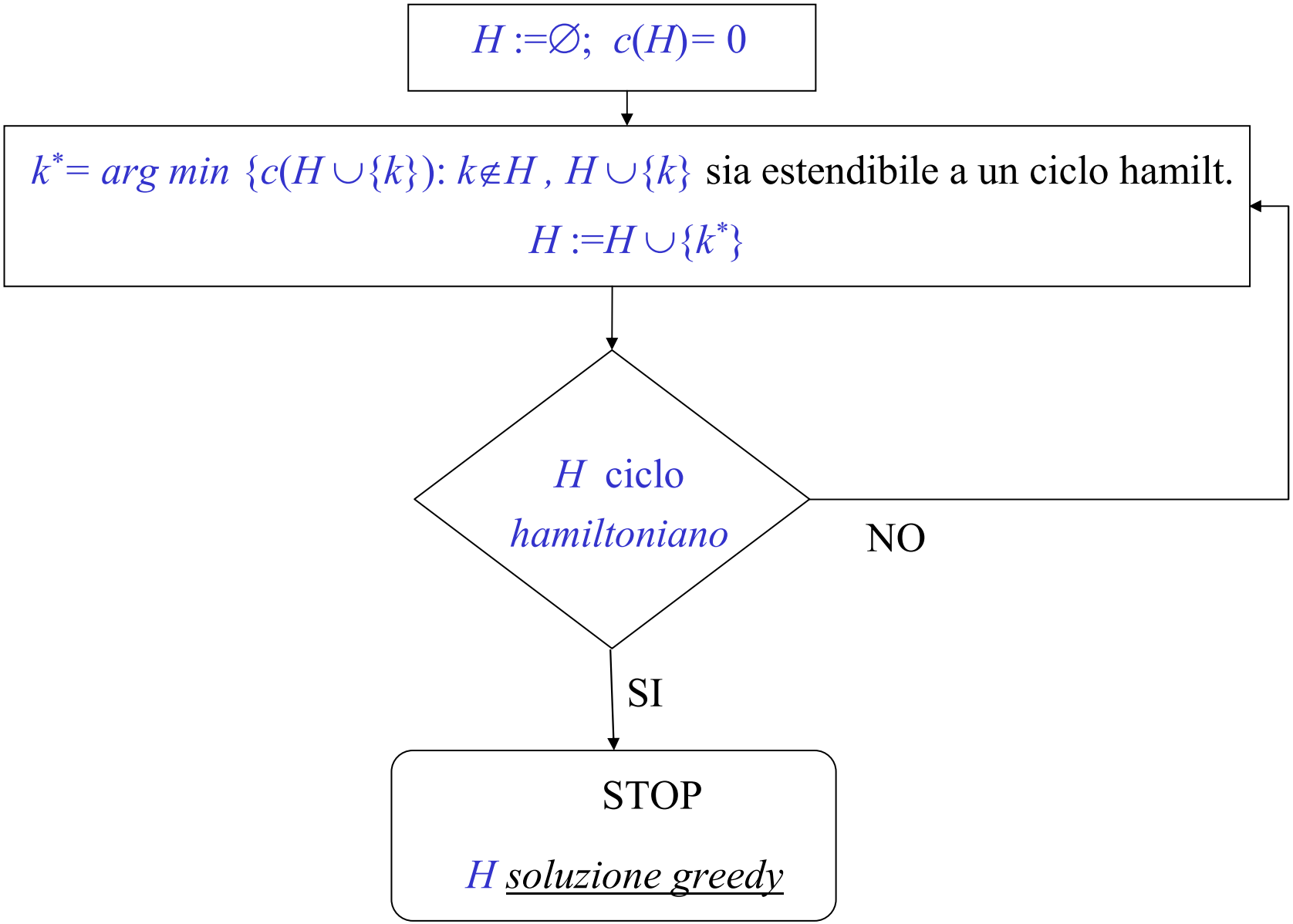
- *Costo di una soluzione parziale* $H = c(H) = \sum_{e \in H} c(e)$

Linee guida dell' Algoritmo Greedy, adattabile a qualsiasi problema di OC

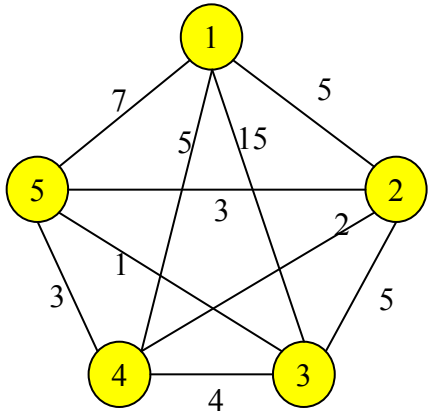
Costruire una sequenza di soluzioni parziali H_0 H_1 H_2 H_3 ...:

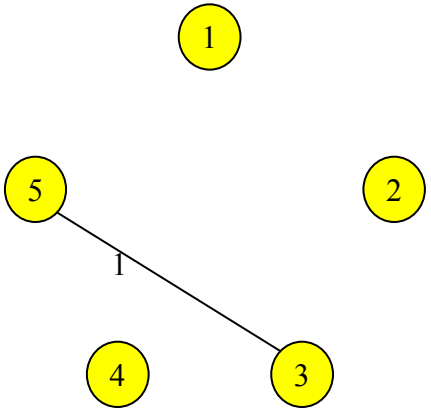
a. a partire dall'insieme vuoto (*soluzione parziale* H_0)
b. aggiungendo, ad ogni passo, l'elemento che produce la soluzione parziale con il minimo costo.
c. arrestandosi quando la soluzione parziale corrente è una soluzione ammissibile (o quando peggiora aggiungendo altri elementi, per problemi in cui la soluzione vuota è ammissibile)

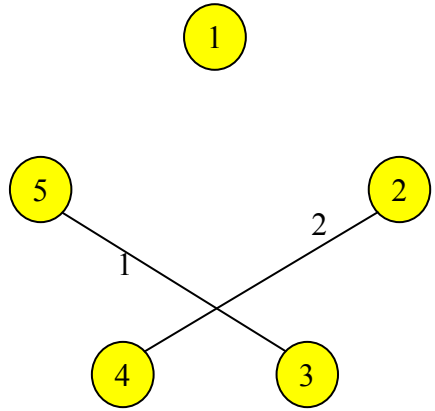
Struttura Greedy per il TSP

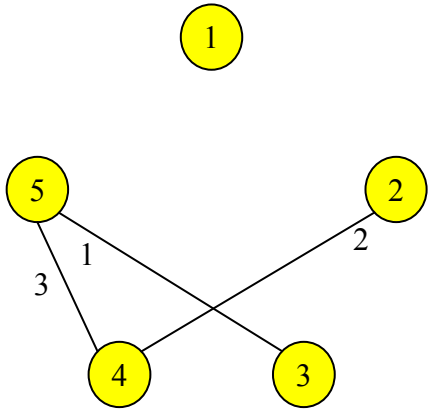


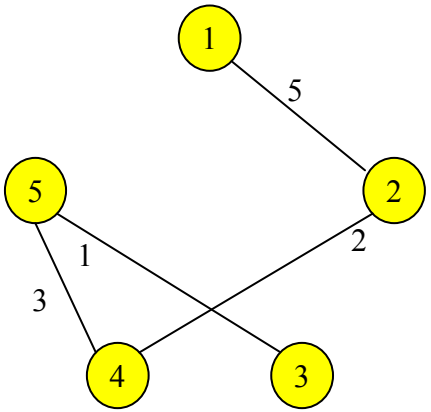
Esempio Applicazione Greedy al TSP

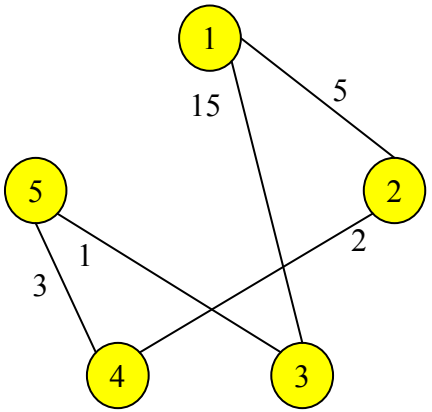












$c(T) = 26$

Osservazioni sul Greedy

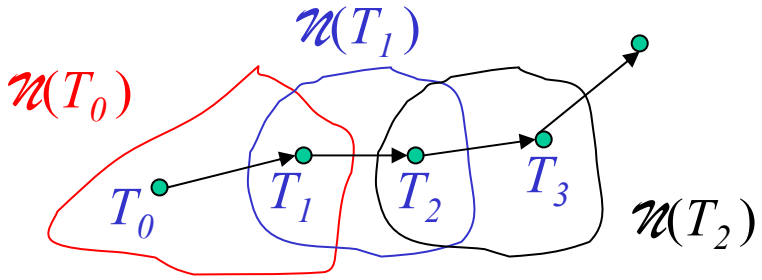
- È una euristica **costruttiva** (costruisce una soluzione)
- Quando un elemento è stato scelto non viene più **abbandonato**
- Ciò rende l'algoritmo **veloce**, ma la scelta degli elementi più **convenienti** sul momento può **obbligare** in seguito a dover prendere elementi **poco convenienti** (nell'esempio visto, alla fine si è obbligati a prendere l'arco che costa 15, cioè il più costoso del grafo)
- Pertanto, la soluzione trovata nel caso del problema del Commesso Viaggiatore può essere anche molto scadente

Algoritmo di Ricerca Locale (Local Search)

- Le soluzioni ammissibili formano un insieme $\mathcal{S} = \{T_1, T_2, \ldots, T_m\}$ $(T_i \subseteq E)$
- Per ogni soluzione ammissibile T posso allora definire un *intorno* $\mathcal{N}(T) \subseteq \mathcal{S}$

(l'insieme delle sol. ammiss. simili a T)

Idea base



Costruire una sequenza di soluzioni ammissibili T_0, T_1, T_2, T_3 ...:

a. a partire da una soluzione ammissibile T_0

b. individuando, al passo k, la **soluzione di minimo costo** T_k appartenente all'intorno $\mathcal{N}(T_{k-1})$ della soluzione corrente T_{k-1}

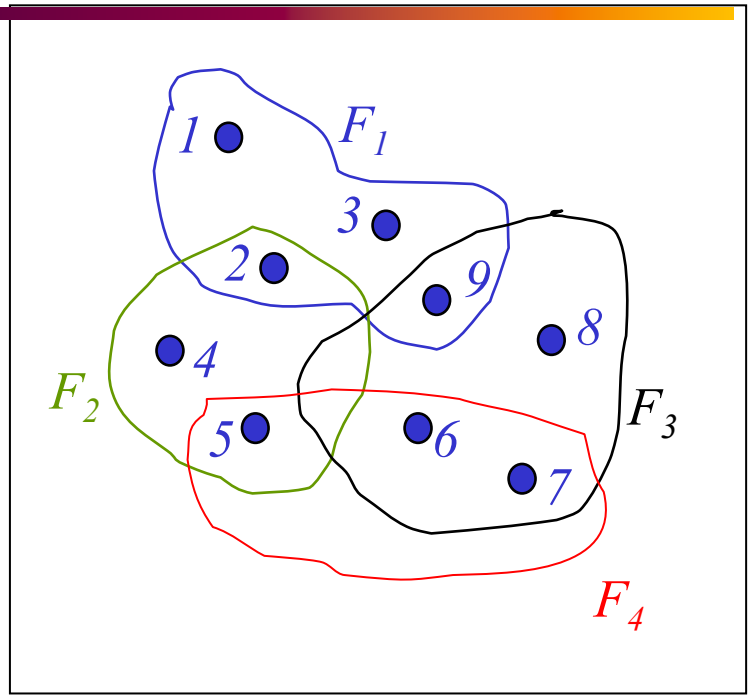
c. arrestandosi quando ogni soluzione appartenente all'intorno $\mathcal{N}(T_{k-1})$ ha un valore della funzione obiettivo **peggiore** (maggiore) di $c(T_{k-1})$ ovvero $c(T_{k-1}) < c(T_k)$

Sistemi di Intorni

Esempi di intorni

$\mathcal{N}(F_1)=\{F_2, F_4\}$ $\mathcal{N}(F_2)=\{F_1, F_4\}$

$\mathcal{N}(F_4)=\{F_1, F_2, F_3\}$ $\mathcal{N}(F_3)=\{F_2\}$



- Serve un sistema di intorni, cioè un criterio per avere ogni volta l'intorno della soluzione. Ne esistono vari:

$I=\{\mathcal{N}(F_i): F_i \in \mathcal{S}\}$ *Sistema di Intorni in* $\mathcal{S}$

$F \in \mathcal{N}_+(F_i) \Leftrightarrow F = F_i \cup k : k \notin F_i, F \in \mathcal{S}$ *Intorno "greedy"*

$F \in \mathcal{N}_-(F_i) \Leftrightarrow F = F_i - k : k \in F_i, F \in \mathcal{S}$ *Intorno "reverse greedy"*

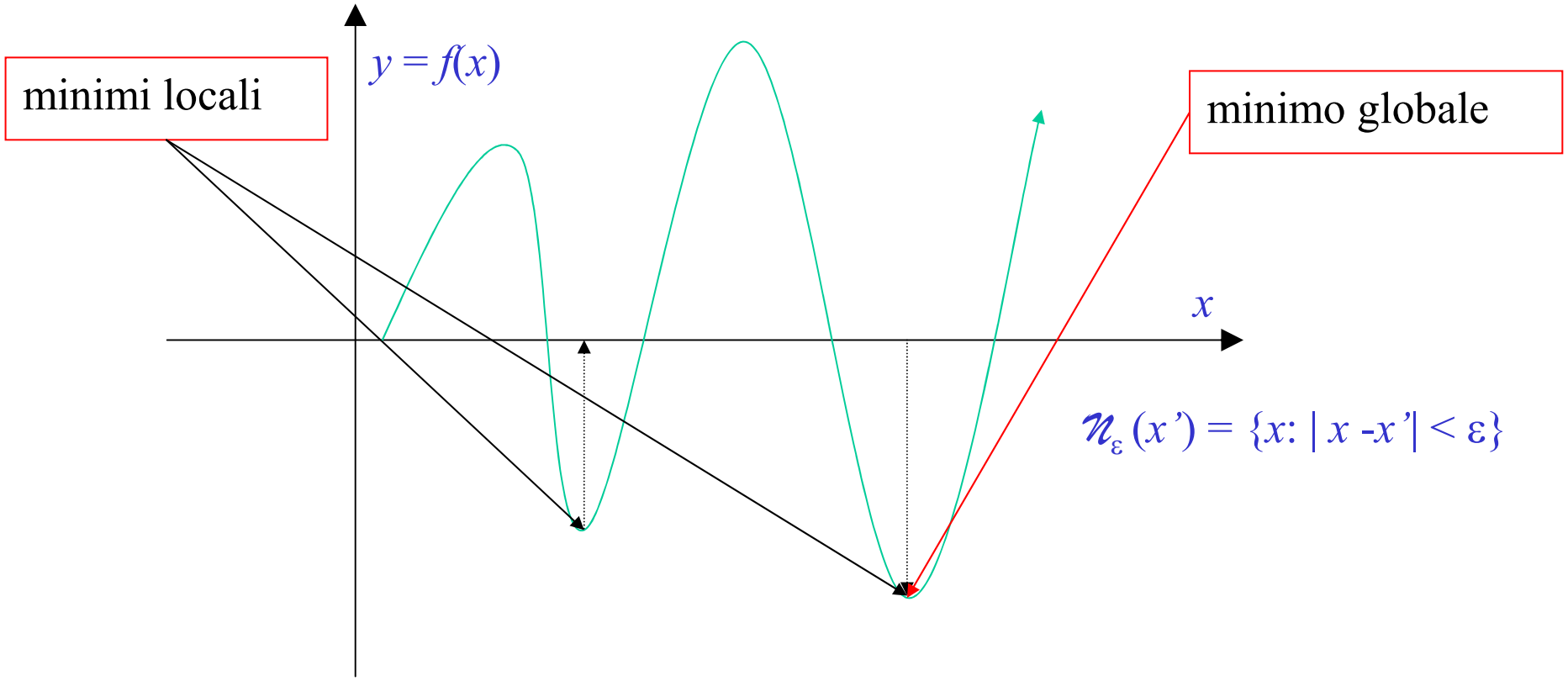
$F \in \mathcal{N}_s(F_i) \Leftrightarrow F_i - \{k\} \cup \{j\} : k \in F_i, j \notin F_i, F \in \mathcal{S}$ *Intorno di "scambio"*

$F \in \mathcal{N}_s(F_i) \Leftrightarrow F_i - \{h,k\} \cup \{j,i\} : h,k \in F_i, j,i \notin F_i, F \in \mathcal{S}$ *Intorno "2-scambio"*

Minimi Locali e Minimi Globali

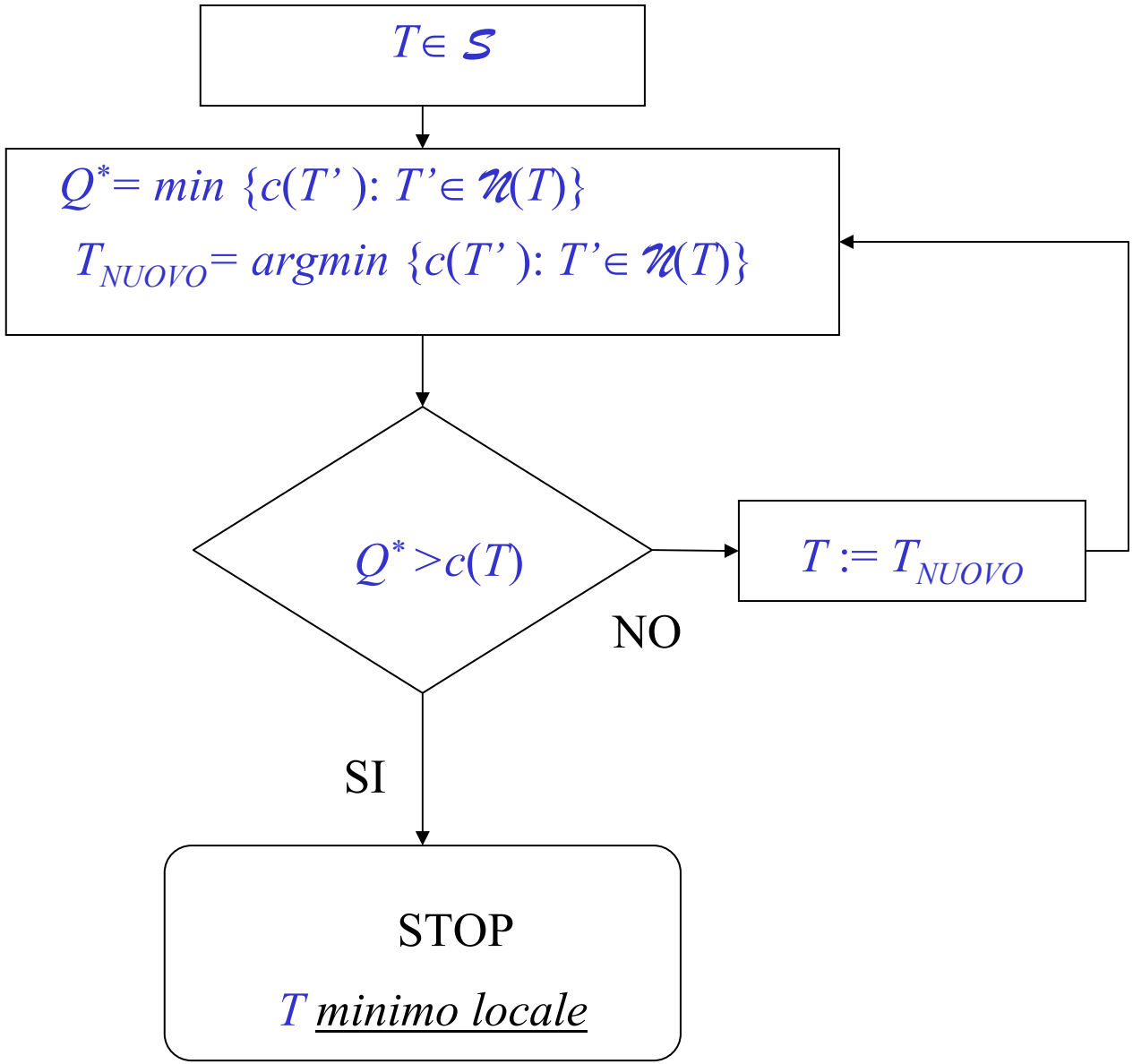
F^* minimo globale $\Leftrightarrow$ $c(F^*) \leq c(F)$ *per ogni* $F \in \mathcal{S}$

F^* minimo locale $\Leftrightarrow$ $c(F^*) \leq c(F)$ *per ogni* $F \in \mathcal{N}(F^*)\}$

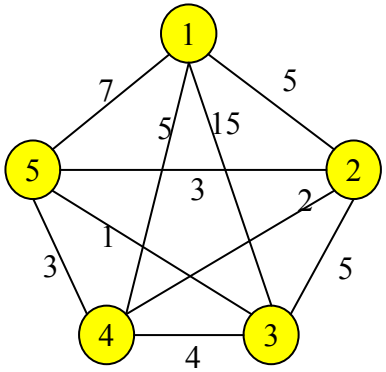


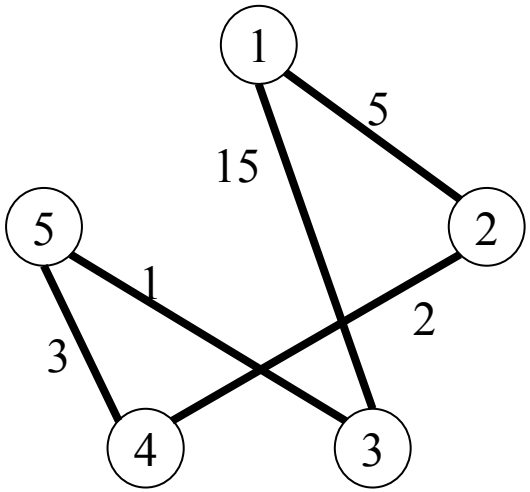
L'algoritmo di **ricerca locale** individua un **minimo locale**

Struttura Ricerca Locale

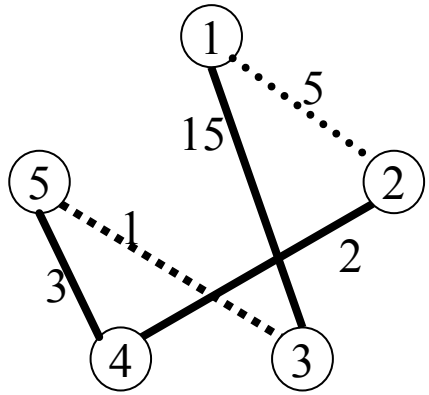


Intorno 2-scambio (1/2)

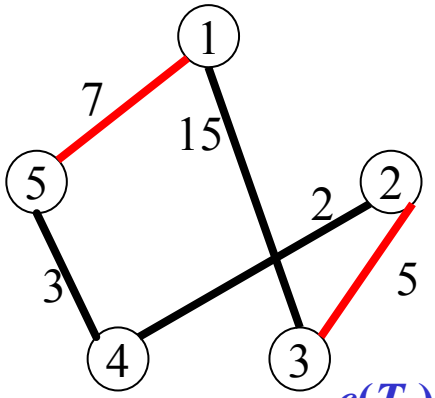




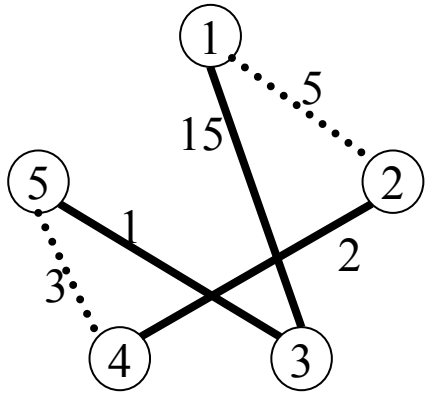
$c(T) = 26$



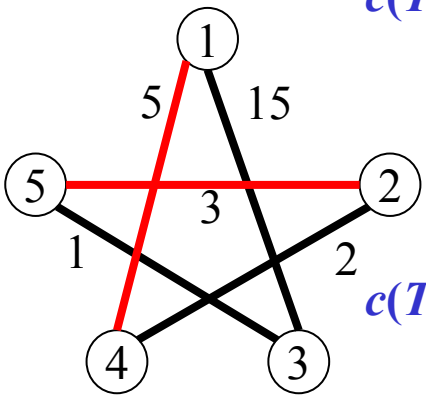
T_1



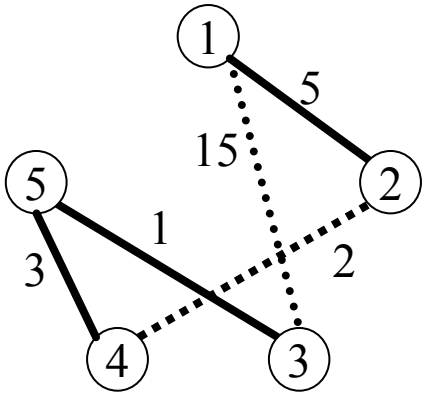
$c(T_1) = 32$



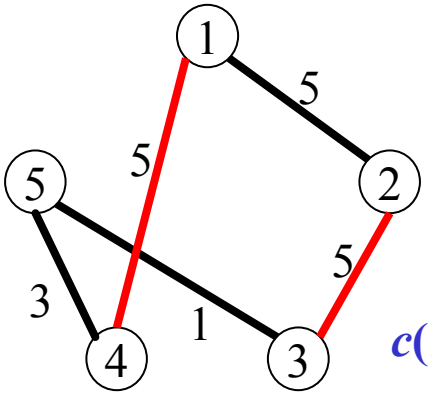
T_2



$c(T_2) = 26$

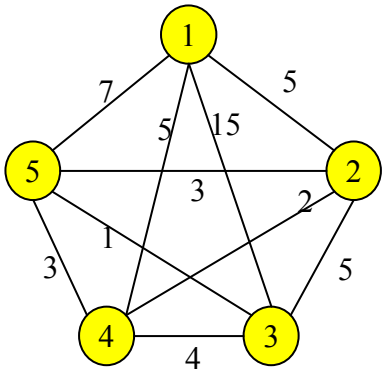


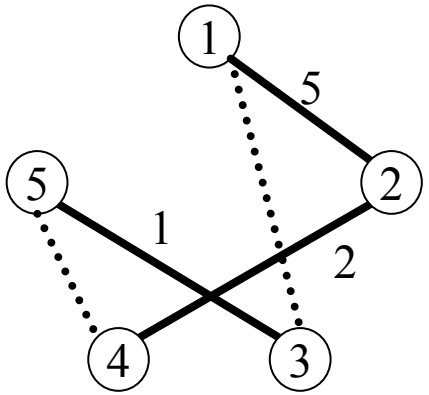
T_3



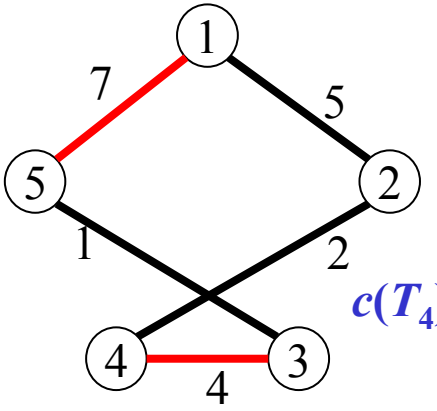
$c(T_3) = 19$

Intorno 2-scambio (2/2)

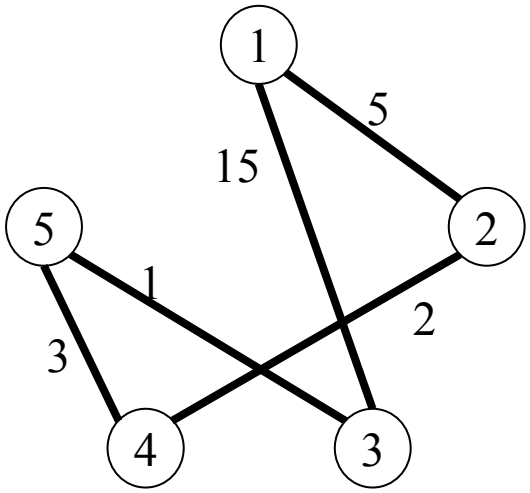




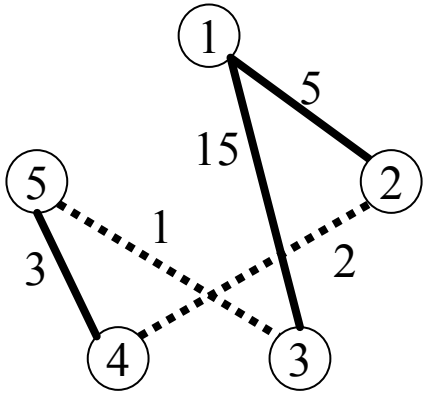
T_4



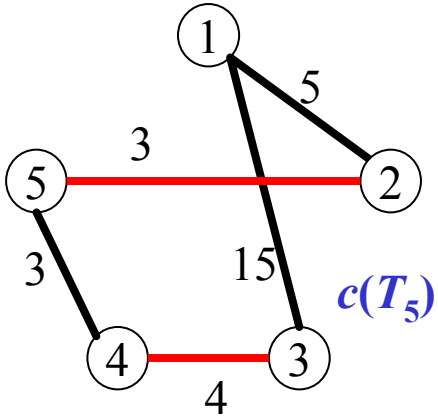
$c(T_4) = 19$



$c(T) = 26$



T_5



$c(T_5) = 30$

Non ci sono altri modi di fare
2-scambio: ho generato tutto $N(T)$

- La miglior soluzione in $N(T)$ è T_4 (o anche T_3, sono equivalenti)

Osservazioni sulla Ricerca Locale

- È una euristica **migliorativa** (ha bisogno di una soluzione di partenza)
- Per questo viene spesso eseguita dopo un Greedy
- Se l'intorno è costituito da pochi elementi, generarlo ed esaminarlo è **veloce**, ma le probabilità di ottenere miglioramenti sono **basse**.
- Al contrario, se l'intorno è costituito da molti elementi, le probabilità di ottenere miglioramenti sono più **alte**, ma generarlo ed esaminarlo potrebbe essere **costoso**
- Sarebbe conveniente riuscire ad usare intorni **vasti ma di rapida soluzione!**

Ricerca Locale con Intorni Esponenziali

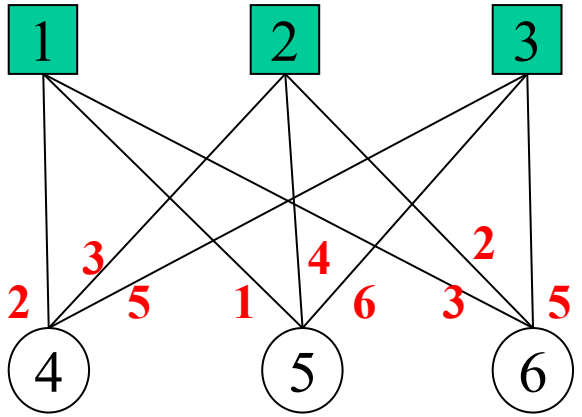
- A ogni iterazione dell'algoritmo di *LS* viene risolto un problema di ottimizzazione *locale*: $min\ \{c(T'): T' \in \mathcal{N}(T)\}$
- Il problema di ottimizzazione *locale* è analogo al problema originale, ma l'insieme delle soluzioni ammissibili (intorno) è più piccolo
- Negli esempi visti finora, la soluzione del problema locale può essere ottenuta efficientemente enumerando le soluzioni ammissibili (*poche*)
- Idea alternativa: definizione di un intorno "*grande*" ma tale che il problema di ottimizzazione *locale* sia risolvibile efficientemente
- Questo tipo di *LS* è chiamata *Ricerca Locale con intorni esponenziali*.
- Consideriamo il caso dell'intorno detto *ASSIGN* per il *TSP*.
- Per risolvere il problema di ottimizzazione associato, dobbiamo introdurre il *problema del matching* in grafi *bipartiti*.

Matching su grafi bipartiti

- Grafo bipartito completo: $H(Z,A)$:

$Z = U \cup W \quad A = \{(u,w): u \in U, w \in W\}$

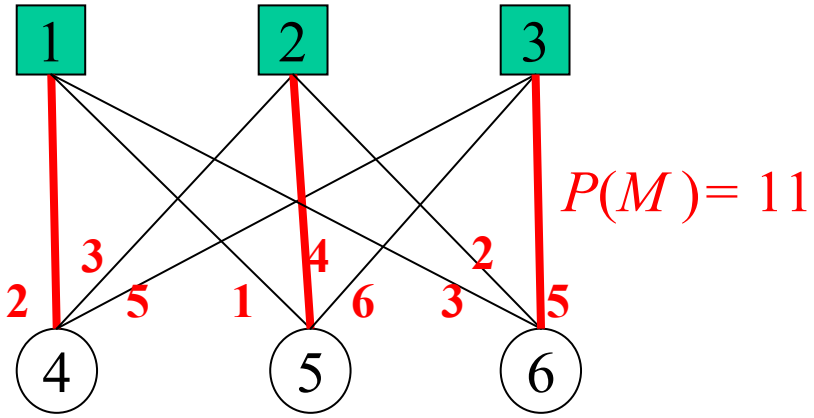
Pesi p_{uw} per ogni $uw \in A$



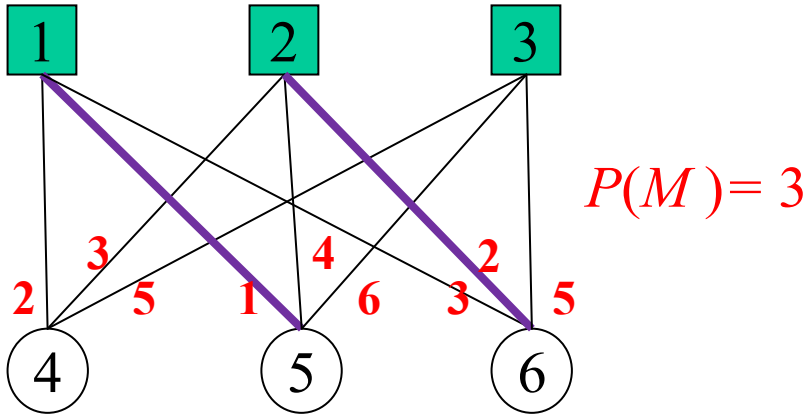
$H(Z,A)$

- Matching: sottoinsieme $M \subseteq A$ di archi *indipendenti* (niente nodi in comune)

$uw, ij \in M, uw \neq ij \Rightarrow u \neq i, u \neq j, w \neq i, w \neq j$



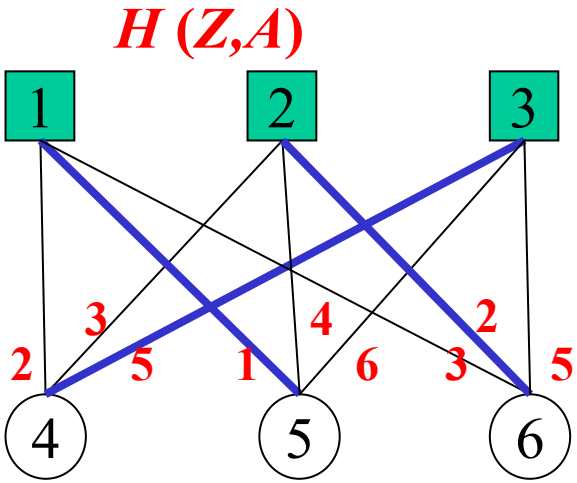
$P(M) = 11$



$P(M) = 3$

Peso Matching M: $p(M) = \sum_{e \in M} p(e)$

Matching perfetto di peso minimo



$P(M) = 8$

- $H(Z,A)$, $Z = U \cup W$, $|U| = |W|$
- Matching perfetto M di H:

$$|M| = |Z/2| = |U| = |W|$$

Problema Matching Perfetto: trova un matching perfetto di peso minimo

- Il problema del Matching Perfetto di peso (costo) minimo viene ridotto a problema di flusso a costo minimo (problema facile) su un grafo orientato capacitato H'
- H' *è* costruito a partire da H *orientando* gli archi da U a W

Trasformazione in Flusso a Costo Minimo

Costruzione $H' = (Z,A')$, con capacità c e costi p

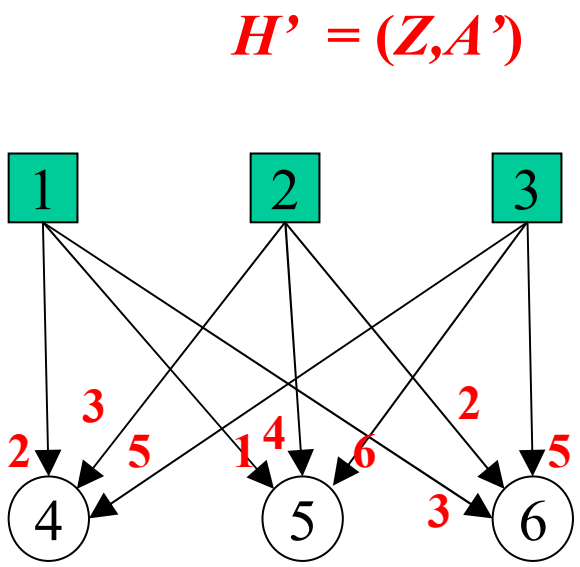
1.Orienta gli archi da U a W:

$A' = \{(u,w): u \in U, w \in W\}$

2. Associa capacità unitaria $c_a = 1$ a ogni $a \in A'$

3. Poni $d_u = -1$ *per ogni* $u \in U$

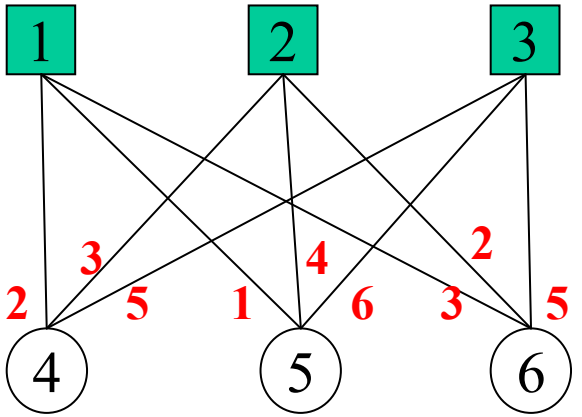
4. Poni $d_w = 1$ *per ogni* $w \in W$.



- Ogni *Matching Perfetto* corrisponde a un flusso intero (0-1) e viceversa.
- Il flusso a costo minimo (con costi p) corrisponde al *matching perfetto di peso minimo*
- Il problema di flusso a costo minimo può essere risolto efficientemente.

Matching e Assegnamento

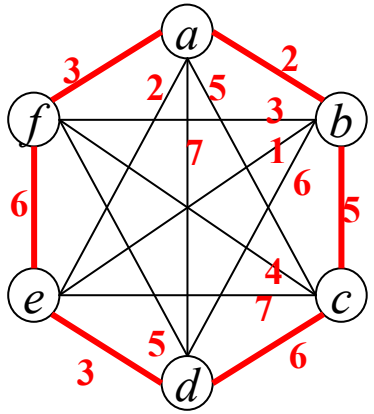
- Il problema del matching perfetto può essere interpretato come problema di *assegnamento*.
- Dati due insiemi di elementi U e W di uguale *cardinalità*
- Dato un costo p_{uw} di assegnare l'elemento $u \in U$ all'elemento $w \in W$.
- Assegnare ogni elemento di U ad un elemento di W in modo da minimizzare il costo complessivo di assegnamento.
- Equivalente al problema di matching perfetto nel grafo bipartito completo non orientato $H(U \cup W, A)$ con pesi p_{uw} per ogni $uw \in A$

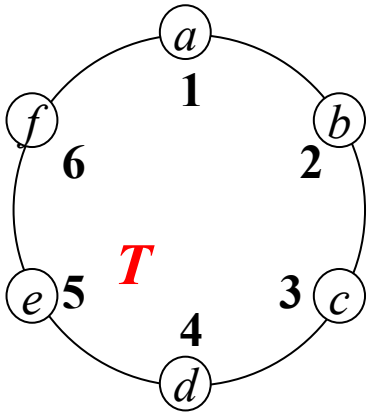


Cicli Hamiltoniani e Permutazioni

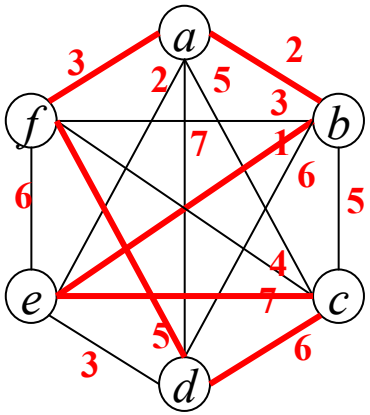
- Ogni *Ciclo Hamiltoniano* corrisponde a una permutazione (non unica) dei vertici, e ogni permutazione corrisponde a un ciclo Hamiltoniano (unico)

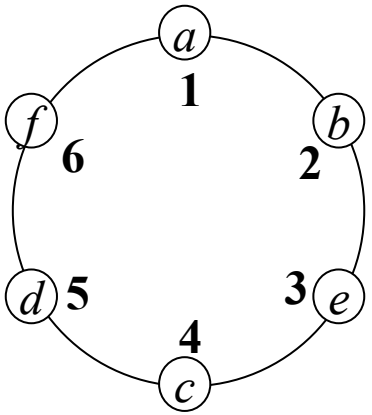
G(V,E)





(a, b, c, d, e, f)





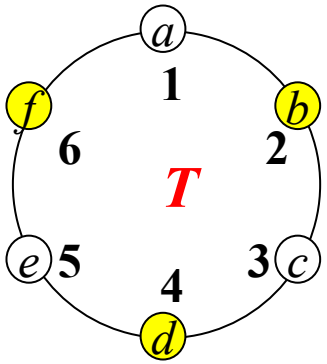
(a, b, e, c, d, f)

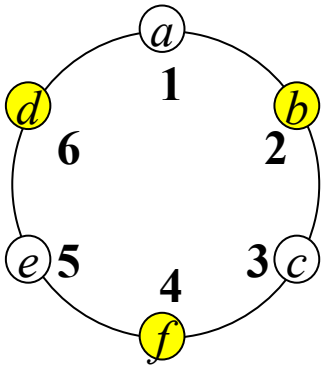
- Fissato il primo elemento nella permutazione (es. *a*), ogni altro elemento occupa univocamente una posizione pari oppure dispari

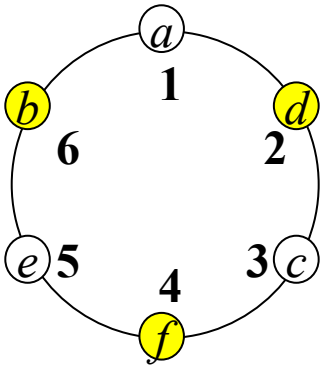
Intorno delle Permutazioni

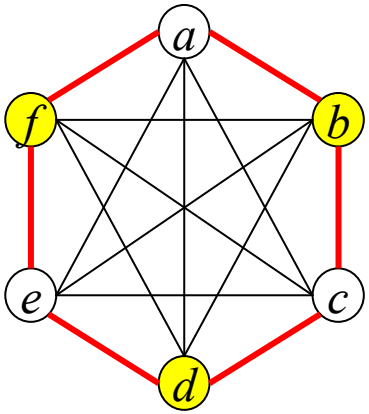
- Dato un ciclo hamiltoniamo T definiamo intorno *ASSIGN di T* *(indicato con* $\mathcal{N}_A(T)$*)* come:

$\mathcal{N}_A(T)$ = {*insieme dei cicli hamiltoniani ottenuti da T riposizionando in tutti i modi possibili i nodi in posizioni pari*}

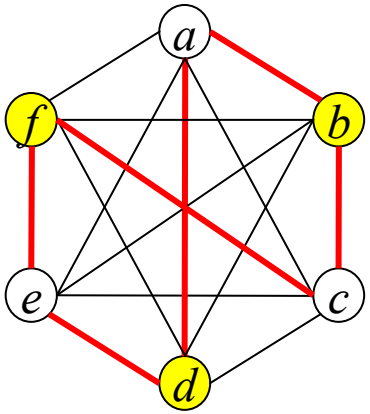




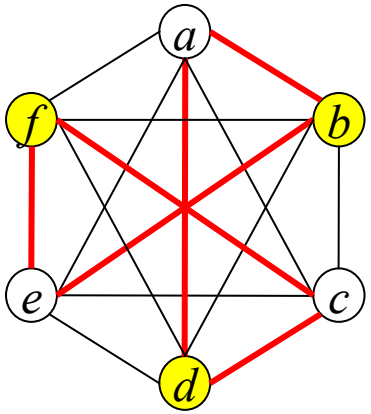




(*a*, ***b***, *c*, ***d***, *e*, ***f***)



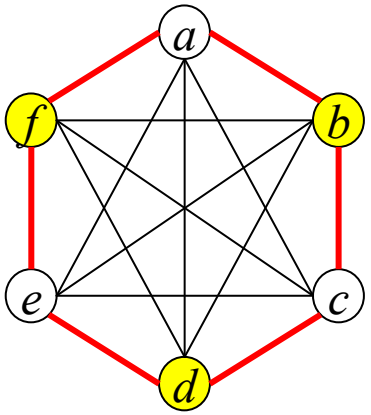
(*a*, ***b***, *c*, ***f***, e, ***d***)

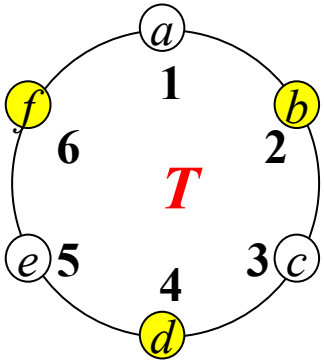


(*a*, ***d***, *c*, ***f***, *e*, ***b***)

Cardinalità dell'Intorno Permutazioni

$\mathcal{N}_A(T)$ = {*insieme dei cicli hamiltoniani ottenuti da T riposizionando in tutti i modi possibili i nodi in posizioni pari*}

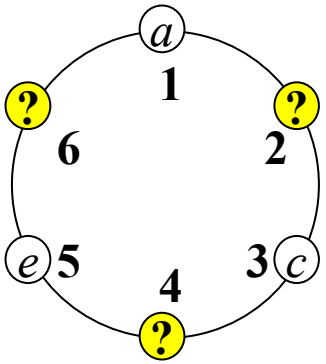




$T = \{v_1 , v_2 , v_3 , v_4 , \ldots, v_n\}$

$U = \{v_2 , v_4 , \ldots, v_{\lfloor n/2 \rfloor \cdot 2}\}$ **Nodi liberi**

$\{2, 4, \ldots, \lfloor n/2 \rfloor \cdot 2\}$ **Posizioni libere**

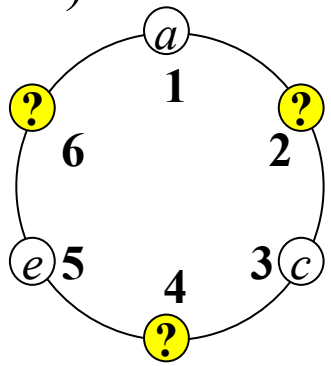


I modi diversi di assegnare i nodi liberi alle posizioni libere sono il numero di permutazioni di $\lfloor n/2 \rfloor$ elementi, che sono $\lfloor n/2 \rfloor !$

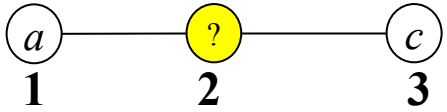
$|\mathcal{N}_A(T)|$ cresce esponenzialmente con n

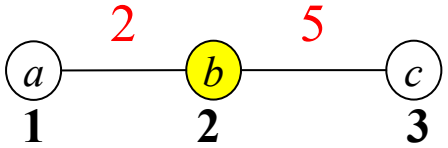
Costo di inserimento di un nodo

- Data una posizione pari, i due nodi adiacenti (posizioni dispari) sono noti .
- Esempio: la posizione 2 è adiacente al nodo *a* e al nodo *c*
- Indichiamo con p_{vi} il costo di inserimento di un nodo "libero" *v* in una posizione "libera" *i*.
- Quanto costa inserire il nodo *b* in posizione **2**?



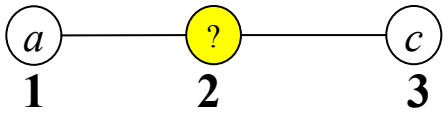
$$p_{b2} = c_{ab} + c_{bc} = 2 + 5 = 7$$

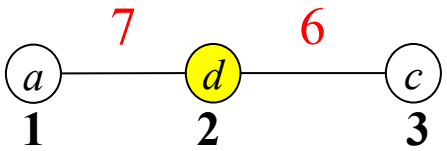




- Quanto costa inserire il nodo *d* in posizione **2**?

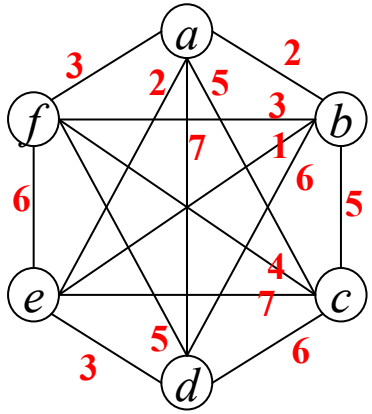
$$p_{d2} = c_{ad} + c_{dc} = 7 + 6 = 13$$

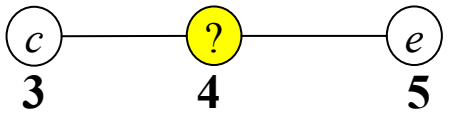


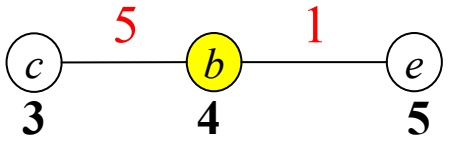


- Quanto costa inserire il nodo *b* in posizione **4**?

$$p_{b4} = c_{cb} + c_{be} = 5 + 1 = 6$$



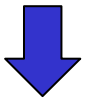




Ottimizzazione nell'Intorno

- Sia U l'insieme dei nodi liberi (quelli in posizione pari) di T
- Sia W l'insieme delle posizioni pari di T.
- Per ogni $v \in U$ e ogni $i \in W$ calcoliamo il costo p_{vi} di assegnare il nodo v alla posizione i : sia u il nodo in posizione i-1 e z il nodo in posizione i+1 (mod $|V|$), allora $p_{vi} = c_{uv} + c_{vz}$.
- Trovare la soluzione ottima in $\mathcal{N}_A(T)$ equivale a risolvere il seguente

- Problema: assegnare ogni nodo $v \in U$ ad esattamente una posizione $i \in W$ in modo da minimizzare il costo complessivo dell'assegnamento

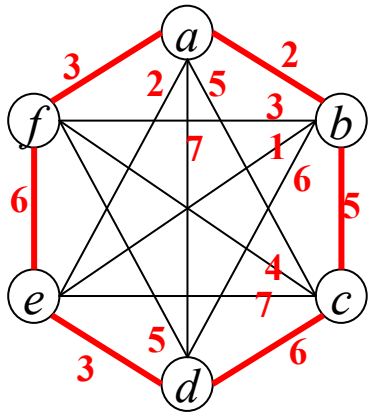


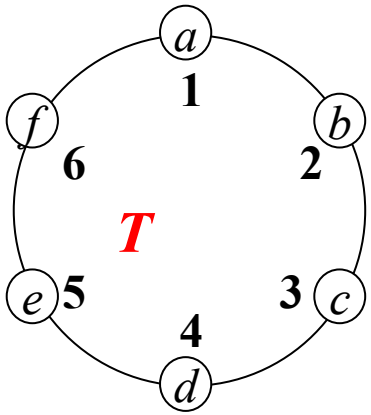
- Equivalente al problema di *matching perfetto* sul grafo bipartito completo $H(U \cup W, A)$ con pesi p_{uw} per ogni $uw \in A$

Esempio

- Consideriamo il ciclo Hamiltoniano {a,b,c,d,e,f} (sol. ammissibile)

G(V,E)



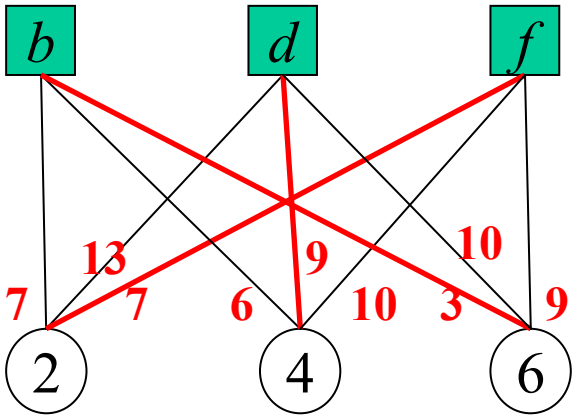


Nodi liberi $U = \{b, d, f\}$

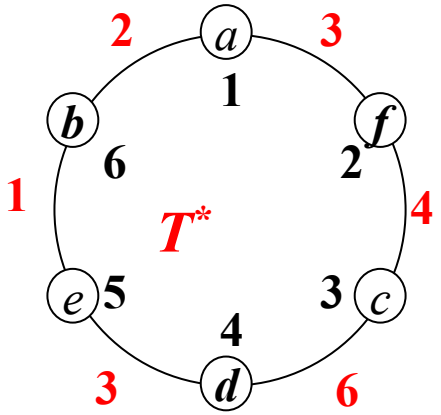
Posizioni $W = \{2, 4, 6\}$

Costo $c(T) = 25$

La miglior soluzione dell'intorno Permutazione è data risolvendo il seguente problema di matching perfetto, cioè il flusso a costo minimo



Questa assegnazione corrisponde alla sol.:



$M^* = \{b6, d4, f2\}$ $p(M^*) = 19$

Costo $c(T^*) = 19$

Qualità di un algoritmo euristico

- Gli algoritmi euristici cercano *buone* soluzioni ammissibili per un problema di ottimizzazione $(\mathrm{Q}): \quad \min \{c(F): F \in \mathcal{S}\} = v(Q)$
- Come definire la qualità della soluzione prodotta da un algoritmo?
- In alcuni casi non si può garantire nulla (la soluzione potrebbe essere molto scadente), in altri ci possono dare delle *garanzie* a priori (es. l'ottimo o entro una certa distanza dall'ottimo)
- Sia $\mathcal{A}$ un algoritmo per (Q) che produce la soluzione $F \in \mathcal{S}$
- Diciamo che $\mathcal{A}$ è un algoritmo che garantisce un'approssimazione α (l'algoritmo $\mathcal{A}$ è α *-approssimato*) se

$$c(F) \leq \alpha\, v(\mathrm{Q})$$

- Illustreremo un algoritmo **2-approssimato** per il *TSP metrico* (cioè la soluzione fornita vale non più del doppio della soluzione ottima)

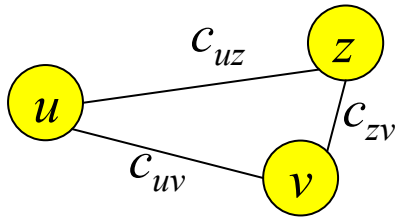
TSP metrico

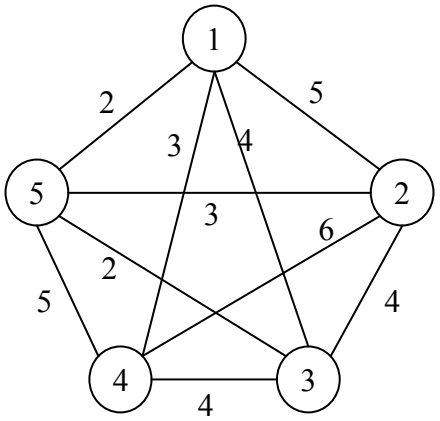
Dato un grafo $G(V,E)$ non orientato, completo con costi c associati agli archi, se

- $c_{uv} \geq 0$ per ogni arco $uv \in E$ e $c_{uv} = 0$ se [se solo se] $u = v$
- c è tale che *per ogni tripla di nodi distinti* $u, v, z,$ si ha che

$$c_{uv} \leq c_{uz} + c_{zv} \quad (\textit{disuguaglianza triangolare})$$

allora c induce una semimetrica [una metrica]





- Trovare il ciclo Hamiltoniano di costo minimo in $G(V,E)$ con vettore dei costi c (semi)metrica

Metriche indotte da norme

- Una classe molto importante di metriche è quella delle **metriche indotte dalle varie norme** $\|\bullet\|_p$:

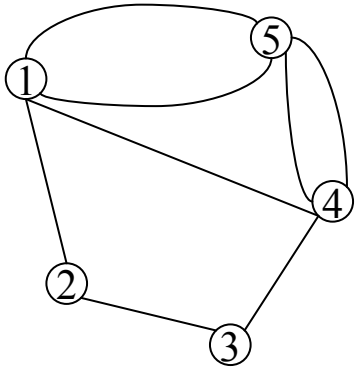
$$d_{\|\bullet\|_p}(i,j) = \| i - j \|_p = \left(\sum_{k=1}^{m} | i_k - j_k |^p \right)^{1/p}$$

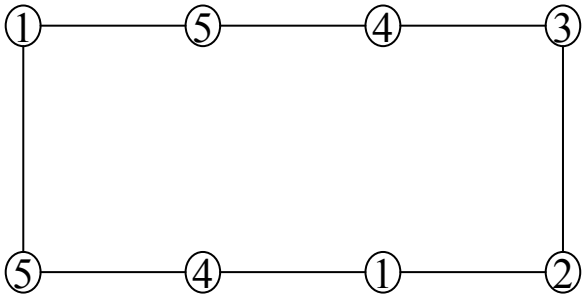
- Se p=1 distanza di Manahattan $d_{\|\bullet\|_1}(i,j) = \sum_{k=1}^{m} | i_k - j_k |$
- Se p=2 distanza Euclidea $d_{\|\bullet\|_2}(i,j) = \left(\sum_{k=1}^{m} | i_k - j_k |^2 \right)^{1/2}$
- Se $p= \infty$ distanza di Chebyshev $d_{\|\bullet\|_\infty}(i,j) = \max_k \{ | i_k - j_k | \}$

Cicli Euleriani

- Per descrivere l'euristica 2-approssimata di Christofides per il TSP metrico dobbiamo introdurre il concetto di ciclo eureliano

Dato un (multi)grafo non orientato, si definisce *Ciclo Euleriano* un walk chiuso che passa esattamente una volta per ogni arco.





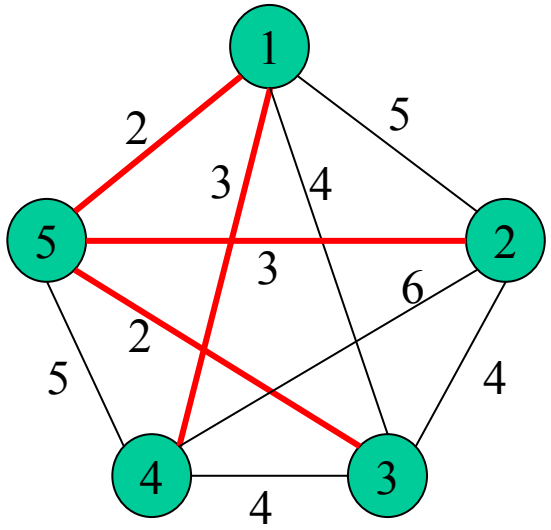
Teorema di Eulero. Un (multi)grafo ammette un ciclo euleriano se e solo se è connesso e ogni nodo ha grado pari.

Un (multi) grafo connesso e con ogni nodo di grado pari è detto *grafo Euleriano*

Albero Ricoprente di peso minimo

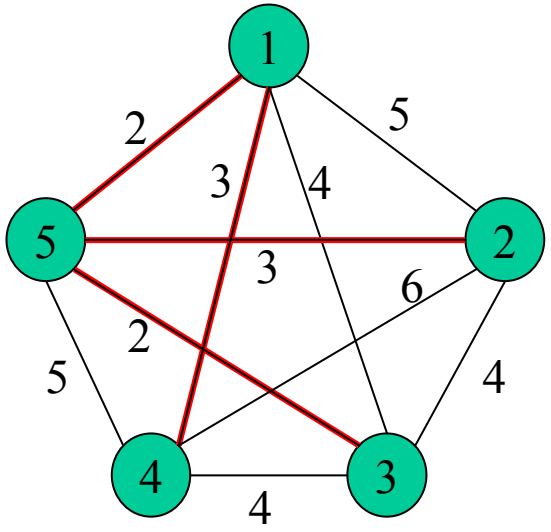
- Problema dell'albero ricoprente di peso minimo (Minimum Spanning Tree, **MST**): dobbiamo raggiungere un certo insieme di clienti effettuando i collegamenti meno costosi

- Definito su un grafo $G=(V,E)$ non orientato connesso con costi c_i associati agli archi
- È un problema di OC, l'insieme base è *E*
- Una soluzione ammissibile S è un **albero ricoprente** (albero che raggiunge tutti i nodi)
- La funzione di costo è $\sum_{i \in T_k} c_i$ (nell'esempio 10)



Si risolve all'ottimo col Greedy

- Particolarizzazione del Greedy: algoritmo di Kruskal
- Ordino gli archi in ordine di costo crescente
- Ad ogni iterazione si sceglie un arco, se **non genera cicli** con quelli già selezionati viene aggiunto alla soluzione
- Termina quando ho collegato tutti i nodi (e quindi ho $|V| - 1$ archi)
- Scelgo nell'ordine (1,5) (3,5) (1,4) (2,5) e ho 4 archi: stop e $c(S) = 10$



- il MST ha una particolare struttura matematica detta di matroide pesato. In tali casi il metodo greedy **garantisce l'ottimo**
- Esistono altri problemi con questa simpatica caratteristica

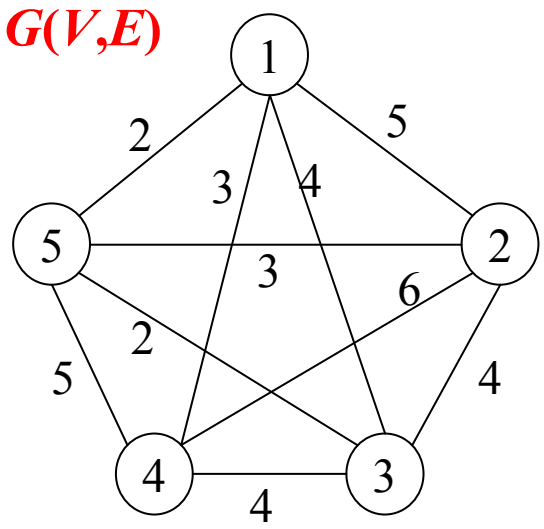
Algoritmo di Christofides

- Grafo $G(V,E)$ non orientato, completo, con costi c dati da una *semimetrica*
- L'algoritmo di Christofides trova un ciclo hamiltoniano di costo al più doppio rispetto al ciclo di costo minimo

L'algoritmo di Christofides

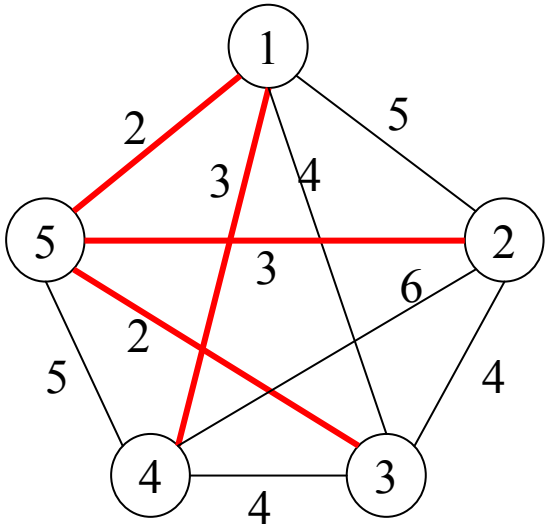
1. Trova il minimo albero ricoprente T di G (facile!)
2. Raddoppia ogni arco di T ottenenendo un grafo euleriano
3. Costruisci un ciclo euleriano $\mathcal{E}$ di questo grafo (facile!)
4. Scegli un nodo iniziale u in $\mathcal{E}$ e visita $\mathcal{E}$ a partire da u
5. Costruisci una permutazione π dei nodi V *sequenziando i nodi* nell'ordine della loro prima apparizione in $\mathcal{E}$ nella visita
6. Restituisci il ciclo hamiltoniano H associato a π

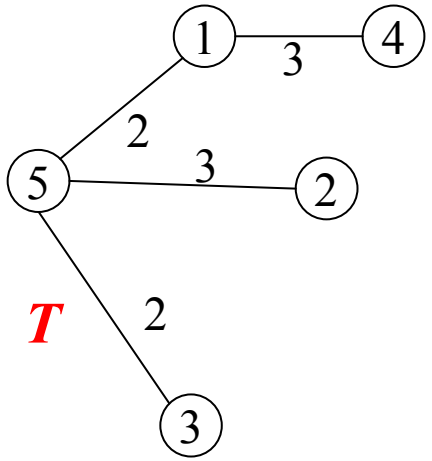
Esempio 1/3



1. Trova il minimo albero ricoprente *T* di *G*.

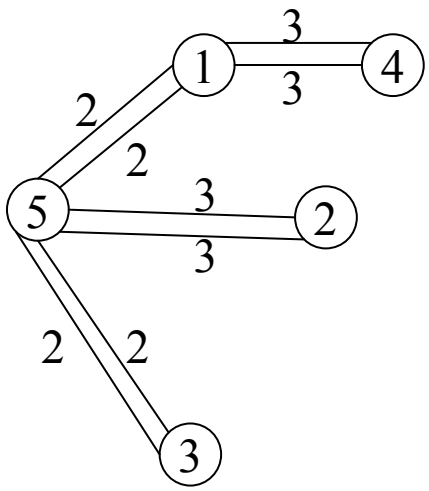
Per questo problema l'algoritmo greedy garantisce l'ottimo: basta scegliere gli archi in ordine di costo crescente ma saltando quelli che chiudono cicli





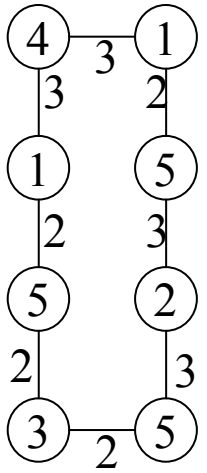
Esempio 2/3

2. Raddoppia ogni arco di T ottenendo un grafo euleriano

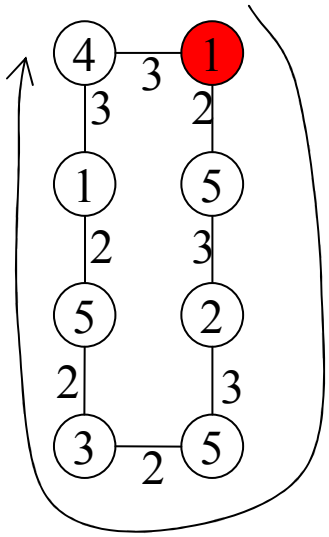


3. Costruisci un ciclo euleriano $\mathcal{E}$ del grafo euleriano

Basta percorrere uno dopo l'altro tutti i cicli ottenuti dai rami dell'albero



$\mathcal{E}$



4. Scegli nodo iniziale u in $\mathcal{E}$ e visita $\mathcal{E}$ a partire da u

5. Costruisci una permutazione π dei nodi V sequenziando i nodi nell'ordine della loro prima apparizione in $\mathcal{E}$ nella visita

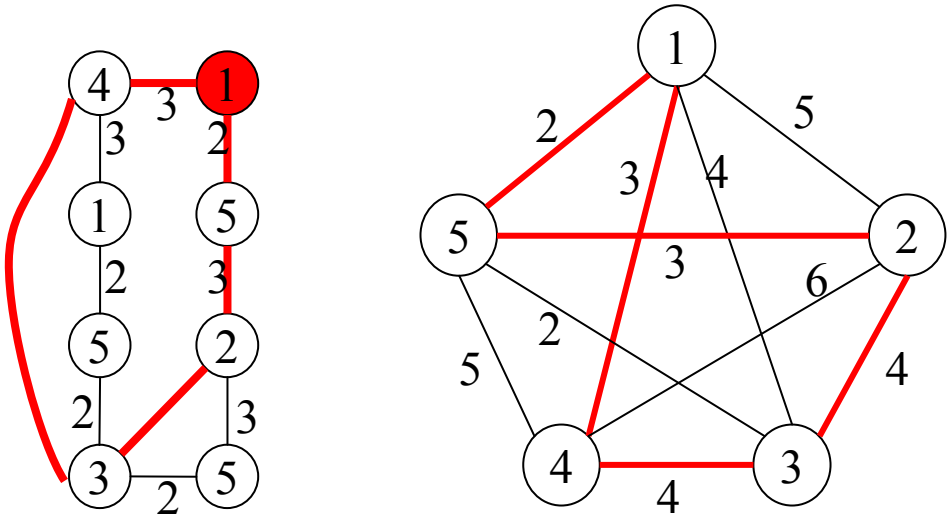
Sequenza di visita di $\mathcal{E}$: (1,5, 2, 5, 3, 5, 1, 4)

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

Esempio 3/3

6. Ciclo hamiltoniano H associato a $\pi = (1, 5, 2, 3, 4)$

H è ottenuto da $\mathcal{E}$ sostituendo cammini con archi

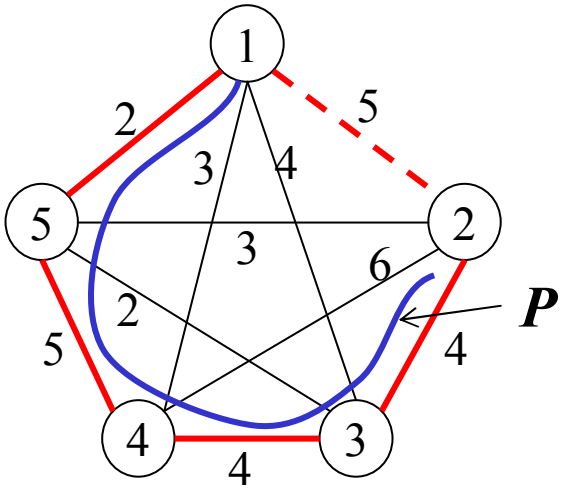


Poiché c **soddisfa la disuguaglianza triangolare**, saltando dei nodi seguo delle scorciatoie. Allora

$$c(H) \leq c(\mathcal{E})$$

Valore della soluzione trovata

Th. 1 Sia H^* il ciclo hamiltoniano di costo minimo, e sia T^* l'albero ricoprente di G di costo minimo. Allora $c(H^*) \geq c(T^*)$



Sia P il cammino semplice ottenuto da H^* *rimuovendo un arco*. Allora P è un albero

$c(H^*) \geq c(P)$ (costi non negativi)

P contiene tutti i nodi di $G \rightarrow P$ è un albero ricoprente

P albero ricoprente, T^* albero ricoprente a costo minimo $\rightarrow c(P) \geq c(T^*)$

$c(H^*) \geq c(P) \quad \rightarrow \quad c(H^*) \geq c(T^*)$

■

Valore della soluzione trovata

Th. 2 Sia H^* il ciclo hamiltoniano di costo minimo, e sia H il ciclo ritornato dall'algoritmo di Christofides. Allora $c(H) \leq 2c(H^*)$

$\mathcal{E}$ ottenuto dall'albero ricoprente di costo minimo T^* raddoppiando gli archi ⇒ $c(\mathcal{E}) = 2c(T^*)$

H ottenuto dal ciclo euleriano $\mathcal{E}$ eventualmente saltando nodi (quindi facendo scorciatoie) ⇒ $c(H) \leq c(\mathcal{E}) = 2c(T^*)$

Dal Teorema precedente $c(T^*) \leq c(H^*)$ ⇒ $c(H) \leq 2c(T^*) \leq 2c(H^*)$

■