Il Problema del Commesso Viaggiatore

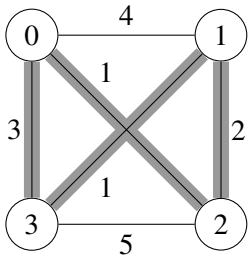
Il *problema del commesso viaggiatore* è uno tra i più semplici fra i problemi di routing e di scheduling e uno dei casi di studio tipici dell'informatica teorica e della teoria della complessità. Esso viene spesso indicato con il suo nome inglese, Travelling Salesman Problem, da cui la sigla *TSP*. Il nome nasce dalla sua più tipica rappresentazione: data una rete di città, connesse tramite delle strade, trovare il percorso di minore distanza che un commesso viaggiatore deve seguire per visitare tutte le città una ed una sola volta.

Questo problema ha diverse applicazioni anche nella sua formulazione più semplice, come la pianificazione, la logistica e la produzione di microchip. Leggermente modificato, appare come un sotto-problema in molte aree, come il sequenziamento del DNA. In queste applicazioni, le "città" rappresentano, ad esempio, clienti, punti di saldatura o frammenti di DNA, e la "distanza" rappresenta tempi o costi di viaggio o una misura di somiglianza tra frammenti di DNA. Il TSP compare anche in astronomia, per esempio quando gli astronomi vogliono minimizzare il tempo trascorso per spostare il telescopio tra le diverse stelle da osservare. In molte applicazioni, possono essere imposti limiti aggiuntivi come risorse limitate o finestre temporali per il completamento del percorso.

TSP come problema su grafi

Il problema del commesso viaggiatore si può rappresentare con un grafo non orientato, pesato e completo $G = (V,E)$ dove i vertici sono le città ed il peso dell'arco $\{u,v\}$ è uguale alla distanza da u a v. Risolvere il TSP significa quindi trovare un circuito Hamiltoniano, ossia un ciclo che visita tutti i vertici esattamente una volta, di costo minimo.

Poiché l'input del problema è un *grafo completo*, ossia dove tra due vertici c'è sempre un arco che li collega, il modo più efficiente per rappresentare l'input del problema è di usare una matrice delle adiacenze $n \times n$ con i pesi degli archi $w[i,j]$. La Figura 1 mostra un'istanza di TSP con quattro città e la corrispondente matrice 4×4 con i pesi. Il circuito di peso minimo è $0,2,1,3$, con peso totale 7, è evidenziato in grigio.



$$w = \begin{bmatrix} 0 & 4 & 1 & 3 \\ 4 & 0 & 2 & 1 \\ 1 & 2 & 0 & 5 \\ 3 & 1 & 5 & 0 \end{bmatrix}$$

Figura 1: Un'istanza di TSP con quattro città.

NP-completezza di TSP

Il fatto che TSP si possa definire come una variante del problema del circuito Hamiltoniano (ossia di un problema NP-completo) su un grafo pesato ci suggerisce che anche TSP è un problema difficile da risolvere. Per poter stabilire con precisione la sua complessità dobbiamo prima "trasformarlo" in un problema di decisione, aggiungendo un limite k per il peso del ciclo all'input del problema.

Definizione 1 — TSP decisionale. Dato un grafo non orientato, completo e pesato $G = (V, E)$ e un valore $k > 0$, esiste un ciclo in G che attraversa tutti i vertici una sola volta di peso inferiore a k?

Il teorema seguente mostra che TSP è un problema NP-completo anche nella sua variante decisionale.

Teorema 1 Il problema del commesso viaggiatore è NP-completo.

Dimostrazione. Dimostriamo per prima cosa che TSP appartiene a NP. Data un'istanza del problema, usiamo come certificato la sequenza degli n vertici del ciclo di peso minimo. L'algoritmo di verifica del certificato deve controllare che la sequenza contenga ogni vertice di G esattamente una volta, sommare il peso degli archi, e controllare che il peso totale del ciclo sia inferiore a k. Questo procedimento si può svolgere sicuramente in tempo polinomiale, più precisamente in tempo $O(n)$.

Per dimostrare che TSP è NP-hard, mostriamo come ridurre il problema del Circuito Hamiltoniano a TSP. Prendiamo un grafo non orientato $G = (V, E)$ e costruiamo un'istanza di TSP che ci permetta di risolvere il problema del circuito Hamiltoniano su G. Costruiamo un grafo non orientato e completo $G' = (V, E')$ con gli stessi vertici di G e $E' = \{\{u, v\} \mid u, v \in V\}$. Il peso degli archi di G' viene assegnato come segue:

$$w(u,v) = \begin{cases} 0 & \text{se } \{u,v\} \in E \\ 1 & \text{se } \{u,v\} \notin E \end{cases}$$

È piuttosto evidente che possiamo costruire il grafo G' in tempo polinomiale rispetto al numero di vertici del grafo di partenza G.

Mostriamo ora che G contiene un circuito Hamiltoniano se e solo se G' ha un ciclo di peso minore o uguale a 0. Supponiamo che G contenga un circuito Hamiltoniano $h = v_1, \ldots, v_n$. Ogni arco che compone h è presente in E e quindi ha peso 0 in G'. Quindi h è un ciclo di G' di peso uguale a 0. Viceversa, supponiamo che G' contenga un ciclo semplice t che attraversa tutti i vertici e di peso minore o uguale a 0. Poiché i pesi degli archi in G' sono solo 0 oppure 1, tutti gli archi che compongono t devono avere costo 0. Quindi tutti gli archi del ciclo sono presenti anche in E e t è un circuito Hamiltoniano per G. ■

Stabilire che TSP è un problema NP-completo ci indica che molto probabilmente non esiste una soluzione polinomiale al problema. In questo laboratorio studieremo tre modi possibili per risolvere TSP:

- una soluzione esatta di complessità più che polinomiale per risolvere istanze piccole (circa 10 città);
- una famiglia di euristiche dette "costruttive" che risolvono in problema con un approccio greedy, e consentono di arrivare a soluzioni approssimate con vari gradi di accuratezza;
- la soluzione 2-approssimata basata sulla costruzione di un albero di copertura per il grafo delle città, sotto l'ipotesi che le distanze rispettino la *disuguaglianza triangolare*.

Algoritmo esatto di Held e Karp

L'algoritmo di Held e Karp, noto anche come algoritmo di Bellman, Held e Karp, è un algoritmo di programmazione dinamica per risolvere TSP che è stato proposto nel 1962 sia da Bellman che in maniera indipendente da Held e Karp.

La programmazione dinamica è una tecnica che permette di risolvere problemi di ottimizzazione combinando le soluzioni di sottoproblemi più semplici. A differenza di altri approcci ricorsivi, dove i sottoproblemi sono disgiunti tra loro e vengono risolti in modo indipendente, la programmazione dinamica si applica quando i sottoproblemi si sovrappongono, ossia quando i sottoproblemi condividono sotto-sotto-problemi. Un algoritmo di programmazione dinamica risolve ognuno dei sottoproblemi una volta sola e salva il risultato in una "tabella" per poterlo riutilizzare in seguito senza doverlo risolvere di nuovo.

Vediamo quindi come possiamo utilizzare la programmazione dinamica per risolvere il problema TSP. L'algoritmo di Held e Karp si basa sulla seguente proprietà dei cammini minimi:

Proposizione 1 Ogni sottocammino di un cammino minimo è a sua volta un cammino minimo.

La proprietà di permette di formulare il problema TSP in modo ricorsivo. Supponiamo che le n città siano numerate da 0 a $n-1$, e assumiamo che il ciclo parta dalla città 0. Dato un sottoinsieme di vertici $S \subseteq V$ e un vertice $v \in S$, definiamo due vettori d e π come segue:

- $d[v,S]$ è il peso del cammino minimo che parte da 0 e termina in v, visitando tutti i nodi in S;
- $\pi[v,S]$ è il predecessore di v nel cammino minimo definito come sopra.

Alla fine dell'esecuzione dell'algoritmo $d[0,V]$ conterrà il peso del cammino minimo che parte da 0 e termina in 0 visitando tutti i nodi del grafo, ossia la soluzione di TSP che stiamo cercando, mentre l'informazione in π ci consentirà di ricostruire il ciclo di peso minimo.

Mostriamo come possiamo calcolare i valori contenuti d e π sull'istanza di TSP mostrata in Figura 1. Poiché sappiamo che il nodo 0 è quello da cui parte e termina il ciclo finale possiamo escluderlo dal calcolo delle soluzioni parziali: d'ora in poi quindi considereremo solamente coppie v,S dove $v \neq 0$ e $S \subseteq \{1,\ldots,n-1\}$, con l'eccezione del sottoproblema $0,V$ che viene calcolato per ultimo per ottenere la soluzione completa.

I sottoproblemi di dimensione minima sono quelli in cui l'insieme S ha dimensione 1: $S = \{v\}$ (e $v \neq 0$). In questo caso abbiamo che $d[v,\{v\}]$ è il peso del cammino minimo che parte da 0 e raggiunge v passando solo per v. L'unico cammino possibile è quello composto dal solo arco $\{0,v\}$, e quindi abbiamo che $d[v,\{v\}] = w(0,v)$, mentre $\pi[v,\{v\}] = 0$. Nell'esempio considerato:

v,S	$d[v,S]$	$\pi[v,S]$
$1,\{1\}$	4	0
$2,\{2\}$	1	0
$3,\{3\}$	3	0

Possiamo ora utilizzare i valori calcolati per ottenere i risultati per tutte le coppie v, S dove S ha dimensione 2. Poiché $v \in S$ abbiamo che le coppie valide sono del tipo $v, \{u, v\}$ e l'unico cammino che parte da 0 e raggiunge v passando per tutti nodi in S è $0, u, v$. Il peso di tale cammino si può ottenere sommando il peso dell'ultimo arco $\{u, v\}$ al peso del cammino minimo che parte da 0 e raggiunge u passando solo per u (calcolato al passo precedente). Quindi si ha che $d[v, \{u, v\}] = w(u, v) + d[u, \{u\}]$ e $\pi[v, \{u, v\}] = u$. Nel nostro esempio abbiamo che i valori possibili sono i seguenti:

v, S	$d[v, S]$	$\pi[v, S]$
$1, \{1, 2\}$	$2 + 1 = 3$	2
$1, \{1, 3\}$	$1 + 3 = 4$	3
$2, \{1, 2\}$	$2 + 4 = 6$	1
$2, \{2, 3\}$	$5 + 3 = 8$	3
$3, \{1, 3\}$	$1 + 4 = 5$	1
$3, \{2, 3\}$	$5 + 1 = 6$	2

Procediamo ora con gli insiemi di dimensione 3. In questo caso abbiamo che i cammini possibili sono più di uno. Per esempio, nel caso della coppia $1, \{1, 2, 3\}$, i cammini che partono da 0 e raggiungono 1 percorrendo tutti e tre i vertici sono $0, 2, 3, 1$ e $0, 3, 2, 1$. Il peso del primo cammino si ottiene sommando il peso dell'ultimo arco $\{3, 1\}$ a $d[3, \{2, 3\}]$, ottenendo il valore $1 + 6 = 7$. Il peso del secondo cammino si ottiene sommando il peso dell'ultimo arco $\{2, 1\}$ a $d[2, \{2, 3\}]$, ottenendo il valore $2 + 8 = 10$. Il valore corretto per $d[1, \{1, 2, 3\}]$ è il minimo tra i due, in questo caso 7. $\pi[1, \{1, 2, 3\}]$ viene valorizzato con 3 perché il cammino minimo è $0, 2, 3, 1$. Questa osservazione ci porta alla definizione ricorsiva generale di $d[v, S]$ e $\pi[v, S]$:

$$d[v, S] = \min_{u \in S \setminus \{v\}} (d[u, S \setminus \{v\}] + w(u, v))$$
$$\pi[v, S] = \operatorname*{argmin}_{u \in S \setminus \{v\}} (d[u, S \setminus \{v\}] + w(u, v))$$

La sua applicazione sull'esempio ci permette di ottenere i seguenti valori:

v, S	$d[v, S]$	$\pi[v, S]$
$1, \{1, 2, 3\}$	7	3
$2, \{1, 2, 3\}$	6	1
$3, \{1, 2, 3\}$	4	1

L'ultimo passo dell'algoritmo calcola i valori di $d[0, \{0, 1, 2, 3\}]$ e $\pi[0, \{0, 1, 2, 3\}]$ usando i valori della tabella immediatamente precedente. Quindi il risultato finale è $d[0, \{0, 1, 2, 3\}] = 7$ e $\pi[0, \{0, 1, 2, 3\}] = 2$, oppure $\pi[0, \{0, 1, 2, 3\}] = 3$ perché entrambe le soluzioni hanno costo minimo 7. Questo è coerente con l'esempio, poiché il ciclo di costo minimo può avere come ultimo vertice 3 oppure 2, a seconda della direzione in cui viene visitato.

Implementare Held-Karp con una visita in profondità

Nell'esempio precedente abbiamo mostrato l'esecuzione dell'algoritmo di Held e Karp partendo "dal basso", ossia dai sottoproblemi più semplici e risalendo verso quelli più complessi fino ad arrivare alla soluzione finale. Nel caso si debba implementare l'algoritmo in un linguaggio di programmazione concreto conviene invece procedere partendo "dall'alto", ossia effettuando una visita in profondità a partire dal nodo iniziale 0. L'Algoritmo 1 mostra lo pseudocodice della variante ricorsiva di Held-Karp. La procedura HK-VISIT prende come argomenti un insieme $S \subseteq V$ ed un vertice $v \in S$ e ritorna il peso del cammino minimo da 0 a v che visita tutti i nodi in S. La procedura assume l'esistenza delle variabili globali d e π che contengono i valori già calcolati.

Algoritmo 1 L'algoritmo di Held e Karp per il problema del commesso viaggiatore

Input: $G = (V, E)$ grafo non orientato completo e pesato, $S \subseteq V$ e $v \in S$
Output: Peso del cammino minimo da 0 a v che visita tutti i vertici in S

```
 1: function HK-VISIT(v, S)
 2:     if S = {v} then                    // Caso base: la soluzione è il peso dell'arco {v, 0}
 3:         return w[v, 0]
 4:     else if d[v, S] ≠ NULL then        // Distanza già calcolata, ritorna il valore memorizzato
 5:         return d[v, S]
 6:     else                               // Caso ricorsivo: trova il minimo tra tutti i sottocammini
 7:         mindist = ∞
 8:         minprec = NULL
 9:         for all u ∈ S \ {v} do
10:             dist = HK-VISIT(u, S \ {v})
11:             if dist + w[u, v] < mindist then
12:                 mindist = dist + w[u, v]
13:                 minprec = u
14:             end if
15:         end for
16:         d[v, S] = mindist
17:         π[v, S] = minprec
18:         return mindist
19:     end if
20: end function
```

Come prima cosa controlla se la distanza da calcolare è già contenuta in d. In caso positivo termina immediatamente ritornando il valore calcolato in precedenza. Altrimenti aggiorna i valori di $d[v, S]$ e $\pi[v, S]$ applicando la definizione ricorsiva vista sopra. Per ottenere la soluzione al problema TSP è sufficiente creare d e π inizialmente vuoti e richiamare HK-VISIT a partire da $0, V$ (Algoritmo: 2)

Algoritmo 2 Inizializzazione e chiamata a HK-VISIT

```
function HK-TSP(G)
    let G = (V, E)
    d = ∅
    π = ∅
    return HK-VISIT(0, V)
end function
```

Analisi di complessità

Qual è la complessità computazionale dell'algoritmo di Held e Karp? Ogni esecuzione di HK-VISIT che non termina immediatamente esegue il ciclo **for** delle linee 9–15 che itera su tutti i vertici in $S \setminus \{v\}$ e quindi impiega tempo $O(n)$, escluse le chiamate ricorsive. Siccome le chiamate a HK-VISIT terminano immediatamente quando la coppia v, S è presente in d, il numero di chiamate che eseguono le istruzioni poste dalla linea 7 in poi è limitato superiormente dal numero di elementi che possiamo avere in d. Siccome il vettore d è indicizzato con coppie (v, S) dove v è un vertice del grafo e S un sottoinsieme dei vertici, abbiamo che il numero di elementi di d sarà al più $n \cdot 2^n$, con $n = |V|$. Quindi il tempo totale impiegato da HK-VISIT è $O(n) \cdot O(n \cdot 2^n) = O(n^2 \cdot 2^n)$. Si tratta di un tempo esponenziale che, per quanto elevato, è comunque inferiore all'$O(n!)$ della soluzione "a forza bruta" che itera su tutti i possibili cicli di n vertici per trovare il minimo.

TSP: euristiche costruttive

Con il termine "euristiche costruttive" si identifica una ampia famiglia di euristiche che arrivano alla soluzione procedendo un vertice alla volta seguendo delle regole prefissate. Lo schema generale di queste euristiche è composto da tre passi:

1. **Inizializzazione:** scelta del ciclo parziale iniziale (o del punto di partenza);
2. **Selezione:** scelta del prossimo vertice da inserire nella soluzione parziale;
3. **Inserimento:** scelta della posizione dove inserire il nuovo vertice.

Vediamo ora come queste regole vengono istanziate in cinque euristiche costruttive diverse.

Nearest Neighbor

1. **Inizializzazione:** si parte con il cammino composto dal solo vertice 0
2. **Selezione:** sia $(v_0, \dots, v_k)$ il cammino corrente. Trova il vertice v_{k+1} non ancora inserito nel circuito a distanza minima da v_k;
3. **Inserimento:** inserisci v_{k+1} subito dopo v_k nel cammino;
4. ripeti da (2) finché non hai inserito tutti i vertici nel cammino;
5. chiudi il circuito inserendo il vertice iniziale 0 alla fine del cammino $(0, \dots, v_n)$.

Random Insertion

1. **Inizializzazione:** considera il circuito parziale composto dal solo vertice 0. Trova un vertice j che minimizza $w(0, j)$ e costruisci il circuito parziale $(0, j, 0)$;
2. **Selezione:** seleziona in modo casuale un vertice k non ancora inserito nel circuito;
3. **Inserimento:** trova l'arco $\{i, j\}$ del circuito parziale che minimizza il valore $w(i,k) + w(k, j) - w(i, j)$ e inserisci k tra i e j;
4. ripeti da (2) finché non hai inserito tutti i vertici nel circuito.

Le euristiche "Closest Insertion", "Farthest Insertion" e "Cheapest Insertion" si differenziano da Random Insertion nella selezione del vertice, che non è più arbitrario ma segue una regola precisa. Inizializzazione e Inserimento non variano.

Cheapest Insertion

1. **Inizializzazione:** considera il circuito parziale composto dal solo vertice 0. Trova un vertice j che minimizza $w(0, j)$ e costruisci il circuito parziale $(0, j, 0)$;
2. **Selezione:** trova un vertice k non presente nel circuito parziale e un arco $\{i, j\}$ del circuito parziale che minimizzano il valore $w(i,k) + w(k, j) - w(i, j)$;
3. **Inserimento:** inserisci k tra i e j;
4. ripeti da (2) finché non hai inserito tutti i vertici nel circuito.

Dato un insieme di vertici $C \subseteq V$ che rappresentano un circuito parziale, ed un vertice $k \notin C$, definiamo la *distanza* di k da C come il minimo peso di un arco che collega k a C:

$$\delta(k, C) = \min_{h \in C} w(h, k)$$

Le euristiche "Closest Insertion" e "Farthest Insertion" selezionano i vertici che minimizzano o massimizzano la distanza dal circuito parziale.

Closest Insertion

1. **Inizializzazione:** considera il circuito parziale composto dal solo vertice 0. Trova un vertice j che minimizza $w(0,j)$ e costruisci il circuito parziale $(0,j,0)$;
2. **Selezione:** trova un vertice k non presente nel circuito parziale C che minimizza $\delta(k,C)$;
3. **Inserimento:** trova l'arco $\{i,j\}$ del circuito parziale che minimizza il valore $w(i,k)+w(k,j)-w(i,j)$ e inserisci k tra i e j;
4. ripeti da (2) finché non hai inserito tutti i vertici nel circuito.

Farthest Insertion

1. **Inizializzazione:** considera il circuito parziale composto dal solo vertice 0. Trova un vertice j che minimizza $w(0,j)$ e costruisci il circuito parziale $(0,j,0)$;
2. **Selezione:** trova un vertice k non presente nel circuito parziale C che massimizza $\delta(k,C)$;
3. **Inserimento:** trova l'arco $\{i,j\}$ del circuito parziale che minimizza il valore $w(i,k)+w(k,j)-w(i,j)$ e inserisci k tra i e j;
4. ripeti da (2) finché non hai inserito tutti i vertici nel circuito.

Fattori di approssimazione delle euristiche costruttive.
È possibile dimostrare che le cinque euristiche costruttive permettono di trovare una soluzione $\log(n)$-approssimata a TSP, quando la disuguaglianza triangolare è rispettata. Nel caso di Closest Insertion e Cheapest Insertion il fattore di approssimazione può essere ulteriormente abbassato, dimostrando che entrambe le euristiche permettono di ottenere un algoritmo 2-approssimato per TSP (quando la disuguaglianza triangolare è rispettata).