

Travelling Salesman Problem

Giacomo Costarelli Cesare Iurlaro Giuseppe Gabbia

Università degli studi di Torino

10 Ottobre, 2019

- 1 Formulazione del problema
- 2 Tour
- 3 Intorni di ricerca locale
- 4 Implementazione
- 5 Risoluzione del TSP con Self Organizing Map
- 6 Osservazioni e risultati

- 1 Formulazione del problema
- 2 Tour
- 3 Intorni di ricerca locale
- 4 Implementazione
- 5 Risoluzione del TSP con Self Organizing Map
- 6 Osservazioni e risultati

Formulazione del problema

Dato un **Travelling Salesman Problem** definiamo:

- un commesso viaggiatore
- un insieme $\{c_1, c_2, \dots, c_n\}$ di città
- e, per ogni coppia distinta di città (c_1, c_2) , una distanza $d(c_1, c_2)$.

Il problema consiste nel trovare un ordinamento π delle città che minimizzi la distanza totale che il commesso dovrà percorrere nel visitare tutte le città una ed una sola volta, ritornando infine nella città di partenza.

Formulazione del problema

Formalmente, le città sono solitamente formulate come nodi di un grafo $G = (V, E)$ dove:

- $V = \{c_1, c_2, \dots, c_n\}$
- $E = \{ij : i, j \in V\}$

All'interno della formulazione da noi trattata supponiamo che il grafo **non** sia orientato, dando così origine ad un problema definito come **TSP** **simmetrico**, e che valga:

$$\forall ij \in E : d(c_i, c_j) = d(c_j, c_i)$$

Per le istanze da noi considerate sono imposte sostanziali restrizioni per quanto riguarda i tipi di distanze consentite.

In particolare, le distanze devono rispettare la cosiddetta **disuguaglianza triangolare**:

$$\forall i, j, k, 1 \leq i, j, k \leq N : d(c_i, c_j) \leq d(c_i, c_k) + d(c_k, c_j)$$

- 1 Formulazione del problema
- 2 **Tour**
- 3 Intorni di ricerca locale
- 4 Implementazione
- 5 Risoluzione del TSP con Self Organizing Map
- 6 Osservazioni e risultati

Un **tour** è una sequenza ammissibile di città che il commesso viaggiatore visita per portare a termine il suo compito.

Un **tour iniziale** è un tour, ottenuto attraverso l'applicazione di un'euristica, sul quale successivamente applicare un intorno per migliorare la soluzione iniziale ed avvicinarsi quanto più possibile alla soluzione ottima.

Euristiche per tour iniziali

Vi sono differenti euristiche per la generazione di un tour iniziale.

Tra di esse troviamo:

- **Random tour**
- **Nearest Neighbor**
- Greedy
- Clarke-Wright
- Christofides

Random tour

Random tour parte dall'insieme di città disponibili all'interno del problema trattato e, a partire da esso, genera un ordinamento $c_{\pi(1)}, \dots, c_{\pi(N)}$ di città in maniera casuale.

Complessità

La complessità temporale dell'euristica Random tour è pari a $O(n)$.

Nearest Neighbor

Nearest Neighbor è l'euristica più naturale e più utilizzata per la costruzione di tour iniziali per il TSP.

Costruisce un ordinamento $c_{\pi(1)}, \dots, c_{\pi(N)}$ di città, con la città iniziale $c_{\pi(1)}$ scelta in modo arbitrario e, più, in generale $c_{\pi(i+1)}$ scelta in modo tale da minimizzare:

$$\{d(c_{\pi(i)}, c_k) : k \neq \pi(j), 1 \leq j \leq i\}$$

Complessità

La complessità temporale dell'euristica Nearest Neighbor, così descritta, è pari a $O(n^2)$.

- 1 Formulazione del problema
- 2 Tour
- 3 Intorni di ricerca locale**
- 4 Implementazione
- 5 Risoluzione del TSP con Self Organizing Map
- 6 Osservazioni e risultati

Per cercare di migliorare la soluzione inizialmente ottenuta tramite l'applicazione dell'euristiche menzionate in precedenza, abbiamo utilizzato **due** degli **intorni di ricerca** locale più conosciuti per il TSP, ovvero:

- 2-OPT
- 3-OPT

2-OPT

Il primo intorno di ricerca locale prende il nome di **2-OPT**. [Croes, 1958]

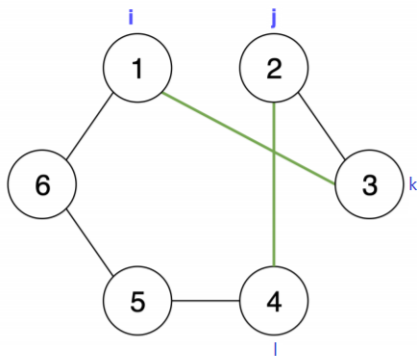
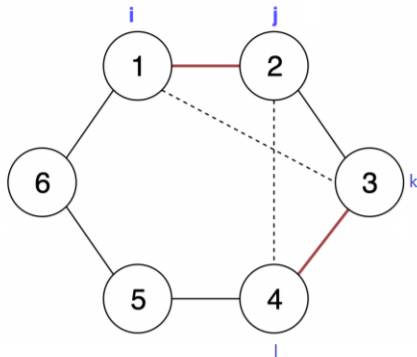
Questa tecnica rappresenta una soluzione ottimizzata per il TSP (sia simmetrico che asimmetrico).

2-OPT cerca di migliorare, in modo iterativo, una soluzione ammissibile iniziale sino a che non raggiunge un ottimo locale e non sia più possibile migliorare tale soluzione,

Dato un tour \mathcal{T} , per ogni coppia di archi $ij, kl \in \mathcal{T} : j \neq k$:

- **elimino** gli archi ij, kl
- per effettuare lo scambio degli archi **verifico** che valga la seguente condizione: $d(c_i, c_k) + d(c_l, c_j) < d(c_i, c_j) + d(c_k, c_l)$
- **riconnetto** il tour \mathcal{T} in maniera tale da rispettare i vincoli di ammissibilità del problema

2-OPT: esempio



Archi considerati: (1,2) (3,4) Percorso risultante: [1, 3, 2, 4, 5, 6, 1]

3-OPT

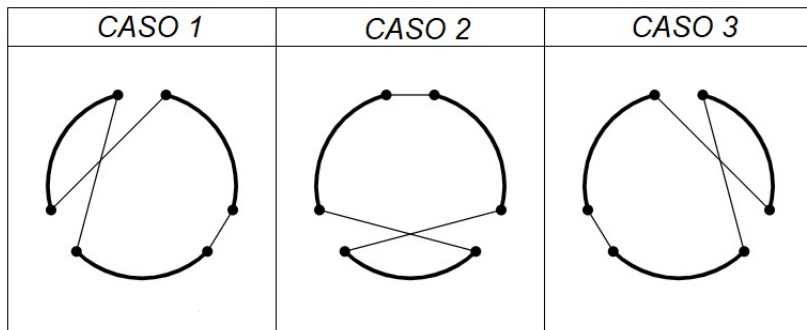
Il secondo intorno di ricerca locale prende il nome di **3-OPT**. [Lin, Kernighan, 1973]. Esso nasce come un'evoluzione dell'intorno di ricerca locale precedentemente visto.

In 3-OPT si rimuovono tre archi ottenendo così tre segmenti aperti all'interno del tour analizzato.

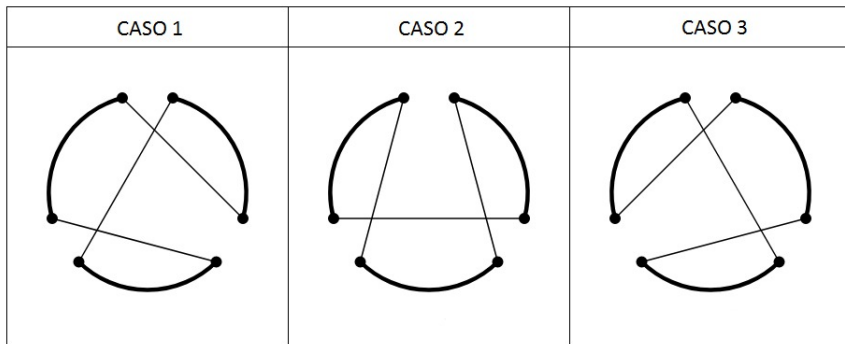
Così facendo si otterranno **otto** possibili **combinazioni** per effettuare la riconnessione del tour. Tali combinazioni sono suddivisibili in **tre sottogruppi** corrispondenti all'effettuare:

- l'applicazione di un singolo swap 2-OPT
- l'applicazione consecutiva di due swap 2-OPT
- l'applicazione consecutiva di tre swap 2-OPT

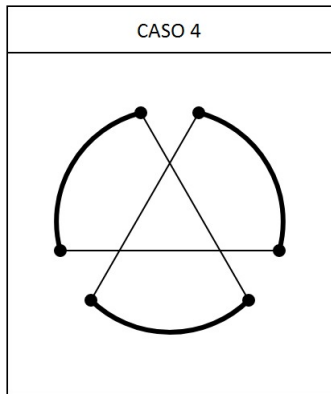
3-OPT: singolo swap 2-OPT



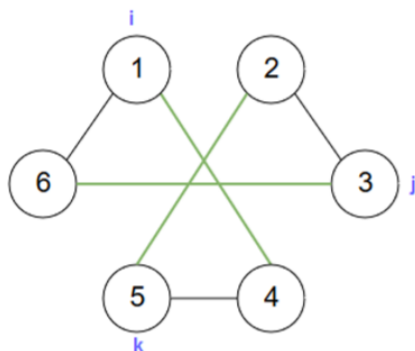
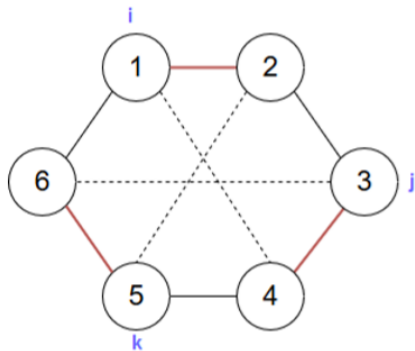
3-OPT: due swap 2-OPT



3-OPT: tre swap 2-OPT



3-OPT: esempio



Archi considerati: (1,2) (3,4) (5,6) Percorso risultante: [1, 4, 5, 2, 3, 6, 1]

- 1 Formulazione del problema
- 2 Tour
- 3 Intorni di ricerca locale
- 4 Implementazione**
- 5 Risoluzione del TSP con Self Organizing Map
- 6 Osservazioni e risultati

Implementazione: risorse utilizzate

Le istanze su cui abbiamo lavorato provengono dalla libreria **TSPLIB** la quale contiene insiemi di istanze TSP (e problemi simili ad esso) provenienti da varie fonti e di vario tipo.

Le **istanze** prese in considerazione sono:

- berlin52 [7542]
- eil101 [629]
- kroA100 [21282]
- pr76 [108159]
- st70 [675]

Implementazione: distanza tra i nodi

All'interno di ciascun file TSP sono presenti dei nodi con le loro rispettive coordinate spaziali.

Per calcolare le rispettive distanze tra i nodi presenti in ciascuna delle istanze da noi scelte abbiamo fatto riferimento alla formula della **distanza euclidea**:

$$d(n_1, n_2) = \sqrt{(x_{n_1} - x_{n_2})^2 + (y_{n_1} - y_{n_2})^2}$$

Implementazione: strutture dati utilizzate

Le due strutture dati di maggior rilevanza presenti all'interno dell'implementazione sono:

- la **matrice delle distanze**
- ed il **tour**.

Implementazione: matrice delle distanze

La **matrice**, avente dimensione $n \times n$ con n numero di nodi del problema, viene utilizzata per rappresentare le istanze TSP da noi trattate.

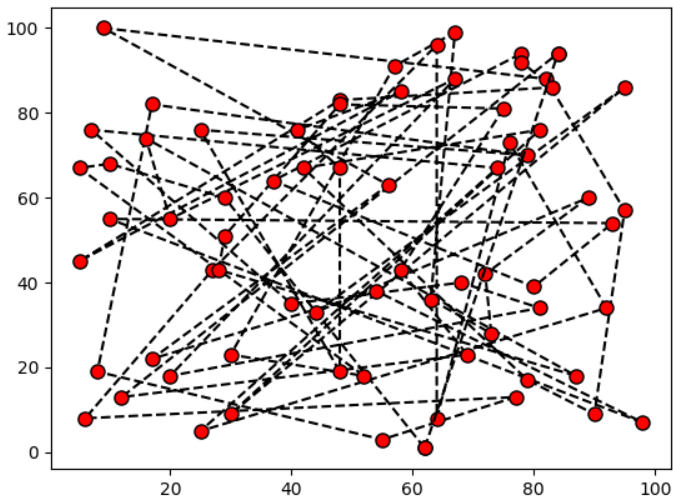
$$\begin{bmatrix} \infty & 666.0 & 281.0 & 396.0 & 291.0 \\ 666.0 & \infty & 649.0 & 1047.0 & 945.0 \\ 281.0 & 649.0 & \infty & 604.0 & 509.0 \\ 396.0 & 1047.0 & 604.0 & \infty & 104.0 \\ 291.0 & 945.0 & 509.0 & 104.0 & \infty \end{bmatrix}$$

Implementazione: tour

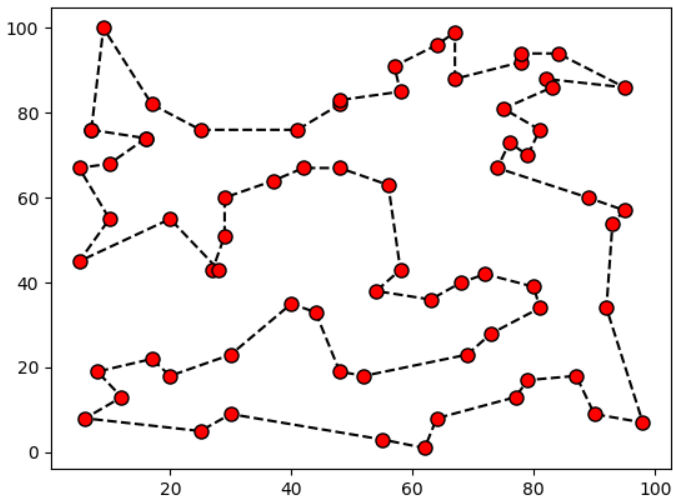
Il **tour**, implementato attraverso una classe definita come ***Tour***, viene utilizzato per **memorizzare il risultato** del:

- tour ottimo associato al problema trattato
- tour iniziale creato tramite l'applicazione di una delle euristiche tra Random o NN
- tour finale ottenuto tramite l'applicazione di un intorno di ricerca locale tra 2-OPT o 3-OPT.

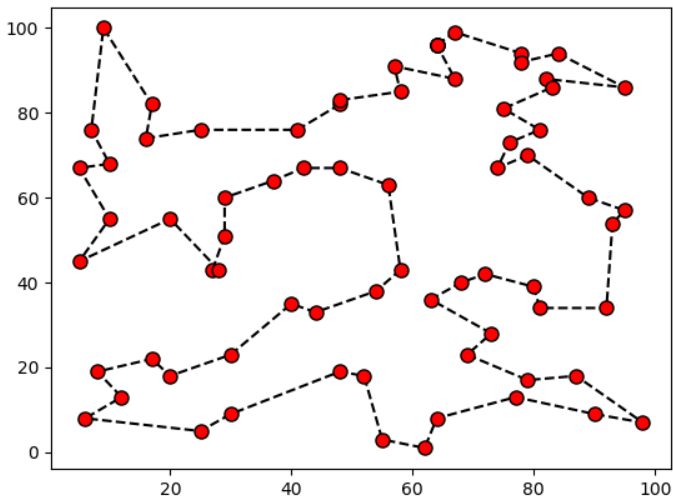
st70: tour iniziale ottenuto con Random Tour



st70: tour ottenuto con 3-OPT



st70: tour ottimo



- 1 Formulazione del problema
- 2 Tour
- 3 Intorni di ricerca locale
- 4 Implementazione
- 5 Risoluzione del TSP con Self Organizing Map**
- 6 Osservazioni e risultati

Risoluzione del TSP con Self Organizing Map

Gli approcci esistenti per risolvere TSP con reti neurali possono essere divisi in due classi principali:

- **Approccio topologico**: organizzazione di neuroni in funzione di una formulazione del TSP come PLI
- **Approccio geometrico**: modellazione di neuroni come punti nello spazio vettoriale

Nell'implementazione svolta abbiamo applicato un approccio di tipo geometrico attraverso l'**architettura SOM**.

Self Organizing Map (SOM)

SOM è una delle possibili architetture per un **approccio geometrico**.

Tale architettura modella ciascun neurone j con un vettore di pesi w_j^T .

In fase di apprendimento, attraverso una *funzione discriminante* $i(x)$, essa computa una qualche **misura di similarità** tra w_j^T ed x :

$$i(x) = \underset{j}{\operatorname{argmin}} \|x - w_j^T\|$$

Applicare questa funzione significa designare un neurone j al fine di aggiornarlo e renderlo più simile all'input x per ottenere infine un *mapping* in una dimensionalità ridotta.

Caratteristiche di una SOM per TSP

- La **funzione di similarità** è la distanza euclidea
- L'input i -esimo è una coppia di coordinate (x_i, y_i)
- I neuroni al termine dell'apprendimento:
 - rappresentano ciascuno un nodo del grafo (output unidimensionale **non circolare**)
 - sono mappati ciascuno a una coppia di coordinate che lo identifichi
 - sono **vicini** (simili) tra loro se adiacenti, **distanti** (dissimili) proporzionalmente alla lunghezza del percorso che li divide

Quindi l'output è un vettore **non** circolare di nodi ordinati in funzione della distanza reciproca.

- 1 Formulazione del problema
- 2 Tour
- 3 Intorni di ricerca locale
- 4 Implementazione
- 5 Risoluzione del TSP con Self Organizing Map
- 6 Osservazioni e risultati**

Osservazioni : Ricerca locale

Durante le varie simulazioni abbiamo osservato come l'euristica di costruzione del tour iniziale influenzi, in maniera importante, sia il numero di intorni da esplorare sia la soluzione finale che sarà possibile ottenere.

Nello specifico abbiamo che attraverso l'utilizzo dell'euristica:

- **Random tour** otteniamo un alto numero di intorni da esplorare (quindi una soluzione iniziale poco performante) e solamente un'unica volta il risultato migliore
- **Nearest Neighbor** otteniamo un numero inferiore di intorni da esplorare (quindi una soluzione iniziale migliore rispetto alla precedente) e nella quasi totalità delle volte un risultato molto vicino alla rispettiva soluzione ottima del problema trattato.

Osservazioni : Self Organizing Maps

I risultati ottenuti sono qualitativamente poco interessanti sia per quanto riguarda il tempo necessario alla risoluzione del TSP attraverso la SOM sia per quanto riguarda il costo delle varie soluzioni ottenute.


Ciò, molto probabilmente, può essere ricondotto al tipo di rappresentazione unidimensionale dell'input all'interno della SOM stessa che non riesce a considerare la distanza tra il primo e l'ultimo nodo facenti parte del tour da restituire in output.


Risultati


Problema	Tour iniziale	2-OPT	3-OPT	Soluzione
berlin52	Random	0.110s	1s	7569
berlin52	NN	0.030s	0.575s	7690
eil101	Random	5s	11s	648
eil101	NN	0.2s	6s	632
kroA100	Random	5s	30s	21621
kroA100	NN	0.7s	28.6s	21383
pr76	Random	1.80s	4.4s	109975
pr76	NN	0.3s	11.4s	109067
st70	Random	1.12s	7.17s	692
st70	NN	0.241s	9s	682


Confronto finale

Problema	Risoluzione	Tempo	Soluzione
berlin52	Random	0.110s	7569
berlin52	NN	0.030s	7690
berlin52	SOM	5.23s	9081
eil101	Random	5s	648
eil101	NN	0.2s	632
eil101	SOM	5.77s	705
kroA100	Random	5s	21621
kroA100	NN	0.7s	21383
kroA100	SOM	5.91s	23557
pr76	Random	1.80s	109975
pr76	NN	0.3s	109067
pr76	SOM	5.38s	118339
st70	Random	1.12s	692
st70	NN	0.241s	682
st70	SOM	5.34s	718

 G.A. Croes (1958)
A Method for Solving Traveling-Salesman Problems
Operations Research 6(6), 791 – 812.

 Lin, Shen; Kernighan, B. W (1973)
An Effective Heuristic Algorithm for the Traveling-Salesman Problem
Operations Research 21(2), 498 – 516.

 Johnson, David S.; McGeoch, Lyle A. (1997)
The Traveling Salesman Problem: A Case Study in Local Optimization
Local Search in Combinatorial Optimization, 215 – 310.

 Potvin, Jean-Yves (1993)
The Traveling Salesman Problem: A Neural Network Perspective