

|  |
| --- |
| Immagine che contiene testo  Descrizione generata automaticamente  Scuola Politecnica e delle Scienze di Base  Corso di Laurea Magistrale in Ingegneria Informatica |

|  |
| --- |
| Elaborato **Ricerca Operativa**  **Simulated annealing for TSP**  Anno Accademico 2020/21 |

|  |
| --- |
| **Michele Maresca M63/1151**  **Vincenzo Riccardi M63/1146** |

Sommario

[Simulated annealing for TSP 3](#_Toc73302456)

[1. Main 3](#_Toc73302457)

[2. Costruttore 4](#_Toc73302458)

[3. Calcolo\_soluzione\_iniziale 4](#_Toc73302459)

[4. Funzione\_obbiettivo e calcolo distanza 6](#_Toc73302460)

[5. Calcolo\_probabilità\_accettazione 6](#_Toc73302461)

[6. Avanza\_t 7](#_Toc73302462)

[7. Mossa2\_opt 7](#_Toc73302463)

[8. Simulatedannealing 7](#_Toc73302464)

[9. Visualizza tour 8](#_Toc73302465)

# Simulated annealing for TSP

Il codice python prodotto consiste in due file principali un main.py che contiene la funzione main e leggi coordinate specificate nel paragrafo 1. Il secondo file prodotto invece è una classe chiamata TSP.py che contiene le funzioni raccontate nei restanti paragrafi, per informazioni rapide cliccare il numero vicino la funzione nell’indice.

## Main

Nel main la prima funzione che incontriamo è “leggi\_coordinate(path)” dove salviamo la linea del file dove è presente la dimensione del problema ovvero il numero di nodi. Nel caso del file burma14.txt salviamo l’indice d’inizio linea dimensione cioè 4 procediamo ugualmente per le coordinate con indice a 9. Questti passaggi tornano utili perché aprendo il file con readlines leggo tutte le linee come fosse una lista di linee di testo e con l’indice giusto punto all’informazione cercata. Per avere il valore numerico della dimensione, per esempio, uso la strip per dividere la riga e leggo il dato cercato. Con un for invece ciclo sulle coordinate con indice ciclo la dimensione e salvo tutto nella lista di coppie che ho creato che è chiamata “coordinate”. Salvo, infine, una variabile dove memorizzo la best solution conosciuta.  
Nel main chiamo la funzione TSP per che è il costruttore del nostro modello di problema del commesso viaggiatore.  
Dopo aver ottenuto il modello chiamo la funzione di tsp “simulatedannealing()” dove calcolo il valore dell’euristica infine calcolo il gap dalla best solution.  
L’ultima funzione visualize\_tour() stampa a video il percorso trovato dall’euristica.

## Costruttore

Passo come parametri le coordinate lette dal file temperatura iniziale e finale e il parametro alpha che regola la legge di decremento con la formula di decrescita lineare tk+1 = alpha\*tk. I valori letti nel file come temp\_iniziale = -1 indica che l’utente non ha settato parametro e il costruttore assegna un valore di default.  
Inizio ad inizializzare tutte le variabili della classe. Il dato L è un intero che indica quante iterazioni devo fare prima con lo stesso valore di temperatura, il valore scelto di default è la dimensione ovvero il numero di nodi da visitare nel ciclo hamiltoniano.  
La temperatura di default è una generica adatta per i problemi di tsp generico ed è calcolato come (10\*obj(sol\_iniziale))/2 (sol\_iniziale è la soluzione ottenuta con l’approccio gready), mentre la temperatura finale è ottenuta come 10^-4\*|obj(sol\_iniziale)|. Essendo che se chiamassi il costruttore non avrei ancora una soluzione iniziale quinid nell’euristica gready mi occuperò di settare i parametri de default se non passati dall’utente. Il parametro T è la temperatura all’iterazione generica ed è sempre sovrascritto. Setto il parametro alpha pari a quello immesso dall’utente o altrimento assegno come valore di default 0.99 se il problema ha meno di mille nodi da esplorare altrimenti in caso di nodi >= a 1000 per procedere più velocemente setto alpha pari a 0.8 (decremento rapido della temperatura tk+1).  
Creo la lista di nodi di dimensioni pari a quelle del problema N.  
Setto infine la best\_solution a none perché non è ancora stata trovata una soluzione, la best objective di conseguenza a infinito che indica che non ho trovato nulla procedo allo sesso modo per definire current\_solution e current\_objective.  
Setto iterazione pari a uno ed è l’indice di iterazioni alla stessa temperatura quando raggiunge il valore di L torna ad uno.

## Calcolo\_soluzione\_iniziale

DA CAMBIARE descrizione algoritmo

NON DA CAMBIARE :

setto best\_objective al valore trovato dalla gready, setto best\_solution e current\_solution pari alla soluzion\_iniziale, setto best\_objective e current objective pari al valore della funzione obbiettivo ottenuto dalla gready. Nella funzione setto eventualmente i valori di default di temperatura iniziale e finale, controllo che il loro valore sia uguale al valore non immissione cioè -1 ed assegno a t0 il valore 10\*(|obj(sol\_in)|/2), mentre la soluzione finale è pari a 10^-4\*|obj(sol\_in)|.

## Funzione\_obbiettivo e calcolo distanza

Il parametro in ingresso è solution che è una lista di nodi. Resetto la current obj perché va aggiornata uso un for ciclando su tutti i nodi e calcolo di volta in volta le distanze tra i nodi successivi chiamando la funzione calcolo distanza e gestisco il vettore dei nodi come un vettore circolare perché mi serve la distanza anche tra ultimo nodo e primo per chiudere il circuito. Calcolo distanze calcola la distanza euclidea tra ii due nodi in ingresso usando le loro coordinate.

## Calcolo\_probabilità\_accettazione

Riceve in ingresso una soluzione candidata ed è sempre una lista generica otteuta ad un iterazione generica dell’algoritmo, la prima cosa che facciamo e chiamare funzione\_obbiettivo per avere il valore della obj e lo sostituiamo a candidate\_obj vecchio. Ora se il valore della funzione obbiettivo è minore o uguale alla obj attuale allora sostituisci perché avrò probabilità 1 di sostituzione del valore di current\_objective e current\_solution che diventano rispettivamente candidate\_objective e candidate\_solution, essendo che è una sostituzione migliorativa verifico se è migliore anche della best\_solution e best\_objective in caso sia meglio dell’ottimo trovato fino a quel momento lo sostituisci.  
Se invece la soluzione candidata è peggiorativa abbiamo una probabilità pari a e^-((candidate\_objective-current\_objective)/tk) di aggiornare la soluzione corrente, per fare ciò usiamo la funzione random che calcola un numero randomico tra 0 ed 1 estremo superiore escluso (distrubuzione di probabilità uniforme) se è minore al valore e^-((candidate\_objective-current\_objective)/tk) allora sostituisci current\_objective e current\_solution che diventano rispettivamente candidate\_objective e candidate\_solution, non aggiorno la best perché è il caso di soluzione peggiorativa, se il valore di rand non è minore ad e^… non fare nulla.

PS:

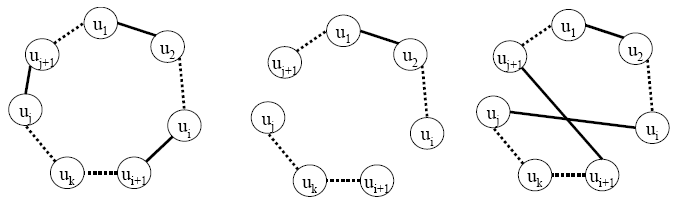
tk = tempratura attuale (T nel codice)

## Avanza\_t

Non ha parametri di input e controlla se il parametro iterazione è pari a L numero di iterazioni massime alla stessa temperatura, se così fosse decrementiamo la tk calcolando tk+1 = alpha\*tk, altrimenti incrementa il valore di iterazione.

## Mossa2\_opt

La funzione mossa2\_opt ha come parametri di ingresso la soluzione candidata e prevede di eseguire lo scambio di collegamenti eseguendo un incrocio a X degli archi come in figura.



Per eseguire lo scambio in modo randomico usiamo un generatore di interi randomici in un intervallo che va da 0 a dimensione del problema meno 1 in modo da pescare un valore randomico nella lista dei nodi della soluzione candidata A quel punto inizializziamo una lista free\_nodes uguale a candidate, ed estraiamo dalla lista il nodo candidato a quel punto sulla lista free\_nodes che sarà priva del nodo i puntato, estraiamo il valore successivo ad i e precedente in modo da non pescare i nodi adiacenti ad i che non possono essere selezionati, i problemi ai bound sono gestiti con un vettore circolare. Calcoliamo j che sarà sicuramente un nodo non adiacente ad i e

## Simulatedannealing

È la funzione fulcro della classe non ha parametri di input e gestisce l’avanzare dell’euristica, consiste in una serie di chiamate a funzioni della classe in modo ordinato.  
Inizialmente chiamo l’euristica gready per i calcolo della soluzione iniziale ed eventualmente il settaggio dei parametro di temperatura iniziale e finale se non specificati dall’utente, in sostanza chiamo la funzione calcolo\_soluzione\_iniziale (3). Il punto focale della funzione è un ciclo while che si arresta se la temperatura attuale T è minore alla temperatura finale. Nel ciclo eseguo la mossa2\_opt per cambiare la posizione dei nodi nella soluzione attuale e chiamo il calcolo della probabilità di accettazione della soluzione candidata ottenuta dalla mossa, in sostanza eseguo una chiamata a mossa2\_opt (7) e una chiamata a calcolo\_probailità\_accettazione (5).  
A questo punto vedo se ho terminato le iterazioni alla stessa temperatura o devo aggiornarla e quindi chiamo la funzione avanza\_t (6). All’uscita del while stampo il valore della funzione obbiettivo migliore trovata con l’algoritmo ed il tour calcolato. La funzione ritorna il valore best\_objective in modo da calcolare il gap dalla soluzione esatta nel main.

## Visualizza tour

DA DEFINIRE