Progetto: Iterated Local Search per S-TSP

Beatrice Laureti

Università degli studi di Ferrara

18 febbraio 2025

ILS per S-TSP

- **Progetto**: Applicare una Iterated Local Search per il TSP simmetrico, comparando l'utilizzo di alcuni intorni diversi sia per la LS che per la mossa di diversificazione.

Problema S-TSP: Modello matematico

- Dato un grafo $G = (V, E)$ indiretto, completo, pesato $c : E \to \mathrm{R}^+$, si cerca il ciclo hamiltoniano di costo minimo.
- **Modello:**

$$\min \sum_{ij \in E} c_{ij} x_{ij}$$

Soggetto alle seguenti condizioni:

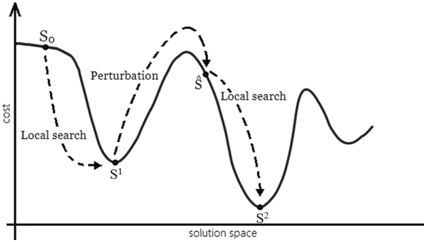
$$\sum_{ij \in E(V1,V2)} x_{ij} \geq 2 \quad \forall \text{ taglio}(V1, V2)$$

$$\sum_{ij \in S(i)} x_{ij} = 2 \quad \forall\ i \in V$$

$$x_{ij} \in \{0, 1\} \quad \forall\ (i, j) \in E$$

Iterated Local Search: Template

$x_0 \leftarrow$ GenerateInitialSolution;
$x^* \leftarrow$ LocalSearch(x_0);
repeat
  $x' \leftarrow$ Perturbation(x^*, history);
  $x^{*'} \leftarrow$ LocalSearch(x');
  $x^* \leftarrow$ AcceptanceTest($x^*, x^{*'}$, history);
until *StopCondition*;



Implementazione dell'algoritmo

Implementazione dell'algoritmo

- MainTSP_ILS → main ILS
- Nearest_neighbour → soluzione iniziale trovata con NN
- LS → Local search
- Perturbation → mossa di diversificazione
  - two_opt → soluzione ottimale nell'intorno 2-opt
  - three_opt → soluzione ottimale nell'intorno 3-opt
  - city_swap → soluzione casuale nell'intorno city swap
  - city_swap_c → soluzione ottimale nell'intorno city swap
  - city_insert → soluzione casuale nell'intorno city insert
  - city_insert_c → soluzione ottimale nell'intorno city insert
  - double_bridge → soluzione casuale nell'intorno double bridge
- AcceptanceTest → criteri di accettazione del nuovo ottimo locale
- costo → costo di una soluzione

Generazione della soluzione iniziale: Nearest Neighbour

Per generare la soluzione iniziale ho utilizzato la greedy Nearest-Neighbour.

Descrizione algoritmo:

- L'algoritmo *Nearest Neighbour* inizia da un nodo arbitrario i, e aggiunge l'arco di costo minimo adiacente a i, sia questo (i, j).
- j diventa il nuovo nodo corrente e si itera su j fino a chiudere il ciclo dopo n iterazioni.

- Esempio tipico di come scelte iniziali localmente ottime possono portare a soluzioni di pessima qualità nei passi finali poichè i vincoli di ammissibilità restringono le scelte.

Implementazione NN

function [n,c,Mdist0] = Nearest_Neighbour (Mcord)

- Funzione che accetta in input le coordinate dei nodi (matrice $2 \times N$) e che restituisce in output la soluzione trovata, il suo costo e la matrice dei costi degli archi.
- Si parte prendendo un nodo casuale p e si trova il nodo ns tale che (p, ns) sia l'arco di costo minimo, attraverso la matrice dei costi.
- Una volta scelto tale nodo, si aggiornano i valori della matrice riguardanti i costi del nodo iniziale (riga e colonna p-esima), ponendoli uguali ad ∞, in questo modo quando l'algoritmo andrà a cercare il nodo successivo da selezionare, sicuramente non sceglierà un nodo già scelto.
- Si itera il procedimento sul nodo scelto ns fino a quando non vengono scelti tutti i nodi.

Implementazione Local Search

function [n,c,it] = LS (n_0,Mdist)

- Funzione che accetta in input una soluzione iniziale n_0 e la matrice dei costi degli archi e restituisce in output la soluzione trovata dalla local search, il suo costo e il numero di iterazioni che ha eseguito per trovarla.
- LS consiste nella ricerca della soluzione ottima locale a partire dalla soluzione n_0 utilizzando un intorno prestabilito:
  - A partire da n_0 si trova la miglior soluzione nell'intorno di n_0, n.
  - Si itera su n.
  - Termina quando non vi è più miglioramento della soluzione (è stato raggiunto l'ottimo locale nel bacino di attrazione da cui partiva n_0) o se viene raggiunto un numero massimo di iterazioni prestabilito maxit.
- Nei casi city_swap e city_insert si prende una soluzione migliorante, non è detto che sia la migliore dell'intorno. In questo caso termina dopo maxit iterazioni.

Implementazione Perturbation

function n = perturbation(n_i,Mdist)

- La funzione perturbazione accetta in input una soluzione iniziale n_i e la matrice dei costi degli archi Mdist e restituisce in output la solzuione n perturbata.
- La soluzione perturbata viene trovata a partire da n_i in un intorno prestabilito.

Implementazione AcceptanceTest

function b = AcceptanceTest(n,n_1,Mdist,his,tol)

- La funzione accetta in input:
  - n: ottimo locale precedente.
  - n_1: nuovo ottimo locale trovato con una LS a partire dalla perturbazione di n.
  - Mdist: matrice dei costi degli archi.
  - his: matrice le cui righe sono ottimi locali già visitati in precedenza.
  - tol: tolleranza di peggioramento.
- Restituisce in output *true* se viene accettata, *false* altrimenti.
- Determina se la nuova soluzione trovata con la ricerca locale (a partire dalla soluzione perturbata) deve essere accettata o no.
- n_1 viene accettata se:
  - Il costo di n_1 è inferiore al costo di n + tol. Ossia se n_1 migliora o se peggiora entro una certa tolleranza tol rispetto a n.
  - E se n_1 non è un elemento della matrice his. Ossia se n_1 non è un ottimo locale già visitato.

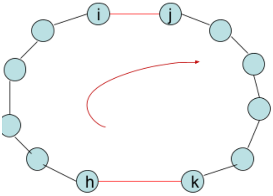
Intorni utilizzati

Sia per la ricerca locale, sia per la mossa di diversificazione, vengono usati intorni diversi, che forniscono prestazioni differenti. Gli intorni utilizzati sono:

- two_opt → soluzione ottimale nell'intorno 2-opt
- three_opt → soluzione ottimale nell'intorno 3-opt
- city_swap → soluzione casuale nell'intorno city swap
- city_swap_c → soluzione ottimale nell'intorno city swap
- city_insert → soluzione casuale nell'intorno city insert
- city_insert_c → soluzione ottimale nell'intorno city insert
- double_bridge → soluzione casuale nell'intorno double bridge

Intorno 2-opt

- L'intorno è dato da tutti i cicli hamiltoniani che differiscono dal ciclo corrente per una coppia di archi non adiacenti.
- Per ogni coppia di archi selezionati esiste un unico modo per riconnettere il ciclo.
- Dimensione dell'intorno: $O(n^2)$.



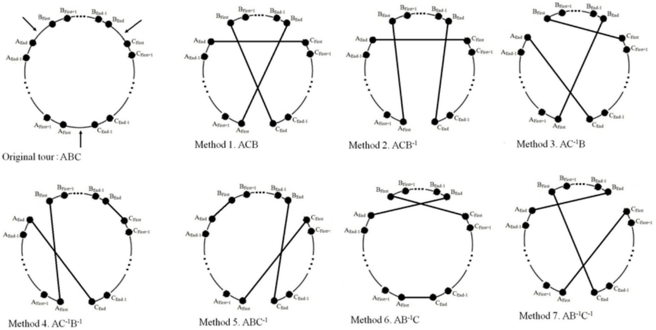
Implementazione intorno 2-opt

function [n,c] = two_opt (n_0,Mdist)

- La funzione accetta in input una soluzione n_0 e la matrice del costo degli archi Mdist e restituisce in output la soluzione migliore dell'intorno e il suo costo.
- Genero l'intorno, ossia trovo tutte le possibili coppie di archi da scambiare (gli archi non devono essere adiacenti) e costruisco le soluzioni in funzione di tali scambi.
- Trovo il miglior scambio per ispezione esaustiva, ossia trovo la soluzione che minimizza $c_n - c_0$.

Intorno 3-opt

- L'intorno è dato dalla sostituzione di tre archi non adiacenti nel ciclo.
- Per ogni terna di archi selenzionati ci sono i 7 modi per riconnettere il ciclo.
- Dimensione dell'intorno: $O(n^3)$



Implementazione intorno 3-opt

function [n,c] = three_opt (n_0,Mdist)

- La funzione accetta in input una soluzione n_0 e la matrice del costo degli archi Mdist e restituisce in output la soluzione migliore dell'intorno e il suo costo.
- Genero l'intorno:
  - Costruisco tutte le terne di archi ammissibili da eliminare (gli archi non devono essere adiacenti).
  - Per ognuna di queste trovo le 4 soluzioni ottenute scambiando i 3 archi per ottenere un ciclo hamiltoniano (scarto le tre soluzioni che scambiano solo 2 dei 3 archi).
- Tra tutte le soluzioni prodotte scelgo la migliore, ossia la soluzione che minimizza $c_n - c_0$.

Intorno city swap e city insert

- L'intorno city swap è dato da tutte le soluzioni ottenibili dallo scambio di due nodi nel ciclo.

1	2	3	4	5	6	7	8	9	10

1	7	3	4	5	6	2	8	9	10

- L'intorno city insert è dato da tutte le soluzioni ottenibili dalla rimozione di un nodo nel ciclo dalla sua attuale posizione e reinserimento in un altro punto.

1	2	3	4	5	6	7	8	9	10

1	2	4	5	6	7	3	8	9	10

Implementazione intorno city swap

function n = city_swap(n_0)

- Accetta in input una soluzione n_0 e restituisce in output una soluzione n, ottenuta dallo scambio di due nodi scelti casualmente

function [n,c] = city_swap_c (n_0,Mdist)

- Accetta in input una soluzione n_0 e la matrice dei costi degli archi Mdist e restituisce in output la soluzione migliore trovata dall'ispezione esaustiva dell'intorno e il suo costo.
- Genero l'intorno:
  - Trovo tutte le possibili coppie di nodi da scambiare.
  - Per ognuna di queste trovo la soluzione data dallo scambio di questi nodi e il suo costo.
- Tra tutte le soluzioni prodotte scelgo quella di costo minore.

Implementazione intorno city insert

function n = city_insert(n_0)

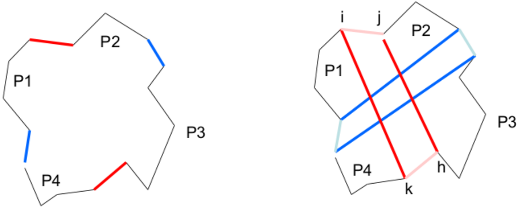
- Accetta in input una soluzione n_0 e restituisce in output una soluzione n, ottenuta dalla rimozione di un nodo (scelto casualmente) dalla sua attuale posizione e dal suo reinserimento in un altro punto scelto casualmente.

function [n,c] = city_insert_c (n_0,Mdist)

- Accetta in input una soluzione n_0 e la matrice dei costi degli archi Mdist e restituisce in output la soluzione migliore trovata dall'ispezione esaustiva dell'intorno e il suo costo.
- Genero l'intorno:
  - Trovo tutte le possibili coppie (nodo,nuova posizione).
  - Per ognuna di queste trovo la soluzione data dal reinserimento del nodo nella nuova posizione, e il suo costo.
- Tra tutte le soluzioni prodotte scelgo quella di costo minore.

Intorno double bridge

- L'intorno è dato dalle soluzioni ottenibili dallo scambio di 4 archi non adiacenti, eseguito nel seguente modo:
    - Si rimuovono i 4 archi scelti, dando origine a 4 cammini P1,P2,P3,P4, nell'ordine in cui erano percorsi nel ciclo.
    - Si riordinano i 4 cammini, inserendo i 4 archi necessari, secondo l'ordinamento P2,P1,P4,P3.



Implementazione intorno double bridge

function n = double_bridge (n_0)

- La funzione accetta in input una soluzione n_0 e restituisce in output una soluzione n nell'intorno double bridge di n_0.
- Si scelgono casualmente 4 archi non adiacenti.
- Si rimuovono tali archi e si ripristina il ciclo inserendo 4 archi in accordo con la mossa double bridge.

Criteri di terminazione dell'algoritmo

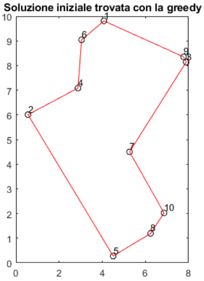
Nell'implementazione della ILS, ho scelto i seguenti criteri di arresto:

- Raggiungimento di un numero prefissato di iterazioni (maxit).
- Assenza di progressi per un numero prefissato di iterazioni (maxit_senza_migl).

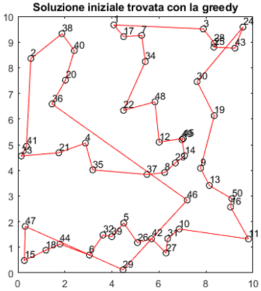
Risultati ottenuti: considerazioni generali

Soluzione iniziale trovata con la greedy NN

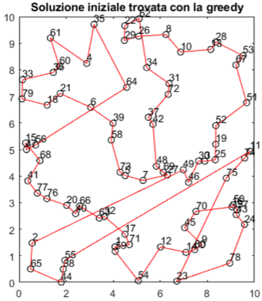
- $N \leq 10$: La miglior soluzione è trovata dalla greedy.
- All'aumentare del numero di città la soluzione trovata dalla greedy è sempre di qualità peggiore (si necessita l'utilizzo di una LS/ILS).



N=10



N=50



N=80

Soluzione trovata con la prima LS

- All'aumentare del numero di città N, le iterazioni svolte dalla prima LS aumentano, mettendoci più tempo a trovare ottimi locali.
- Questo accade poichè per N grandi, la greedy produce soluzioni di qualità sempre peggiore.
- Per valori di N ridotti (20-30), la LS è già in grado di produrre buone soluzioni. L'ILS, pur non portando sempre miglioramenti significativi, riesce a dare un piccolo vantaggio in alcune situazioni, perfezionando ulteriormente il risultato. Spesso non necessaria poichè aumenta di molto il costo computazionale per miglioramenti non significativi.

Soluzione trovata con ILS

- Per $N \leq 30$ l'ILS porta raramente miglioramenti significativi, poichè la prima LS è in grado di trovare già soluzioni di buona qualità.
- Per N molto grandi ILS tende a rimanere bloccata nell'ottimo locale trovato dalla prima LS, spesso non riuscendo a portare miglioramenti significativi.
- Questo accade perché la strategia di diversificazione utilizzata non riesce a introdurre abbastanza varietà per N grandi, limitando le opportunità di esplorare soluzioni migliori.

  Considerando che l'ILS ha un costo computazionale significativamente più elevato rispetto alla LS, per valori di N molto grandi non è vantaggioso utilizzare l'ILS, soprattutto se i miglioramenti che apporta risultano essere limitati.
- L'ILS si rivela particolarmente utile quando $40 \leq N \leq 70$, producendo soluzioni di qualità migliore rispetto alla prima LS. In questo caso, il costo computazionale è giustificato dai miglioramenti ottenuti.

Risultati ottenuti: utilizzo di intorni

Considerazioni su intorno utilizzato per la LS

- Come mossa di LS si sono mostrate più efficaci in ordine: 3-opt, 2-opt, city insert e city swap.
- In particolare, **3-opt** e **2-opt** trovano una soluzione con la prima LS già molto buona, arrivando a migliorare la soluzione iniziale in alcuni casi fino al 22%.
- 3-opt è quello più costoso a livello computazionale, 2-opt fornisce prestazioni simili ma con costo minore. Per N grandi (80-100) non è utilizzabile poichè troppo costoso.
- **City insert** per $N \leq 50$ fornisce prestazioni molto simili (a volte anche migliori) rispetto a 2-opt e 3-opt, arrivando a migliorare la soluzione iniziale fino al 14%.
- L'intorno **city swap** rispetto a tutti gli intorni produce soluzioni meno miglioranti (massimo fino al 7%).

Considerazioni su intorno utilizzato per la mossa di diversificazione

- La scelta dipende dall'intorno utilizzato nella LS, nonostante ciò si possono fare delle considerazioni generali.
- Tutti gli intorni impiegati come mossa di diversificazione hanno portato a risultati soddisfacenti.
- Tuttavia, i migliori risultati sono stati ottenuti con le mosse **double bridge** e **city insert**, che si rivelano particolarmente distruttive e capaci di sfuggire agli ottimi locali. In particolare, city insert ha dato ottimi risultati anche per valori di N più elevati (fino a 100), mentre double bridge ha mostrato le migliori prestazioni per N più ridotti, non riuscendo a diversificare abbastanza per N maggiori.

- **3-opt** ha avuto successo solo in alcune occasioni, mentre **2-opt** ha mostrato risultati ancora meno frequenti, con entrambe le tecniche che tendono a rimanere bloccate in minimi locali. Questo accade perché entrambe le mosse non introducono elementi di stocasticità, limitando la capacità di esplorare nuovi spazi di soluzione.
- **City swap** ha mostrato una limitata capacità di diversificazione, efficace solo con un numero ridotto di città (non abbastanza distruttiva, soprattutto per $N \geq 60$).
- **Conclusione:** Come mossa di diversificazione si sono mostrate più efficaci le mosse con una componente stocastica.
- Per valori di N molto grandi (80-100), tutte le mosse tendono a restare intrappolate in minimi locali, portando a soluzioni che non presentano miglioramenti significativi rispetto a quella trovata con la prima LS. Le migliori soluzioni ottenute, infatti, mostrano un miglioramento massimo del 3% rispetto alla soluzione ottenuta tramite la prima LS.

2-opt vs 3-opt

- In generale, 3-opt ha mostrato prestazioni superiori rispetto a 2-opt, ma il suo costo computazionale è notevolmente più elevato, soprattutto per valori di N grandi, rendendo l'uso di 3-opt poco praticabile.
- Nonostante le sue prestazioni migliori, il costo computazionale di 3-opt non giustifica i miglioramenti ottenuti, poiché le performance di entrambe le mosse risultano comunque simili.
- La combinazione 2-opt (LS) e 3-opt (diversificazione) consente un equilibrio tra qualità della soluzione e tempo di calcolo. Infatti, 2-opt fornisce buone soluzioni rapidamente, ottimizzando localmente il percorso, mentre 3-opt permette di esplorare nuove aree dello spazio delle soluzioni, contribuendo a evitare minimi locali.

City Insert/Swap: Certi vs Stocastici

- Gli intorni stocastici si sono rivelati decisamente più efficaci rispetto a quelli certi nella mossa di diversificazione. Infatti, i metodi certi tendevano a rimanere bloccati nei minimi locali.
- Gli intorni certi sono stati invece utilizzati per la ricerca locale, poiché quelli stocastici non esploravano in modo esaustivo l'intero intorno, limitandosi a trovare soluzioni miglioranti senza raggiungere l'ottimo locale.
- In alcune situazioni, nell'utilizzo di city swap e insert come intorni di diversificazione, è stato necessario introdurre una tolleranza al peggioramento (tol=2/3), che ha aiutato la soluzione ad uscire dagli ottimi locali.

Combinazioni migliori di intorni

Considerando le osservazioni precedenti e i risultati ottenuti, le combinazioni più efficaci di intorno per la ricerca locale e mossa di diversificazione per l'ILS sono:

- Per $N \geq 20$
  - Intorno LS: 2-opt - Perturbazione: city insert random
  - Intorno LS: 2-opt - Perturbazione: double bridge
  - Intorno LS: 2-opt - Perturbazione: 3-opt
- Per $20 \leq N \leq 70$
  - Intorno LS: 3-opt - Perturbazione: city insert random
  - Intorno LS: 3-opt - Perturbazione: city swap random
  - Intorno LS: 3-opt - Perturbazione: double bridge
- Per $N \leq 50$
  - Intorno LS: city insert certo - Perturbazione: double bridge

Altre considerazioni

- La soluzione iniziale generata in modo casuale, anziché tramite il metodo NN, non comporta grandi differenze, tranne per il numero di iterazioni della prima ricerca locale. Tuttavia, per valori elevati di N, si osservano differenze più significative.
- I parametri (come la tolleranza, il massimo numero di iterazioni, il massimo numero di iterazioni senza miglioramento) devono essere determinati in maniera empirica, come anche la combinazione degli intorni utilizzati.
- Nonostante l'algoritmo produca buoni risultati (con l'utilizzo dei giusti intorni), si potrebbe ottenere un possibile miglioramento tramite l'ibridazione, ossia combinando diverse tecniche per ottenere risultati superiori.

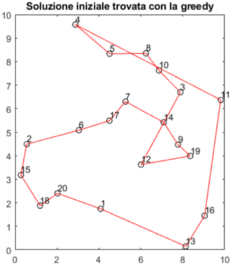
Risultati ottenuti: soluzioni migliori ottenute in funzione di N

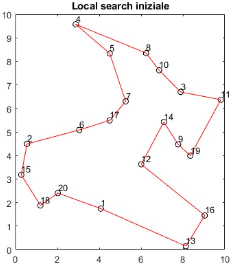
Grafici utilizzati

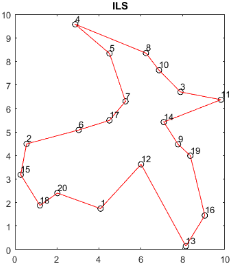
Per studiare i risultati e confrontare l'utilizzo dei vari intorni, sono stati utili alcuni grafici.

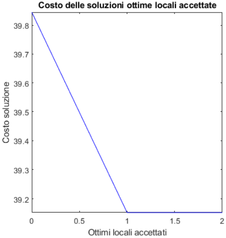
- I tre grafici che mostrano la soluzione ottenuta con la greedy, quella con la LS iniziale e quella con l'ILS.
- I due grafici che mostrano l'andamento del costo delle soluzioni accettate (con una certa tolleranza) nella ILS e l'andamento del costo della soluzione nel corso delle iterazioni.

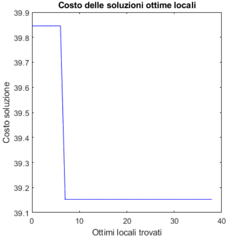
N=20: 2-opt - Double bridge











NN: 45.879089

LS: 39.845682 - 4 it

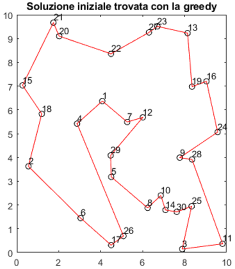
ILS: 39.152833 - 7° it - 2 acc.

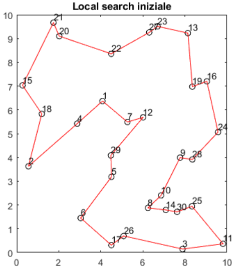
ILS-NN: 14.66%

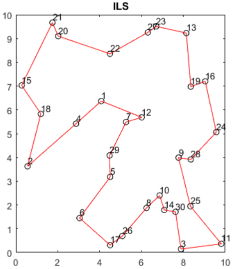
LS-NN: 13.15%

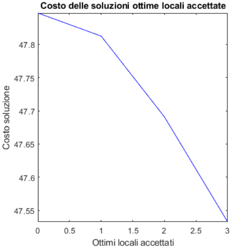
ILS-LS: 1.74%

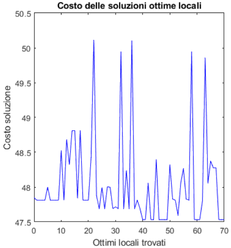
N=30: 3-opt - city swap random











NN: 52.395929

LS: 47.847414 - 5 it

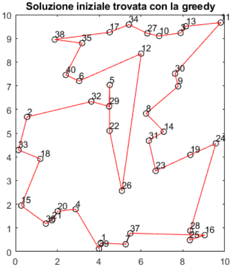
ILS: 47.532939 - 39° it - 3 acc.

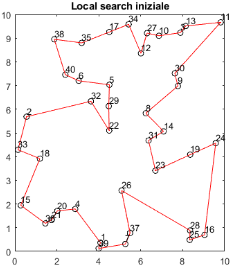
ILS-NN: 9.28%

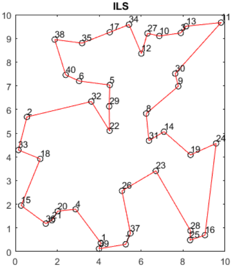
LS-NN: 8.68%

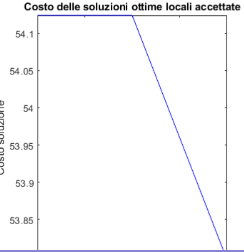
ILS-LS: 0.66%

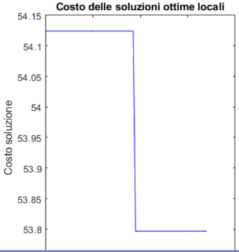
N=40: City insert certo - City swap random











NN: 62.540251

LS: 54.124242 - 7 it

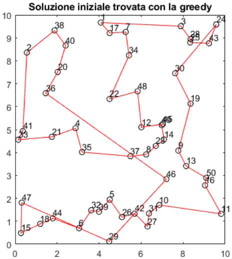
ILS: 53.796424 - 37° it - 2 acc

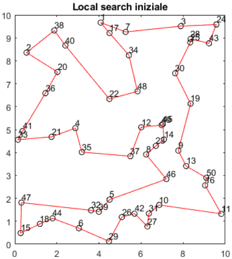
ILS-NN: 13.98%

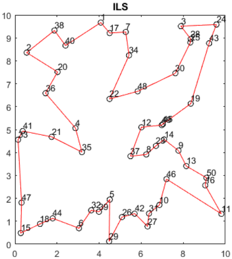
LS-NN: 13.45%

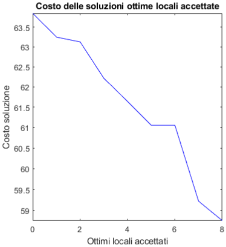
ILS-LS: 0.61%

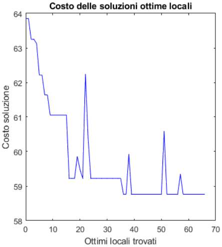
N=50: 2-opt - City insert random











NN: 70.712619

LS: 63.851890 - 12 it

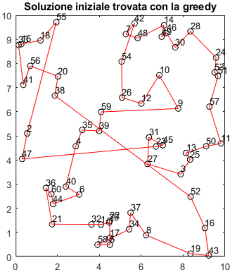
ILS: 58.763623 - 35° it - 8 acc.

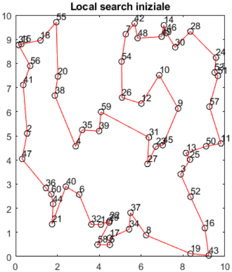
ILS-NN: 16.90%

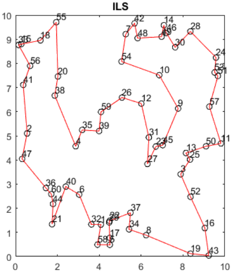
LS-NN: 9.70%

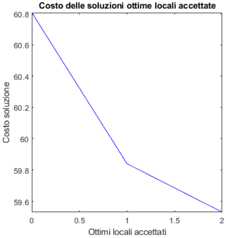
ILS-LS: 7.97%

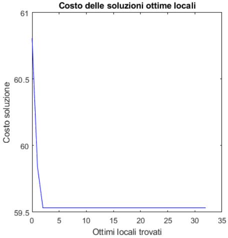
N=60: 2-opt - 3-opt











NN: 76.982338

LS: 60.808508 - 18 it

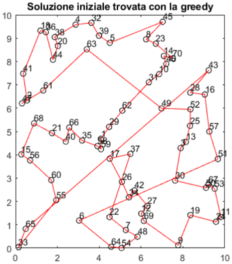
ILS: 59.532702 - 1° it - 2 acc.

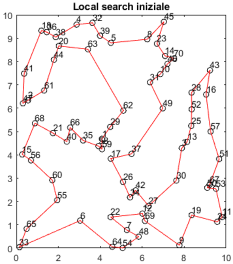
ILS-NN: 22.67%

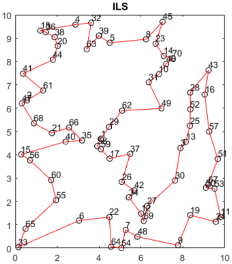
LS-NN: 21.01%

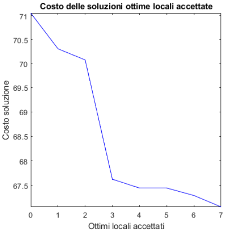
ILS-LS: 2.10%

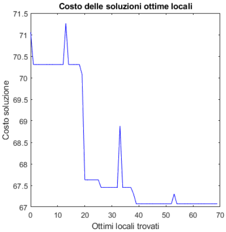
N=70: 2-opt - City insert random











NN: 88.948289

LS: 71.056873 - 14 it

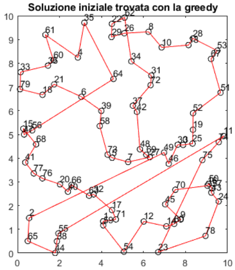
ILS: 67.074273 - 38° it - 7 acc.

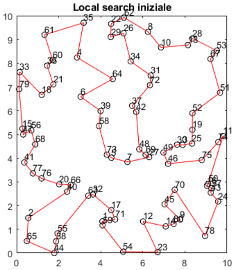
ILS-NN: 24.59%

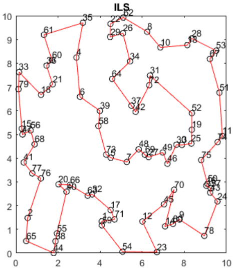
LS-NN: 20.11%

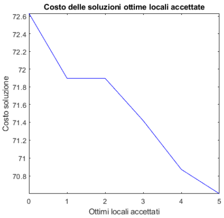
ILS-LS: 5.60%

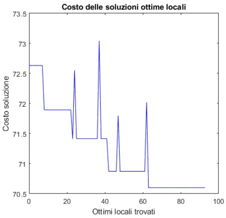
N=80: 2-opt - City insert random











NN: 90.041424

LS: 72.632481 - 22 it

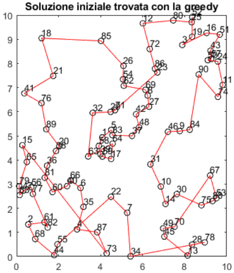
ILS: 70.602463 - 62° it - 5 acc.

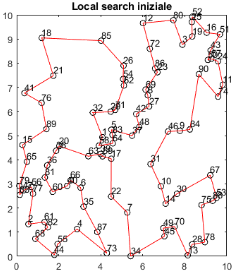
ILS-NN: 21.59%

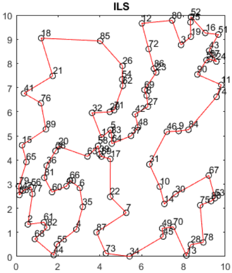
LS-NN: 19.33%

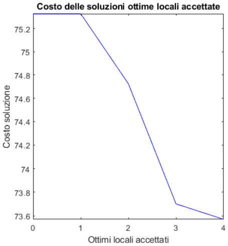
ILS-LS: 2.79%

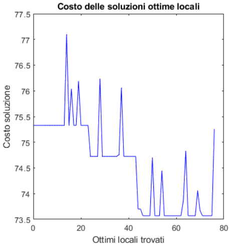
N=90: 2-opt - City insert random











NN: 85.465917

LS: 75.327795 - 19 it

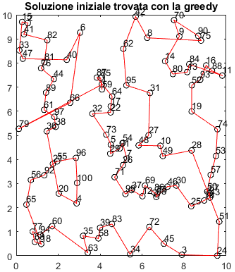
ILS: 73.569769 - 45° it - 4 acc.

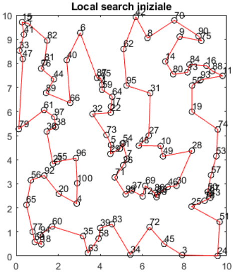
ILS-NN: 13.92%

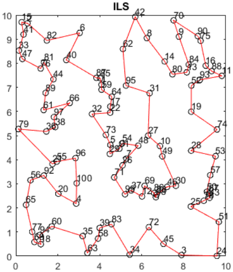
LS-NN: 11.86%

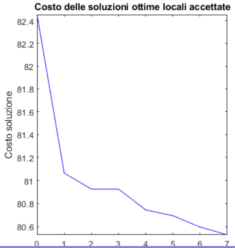
ILS-LS: 2.33%

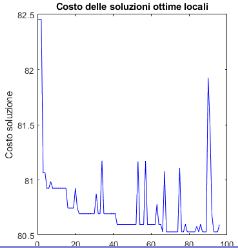
N=100: 2-opt - City insert random











NN: 91.378465

LS: 82.454921 - 14 it

ILS: 80.528943 - 65° it - 7 acc.

ILS-NN: 11.87%

LS-NN: 9.77%

ILS-LS: 2.34%