IT+Robotics | Spin-off of University of Padua

# Titolo

## Sottotitolo

# Indice

Metodi e Modelli per l'Ottimizzazione Combinatoria Progetto d'esame, aa 2017-2018 Anna Bonaldo

# 1 Modellazione

Il problema da affrontare può essere rappresentato in modo generico come un insieme di punti in uno spazio bidimensionale. In particolare, è modellabile come un TSP asimmetrico, in quanto è sufficiente utilizzare spigoli (e non archi orientati) per rappresentare il percorso eseguito per la foratura di una scheda. La direzione in cui viene eseguito il percorso non è influente sul tempo impiegato per percorrere l'intero tragitto.

Le istanze del problema su cui vengono testati gli algoritmi (esatti ed euristici) sono di più tipologie:

- Punti generati randomicamente in uno spazio di raggio determinato. Questo insieme di istanze del problema rappresenta un caso molto generico, e compare spesso anche in letteratura, per svariate tipologie di analisi sugli algoritmi per la risoluzione del TSP. In particolare, vengono utilizzate più misure per il raggio in cui generare casualmente i punti. In seguito una tabella che elenca tutte le possibilità esaminate.

  Dataset RANDN raggio uguale alla dimensione del problema.

  CONSTRAND50 Raggio costante uguale a 50

  CONSTRAND5000 Raggio costante uguale a 5000

  GRI N punti disposti a griglia di lato $\sqrt{N}$ equi distanziati.

| N punti | RANDN | CONSTRAND50 | CONSTRAND5000 | GRID |
|---|---|---|---|---|
| 10 punti | Raggio 10 | Raggio 50 | Raggio 5000 | Griglia di 10 punti |
| 50 punti | Raggio 50 | Raggio 50 | Raggio 5000 | Griglia di 50 punti |
| 100 punti | Raggio 100 | Raggio 50 | Raggio 5000 | Griglia di 100 punti |
| 200 punti | Raggio 200 | Raggio 50 | Raggio 5000 | Griglia di 200 punti |

- Punti posizionati "a griglia", tra loro uniformemente distanziati. Questo insieme di istanze del problema, invece, è probabilmente più aderente al problema in questione. E' infatti probabile che nel caso in esame i punti in cui eseguire la foratura non siano sparsi casualmente sulla superficie dell'oggetto.

Le varie tipologie di dataset utilizzati sono state scelte per confrontare gli effetti della morfologia del problema sulle prestazioni dell'algoritmo utilizzato per la risoluzione.

## 1.1 Struttura del progetto

Il progetto include:

- Una prima parte realizzata con l'utilizzo del framework **CPlex Studio**, utilizzata per la risoluzione delle istanze di TSP con metodi esatti.

- Una seconda parte, realizzata in **c++, in ambiente Windows 10** (Visual Studio 2017) per la risoluzione delle istanze del TSP con metodi euristici. Il programma realizzato utilizza la libreria OpenCV per rappresentare graficamente (e salvare su file) le soluzioni calcolate delle istanze del problema.

# 2 Implementazione dei metodi euristici per la soluzione del TSP

I metodi euristici implementati sono i seguenti:

- **Greedy Nearest-Neighbour** Ad ogni passo di esecuzione sceglie come nuovo passo del cammino il punto più vicino tra quelli disponibili. Viene mantenuta una lista sei nodi non ancora inclusi nel cammino. L'insieme dei nodi "liberi", ovvero non inclusi nel cammino, è implementata attraverso la struttura datdella libreria standard c++. Quest'ultima permette di verificare in tempo $O(\log n)$ se un nodo è già stato inserito nel cammino. Poiché ad ogni passo è necessario scegliere il nodo a distanza minima tra i disponibili, questa implementazione ha un tempo di esecuzione che è al più $O(n! \log(n))$. E' possibile realizzare implementazioni, un po' più complesse ma più efficienti, utilizzando strutture dati più complesse.
- **Simulated Annealing**
- **Tabu Search**

### 2.0.1 Ottimizzazione delle soluzioni esistenti

Algoritmo di ottimizzazione di un percorso basato su euristica 2-opt L'implementazione verifica per ogni coppia di archi nel cammino corrente se tali archi sono tra loro secati. In se la soluzione ottenuta eliminando l'incrocio tra due archi nel percorso è migliore della corrente, la nuova soluzione prende il posto di quella attuale. Poiché l'algoritmo deve iterare su tutte le coppie di archi, la complessità del calcolo è di O(n2) . I test per verificare se due archi sono secanti occupano un tempo O(1), utilizzando semplici nozioni geometriche per verificare la presenza di intersezioni.

# 3 Dataset

Tutti i dataset utilizzati sono stati prodotti utilizzando Cplex Studio.
Questi, una volta generati, vengono salvati su file e riutilizzati da tutti gli algoritmi implementati. In questo modo si assicura che i risultati possano essere tra loro confrontabili.

Per quanto riguarda il dataset GRID, lo script realizzato produce una sola istanza del problema per ogni dimensione considerata(10, 50, 100 e 200 punti). Questa scelta è motivata dal fatto che la procedura di realizzazione di una griglia è deterministica.

Per quanto riguarda i restanti dataset: **RAND N** , **CONSTRAND50**, e **CONSTRAND5000** vengono **prodotte 10 diverse istanze del problema** per ogni dimensione considerata. Ogni generazione di un'istanza per un certo dataset utilizza un diverso seed, per inizializzare la funzione rand().In questo modo si ha una discreta rappresentazione della variabilità del problema per il dataset considerato.

# 4 Analisi dei dati

## 4.1 Soluzione esatta del TSP

L'implementazione dell'algoritmo esatto del TSP realizzata con CPLEX Studio è stata utilizzata per risolvere il problema su tutti i dataset utilizzati. Le soluzioni ottenute sono state poi utilizzare come punto di riferimento per valutare gli algoritmi euristici.

I **tempi medi** per il calcolo della soluzione con metodi esatti sono riportati in tabella TODO, con la relativa deviazione standard.

Dai risultati si evince che la densità di punti generati nello spazio bidimensionale non influenza la varianza del tempo di calcolo. Al contrario se con il numero di punti del problema diminuisce anche la relativa densità nello spazio (e quindi aumenta il raggio entro cui vengono generati i punti del problema), la deviazione standard aumenta in modo considerevole. Se ne deriva che l'algoritmo utilizzato ha più difficlotà nel trovare la soluzione. Per quanto riguarda il problema con 200 punti, il programma fallisce con errore per insufficienza di memoria disponibile. Non si dispone quindi di una soluzione esatta. Per i successivi calcoli che vengono valutati in rapporto alla soluzione esatta, si utilizza in sostituzione della soluzione ottima (non disponibile) la migliore soluzione ottenuta con algoritmi euristici.

Il tempo di calcolo per ottenere la soluzione esatta è notoriamente esponenziale sulla dimensione del problema. Il grafico riporta i dati medi per ogni dataset. Si evidenzia che il raggio entro cui sono distribuiti i punti del problema influenza in modo negativo i tempi di calcolo della soluzione. In particolar modo, il caso in cui il raggio dello spazio dei punti sia ugiale ad N (numero di punti) appare particolarmente critico.

## 4.2 TSP Greedy Nearest Neighbors

L'euristica costruttiva Nearest Neighbours rappresenta un metodo molto veloce per il calcolo della soluzione del TSP. Tuttavia la qualità della soluzione non è generalmente buona nel caso generico. Per quanto riguarda casi particolari, come ad esempio per il dataset GRID, in cui i punti sono disposti regolarmente, si riescono ad ottenere soluzioni buone anche con il solo utilizzo di questa euristica. Si veda un esempio in figura. TODO

E' tuttavia possibile migliorare i risultati ottenuti utilizzando un'euristica *2-Opt* che elimina eventuali spigoli che si intersecano nel cammino calcolato. Nell'esperimento realizzato, si vede che l'utilizzo di un'ottimizzazione *2-opt* su una soluzione precalcolata con Greedy Nearest Neighbours porta a buone soluzioni in tempi inferiori ad euristiche come Tabu Search e Simulated Annealing 1.

Con questo metodo, per i dataset utilizzati, in generale, si ottengono soluzioni migliori in media del 5% rispetto alla soluzione con solo algoritmo Greedy per problemi di dimensione maggiore o uguale a 50 nodi. Per problemi di dimensione inferiore, il guadagno in termine di bontà della soluzione è tra il 5 ed il 15%.

Essendo la procedura di ottimizzazione *2-Opt* $O(n^2)$, il tempo di calcolo in rapporto al precedente aumenta in modo considerevole (fino al 20000%). Tuttavia rimane inferiore a quello necessario ad altri algoritmi (Simluated Annealing) per produrre soluzioni peggiori. TODO

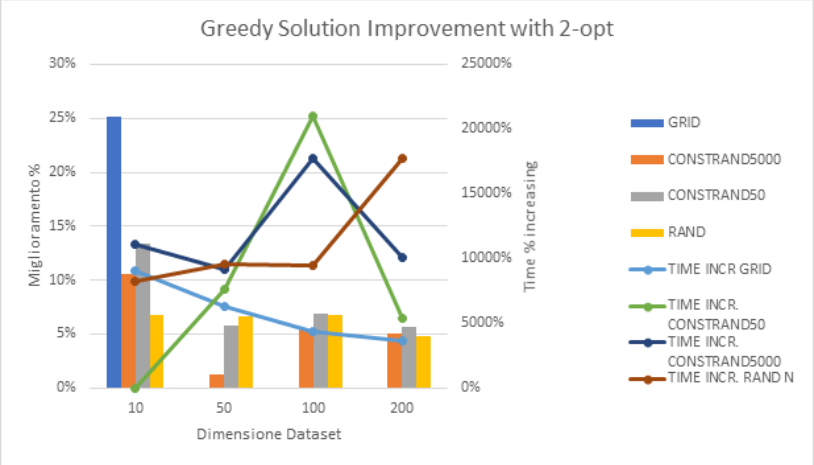


Figura 1

# 5 Compilazione del progetto