|  |  |
| --- | --- |
| Traveling Salesman Problem -Coppa Algoritmi | |
|  | |
| Studente/i | Relatore |
| Gattinoni Federico | - |
| Correlatore |
| Marco Cinus |
| Committente |
| Luca Gambardella |
| Corso di laurea | Modulo |
| Bachelor in Ingegneria Informatica | Algoritmi 2018 |
| Anno |  |
| 2018 |  |

|  |  |
| --- | --- |
| 02-05-2018 |  |
| Problema |  |

Il problema sottoposto è quello del commesso viaggiatore. Si tratta di un problema di ottimizzazione combinatoria appartenente alla classe dei problemi NP-Hard e consiste nel, dato un insieme di città ed una funzione di distanza, trovare il percorso minimo che visita tutte le città una sola volta.

Gli algoritmi implementati verranno poi testati con un insieme di file tps avendo numero di città e dislocazione differente che caratterizzano ogni problema. Di questi problemi vengono forniti le lunghezze dei percorsi ottimali. Lo scopo del progetto è quello di trovare il percorso che si avvicini di più all’ottimale riducendo così il più possibile la percentuale di errore.

L’esecuzione di ogni problema deve avvenire all’intorno di un intervallo di tempoo fissato a 3 minuti.

Algoritmi Implementati

Random

L’algoritmo Random è un algoritmo costruttivo che prende l’elenco delle città e produce un tour scegliendo l’ordine di visita delle città utilizzando una scelta delle città prettamente casuale generando numeri da 0 al numero di città del problema in questione.

Questo algoritmo è particolarmente utile in altri algoritmi: ad esempio negli algoritmi genetici dove servono molti tour ammissibili che occupino il minor tempo possibile. Avendo scelto di implementare altri algoritmi in cui questo procedimento non risultava molto utile e per questo non è stato poi utilizzato.

Nearest Neighbour

L’algoritmo costruttivo che invece è stato usato è il Nearest Neighbour.

Questo algoritmo permette di creare tour che, partendo da una città casuale prende come prossima città del tour quella più vicina a sé. Ogni volta che la città più vicina è già stata visitata si passa a quelle meno vicine.

Proprio perché non sempre le città più vicine possono essere ancora visitate i primi archi saranno i più corti e poi man mano che le città già visitate aumentano le distanze tra i nodi aumentano. Questo algoritmo è usato per generare il primo tour ammissibile.

2-opt

L’algoritmo di Local Search implementato è il 2-Opt. SI tratta di un algoritmo che partendo da un tour ammissibile, scambia due archi alla volta e verifica se lo scambio migliora la soluzione oppure no.

Ci sono due modi di implementare questo algoritmo: uno in cui ci si ferma al primo scambio che migliora la soluzione, l’altro modo invece consiste nel provare tutti i possibili scambi memorizzando lo scambio migliore.

Quello implementato è il secondo modo (best improvement).

Per evitare di fare continui scambi vengono memorizzati solamente gli archi che permettono il miglioramento maggiore e lo scambio degli archi migliori avviene solamente dopo aver verificato tutti i possibili scambi.

Questo algoritmo non permette però di uscire da un minimo locale in quanto si basa solo su informazioni locali.

Proprio per questo motivo questo algoritmo e il Nearest Neighbour non permettono di trovare la soluzione ottima per nessuno dei problemi proposti e per questo è usato all’interno di un algoritmo meta-euristico che permette di risolvere il problema di questa implementazione.

L’algoritmo usato è l’Ant Colony System (ACS). Il 2-Opt è usato all’interno dell’ACS per migliorare le soluzioni generate.

Particolarità implementative rispetto allo standard sta nel fatto che non vengono verificati tutti gli scambi possibili in quanto l’indice del secondo nodo non parte mai da 0 ma parte da due nodi successivi al primo indice.

Ant Colony System (ACS)

L’Ant Colony System è un algoritmo meta-euristico che permette di uscire da un minimo locale permettendo di accettare anche soluzioni che peggiorano quella corrente oppure esplorare stocasticamente un sottoinsieme di soluzioni vicine.

Utilizza un numero arbitrario da definire di Ant che generano ognuna un tour ammissibile e utilizza un costo dinamico, detto pheromone che insieme alla distanza permettono di modificare dinamicamente le scelte fatte dalle formiche per quanto riguarda la nuova città da visitare.

Ogni volta che una formica aggiunge una città al suo tour viene modificata la quantità di pheromone sull’arco che porta dall’ultima città visitata a quella nuova.

Successivamente, una volta che tutte le formiche hanno creato il loro tour ammissibile viene ulteriormente modificato il pheromone, però questa volta la modifica avviene lungo tutto il tour della formica che ha generato il percorso più breve tra tutte.

Il tuor migliore è però scelto dopo aver migliorato il percorso di ogni formica utilizzando il 2-Opt, questo permette dunque di avere soluzioni migliori.

Differenze implementative rispetto allo standard riguardano solamente la scelta del percorso migliore tra i tour di ogni formica. Più precisamente come già detto la scelta è fatta solamente dopo aver applicato l’algoritmo di ottimizzazione locale a tutti i tour. Cosi facendo si ottengono migliori soluzioni in quanto il 2-Opt migliora leggermente il tour.

Esecuzione del programma

**Piattaforma**

Il PC usato per testare i problemi è un PC fisso con Sistema Operativo Windows 7 fornitoci.

**Compilazione ed esecuzione dei problemi**

Per compilare ed eseguire è necessario solamente eseguire il comando:

java -jar CoppaAlgoritmi.jar ALGO\_cup\_2018\_problems/tourFile seed

dove tourFile rappresenta il fil con estensione \*.tsp che contiene l’elenco delle città con le relative coordinate.

seed invece rappresenta il seme per inizializzare il random, questo parametro è opzionale. Nel caso questo parametro venga omesso il seed sarà scelto autonomamente dall’applicazione.

Due esempi di esecuzione sono di seguito rappresentati:

*java -jar CoppaAlgoritmi.jar ALGO\_cup\_2018\_problems/d198.tsp 1525369930397*

*java -jar CoppaAlgoritmi.jar ALGO\_cup\_2018\_problems/u1060.tsp*

Risultati

I risultati migliori ottenuti per ogni problema sono riportati nel file *ALGO\_cup\_2018\_GattinoniFederico.xls*

Conclusioni

Per facilitare ed automatizzare l’esecuzione dei problemi è stato aggiunto un piccolo programmino scritto in Batch Scripting in cui i problemi vengono eseguiti uno dopo l’altro senza dover cambiare ogni volta la riga per lanciare il programma.

L’implementazione di questi algoritmi è stato molto utile per conoscere gli algoritmi per risolvere problemi riguardante il TSP.