Progetto di Fisica Statistica e Teoria dell'Informazione

Benedetta Picano, Stefano Vannacci

20 agosto 2014

Indice

Capitolo 1

Introduzione

Il progetto in questione consiste nell'implementazione e confronto di tre diverse versioni dell'algoritmo del Commesso Viaggiatore.
Il commesso viaggiatore (conosciuto in inglese come *Travelling Salesman Problem o TSP*) è un problema di teoria dei grafi. Il problema che l'algoritmo vuole risolvere può essere riassunto da questo esempio:

data una rete di città, connesse tramite delle strade, trovare il percorso di minore lunghezza che un commesso viaggiatore deve seguire per visitare tutte le città una e una sola volta

In teoria dei grafi può essere tradotto come dato un grafo completo pesato, trovare il cammino di costo minore visitando tutti i nodi una sola volta e tornando al nodo di partenza. La rete di città può essere rappresentata come un grafo in cui le città sono i nodi, le strade gli archi e le distanze i pesi sugli archi

Al momento non esistono algoritmi efficienti per la risoluzione di tale problematica. L'unico metodo risolutivo consiste nell'elaborazione di tutti i possibili cammini sul grafo per la successiva scelta di quello migliore. È facile immaginarsi che la complessità dell'operazione rende tale metodo impraticabile per grafi di dimensioni molto grandi e quindi simili ai problemi reali.
La complessità di tale metodo è infatti $\mathcal{O}(n!)$ Il problema è anche catalogato come NP-Completo. Per risolvere problemi di tipo NP-Completo si possono seguire tre strade:

- progettare algoritmi per trovare la soluzione esatta, ragionevolmente veloci solo per problemi con un numero di città relativamente basso;
- progettare algoritmi euristici, cioè algoritmi che producono soluzioni probabilmente buone, ma impossibili da provare essere ottimali;

- trovare un caso specifico del problema (sotto problema) per il quale sia possibile o una soluzione esatta o un'euristica migliore.

Nel nostro caso abbiamo implementato due algoritmi che trovano la soluzione esatta, con uno che trova una soluzione approssimata mediante l'utilizzo del linguaggio Python.

Capitolo 2

Try All Paths

Questo algoritmo è l'implementazione convenzionale del problema del commesso viaggiatore e può essere riassunto come di seguito:

1. Genera una lista di tuple del tipo (x, y) rappresentanti i nodi del grafo, ovvero le città.

2. Per tutti i percorsi possibili valuta qual'è il percorso più breve. Il peso degli archi del grafo è dato dalla distanza che c'è tra due punti nel grafo.

3. Stampa a video il percorso più breve.

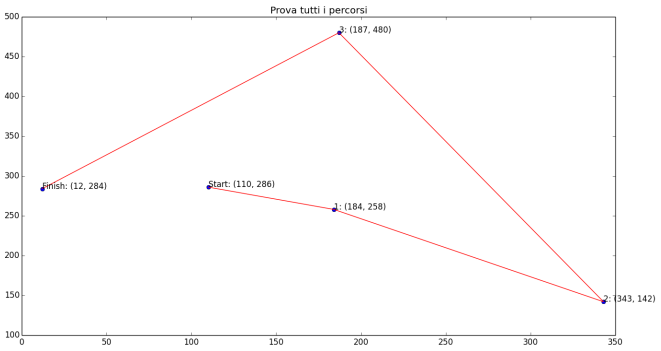

Figura 2.1: Un possibile output

La complessità di tale algoritmo è $\mathcal{O}(n!)$

Capitolo 3

Try Non Redundant Paths

Variante del precedente algoritmo, dove vengono presi in considerazione tutti i percorsi che partono dal solito nodo, quindi dalla solita città. Si impone quindi un vincolo in partenza:

1. Genera una lista di tuple del tipo (x, y) rappresentanti i nodi del grafo, ovvero le città.
2. Dai percorsi generati vengono scelti tutti i percorsi che partono da un solito nodo scelto dalla funzione. Gli altri vengono eliminati.
3. Per tutti i percorsi possibili valuta qual'è il percorso più breve. Il peso degli archi del grafo è dato dalla distanza che c'è tra due punti nel grafo.
4. Stampa a video il percorso più breve.

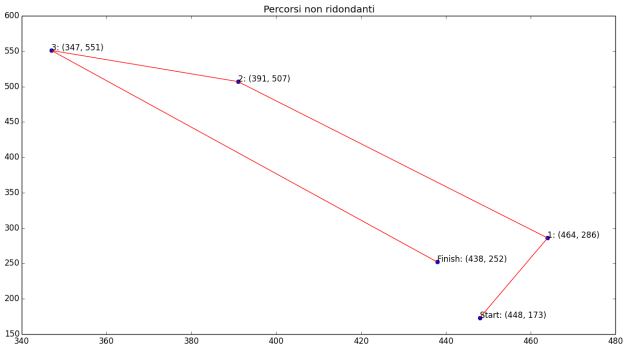

Figura 3.1: Un possibile output

La complessità di tale algoritmo è $\mathcal{O}((n-1)!)$

Capitolo 4

Confronto tra Try All Paths e Try Non Redundant Paths

Di seguito è riportato un grafico che compara i due algoritmi sopra descritti: Come si può notare, impiegano svariati minuti a calcolare una decina di

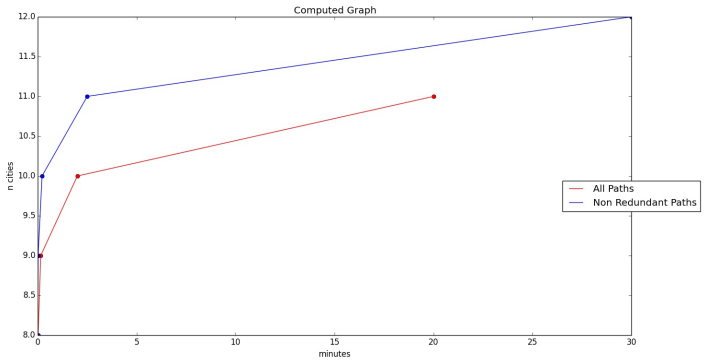

Figura 4.1: Confronto tra gli algoritmi *Try All Paths* e *Try Non Redundant Paths*

punti, il che li rende non utilizzabili per problemi reali. In nostro soccorso arrivano algoritmi di approssimazione, che calcolano una soluzione buona ma non del tutto esatta.

Capitolo 5

Greedy Nearest Neighbor algorithm

Variante che approssima la soluzione dell'algoritmo del commesso viaggiatore. A partire da un punto, ad ogni passo aggiunge al percorso il punto con distanza più breve dall'attuale che non è stato ancora visitato.
L'algoritmo può essere riassunto come segue:

1. Genera una lista di tuple del tipo (x, y) rappresentanti i nodi del grafo, ovvero le città.
2. Viene scelto un punto di partenza e generata una lista di nodi non visitati.
3. Ad ogni iterazione calcola la distanza tra il nodo corrente e quelli presenti tra i non visitati. Salva nel percorso finale il nodo con distanza più breve da quello corrente e lo toglie dalla lista dei nodi non visitati.
4. Stampa infine il percorso approssimato.

La velocità di computazione per calcolare più di 100 punti è inferiore al secondo.

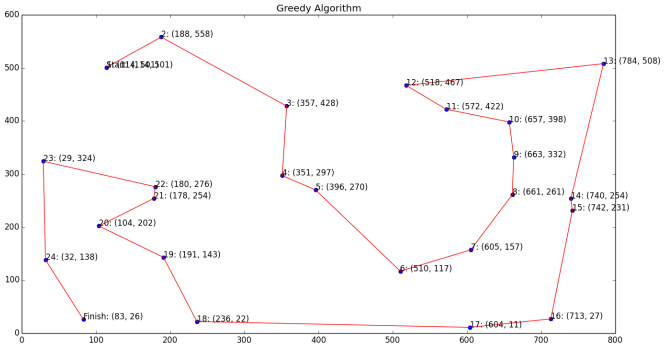

Figura 5.1: Un possibile output con 25 nodi

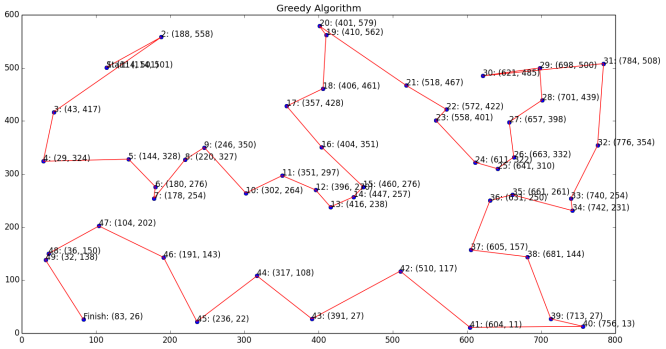

Figura 5.2: Un possibile output con 50 nodi

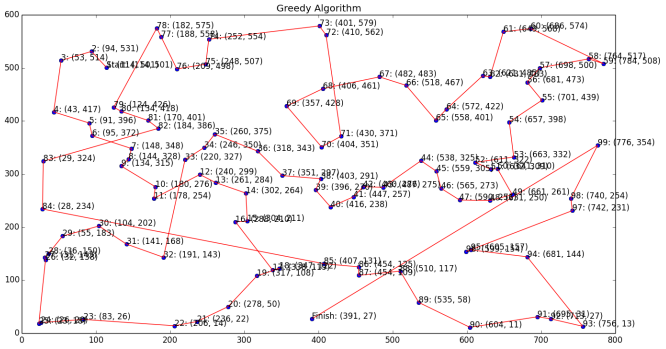

Figura 5.3: Un possibile output con 100 nodi