Algoritmo di Branch-&-Bound eseguito in parallelo per risolvere il TSP problem

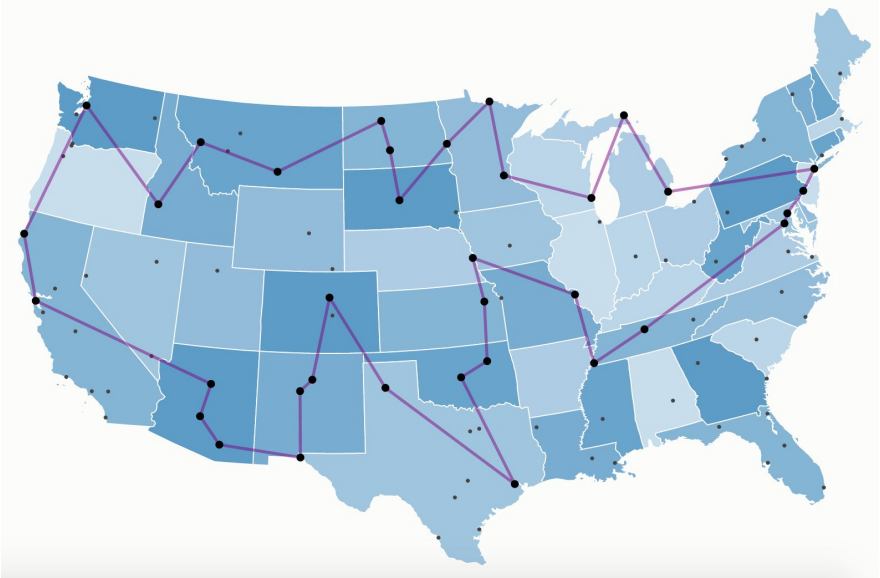
Università di Bologna
Laurea Magistrale in Informatica
Corso di Scalable and Cloud Programming
A.A. 2018-2019

Lorenzo Massimiliani

Lorenzo Vainigli

Problema del Commesso Viaggiatore

Date n città e le distanze tra tutte le coppie di esse, si vuole determinare un percorso ciclico che includa ogni città una e una sola volta percorrendo complessivamente la minor distanza possibile.



Complessità

È stato dimostrato che TSP è NP-arduo.
Si possono utilizzare diversi algoritmi per la risoluzione di questo problema:

Tipo di algoritmo	**Complessità**
Enumerazione totale	$O(n!)$
Programmazione dinamica	$O(n^2 2^n)$
Branch & bound	$O(n!)$

Nonostante l'algoritmo di programmazione dinamica sia il migliore nel caso pessimo, la tecnica branch and bound può risultare, nel caso medio, più efficiente.

TSP Branch and bound

L’algoritmo prende in input una matrice nodi-nodi delle distanze (che rappresenta la distanza di ogni nodo con gli altri).

Viene scelto un arco (con una euristica) e generate due configurazioni che corrispondono all’inclusione e all’esclusione di tale arco dalla soluzione.

In questo modo si forma un albero e grazie ad una funzione LB (lower bound) l’algoritmo riesce a determinare preventivamente se un sottoalbero può contenere la soluzione ottima oppure no e, in caso negativo, si evita di esplorarlo.

Vengono usate tre procedure per calcolare il LB e per trovare la soluzione ottima:

- Riduzione della matrice
- Inclusione di un arco
- Esclusione di un arco

Riduzione della matrice

Distanza (km)	Bologna	Roma	Firenze	Palermo
Bologna	∞	377	118	1272
Roma	377	∞	271	923
Firenze	118	271	∞	1173
Palermo	1272	923	1173	∞

Riduzione per righe

Distanza (km)	Bologna	Roma	Firenze	Palermo
Bologna	∞	359	0	1154
Roma	106	∞	0	652
Firenze	0	153	∞	1055
Palermo	349	0	250	∞

118

271

118

923

Riduzione per colonne

Distanza (km)	Bologna	Roma	Firenze	Palermo
Bologna	∞	359	0	502
Roma	106	∞	0	0
Firenze	0	153	∞	403
Palermo	349	0	250	∞

0 0 0 652

Lower bound:
118 + 271 + 118 + 923 + 652 = 2082

Inclusione ed esclusione archi

Inclusione dell'arco
Bologna - Roma

Distanza (km)	Bologna	Roma	Firenze	Palermo
Bologna	–	–	–	–
Roma	∞	–	0	0
Firenze	0	–	∞	403
Palermo	349	–	250	∞

Lower bound += distanza Bologna-Roma

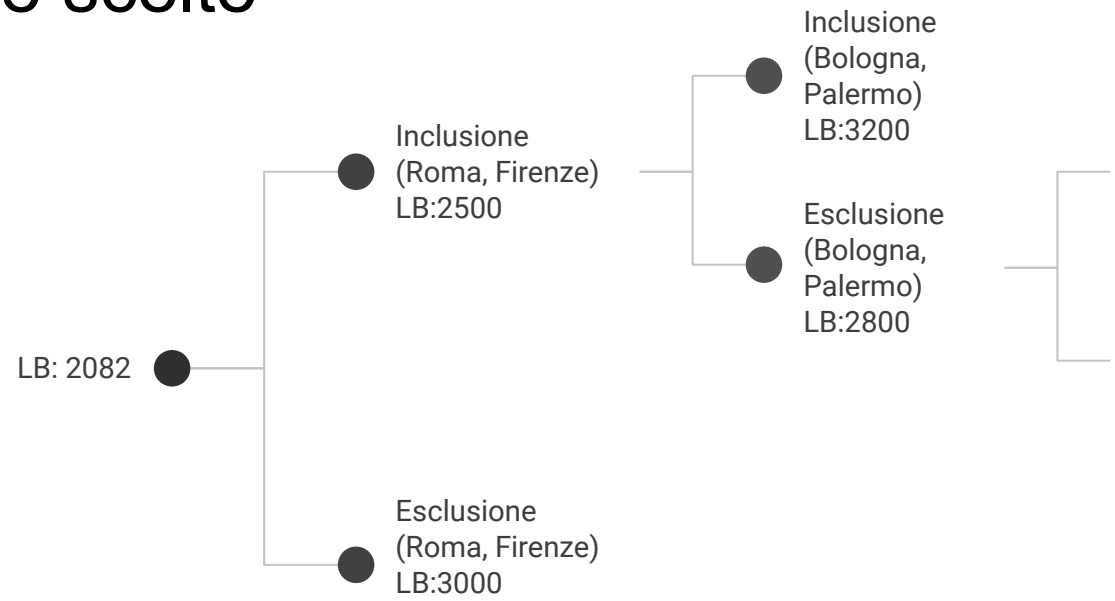
Riduzione della matrice

Esclusione dell'arco
Bologna - Roma

Distanza (km)	Bologna	Roma	Firenze	Palermo
Bologna	∞	∞	0	502
Roma	106	∞	0	0
Firenze	0	153	∞	403
Palermo	349	0	250	∞

Riduzione della matrice

Albero delle scelte



Si sceglie di espandere sempre il nodo con lower bound minore.
Si procede considerando l'inclusione e l'esclusione dell'arco, non ancora considerato, la cui esclusione massimizza il lower bound.
Giunti ad una foglia, la soluzione è data dal cammino radice-foglia appena percorso (che rappresenta un ciclo dato da tutti gli archi inclusi).

Implementazione in Scala

TSP branch and bound è stato implementato in linguaggio Scala utilizzando l’approccio della programmazione funzionale.

- funzioni anonime
- unzip
- filterKeys
- foreach
- toMap

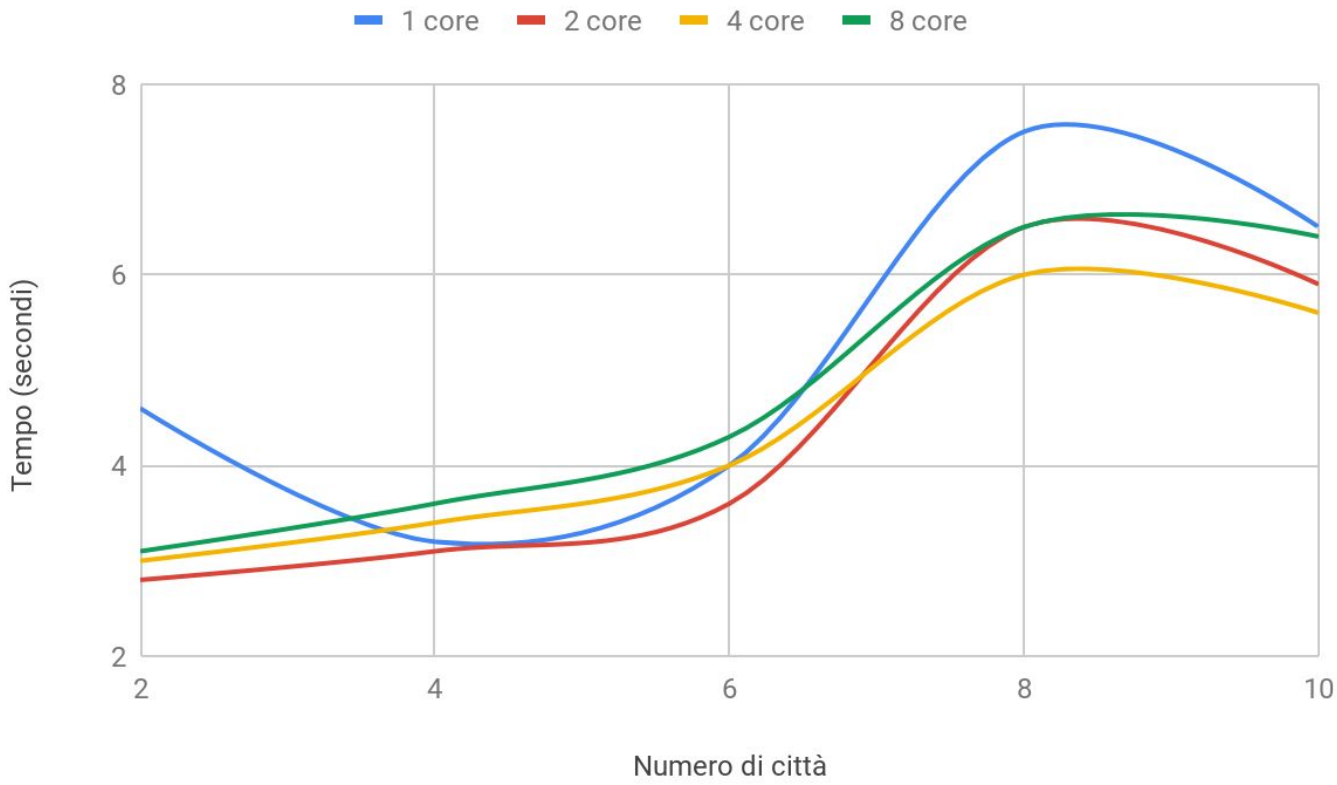
Parallelizzazione con Spark

- In una lista sono inserite tutte le configurazioni che corrispondono ai nodi dell’albero precedente.
- **Nel cluster di n unità di elaborazione vengono distribuite le n configurazioni con lower bound più basso. In modo che ad ogni unità venga assegnata una configurazione.**
- In parallelo ogni nodo del cluster determinerà quale arco è quello la cui esclusione massimizza il lower bound. E verrà calcolata la matrice corrispondente all’esclusione e all’inclusione di esso.
- Le 2n nuove configurazioni trovate verranno inserite nella lista principale.
- Si itera fino a quando non si trova un percorso radice-foglia, che corrisponderà alla soluzione.

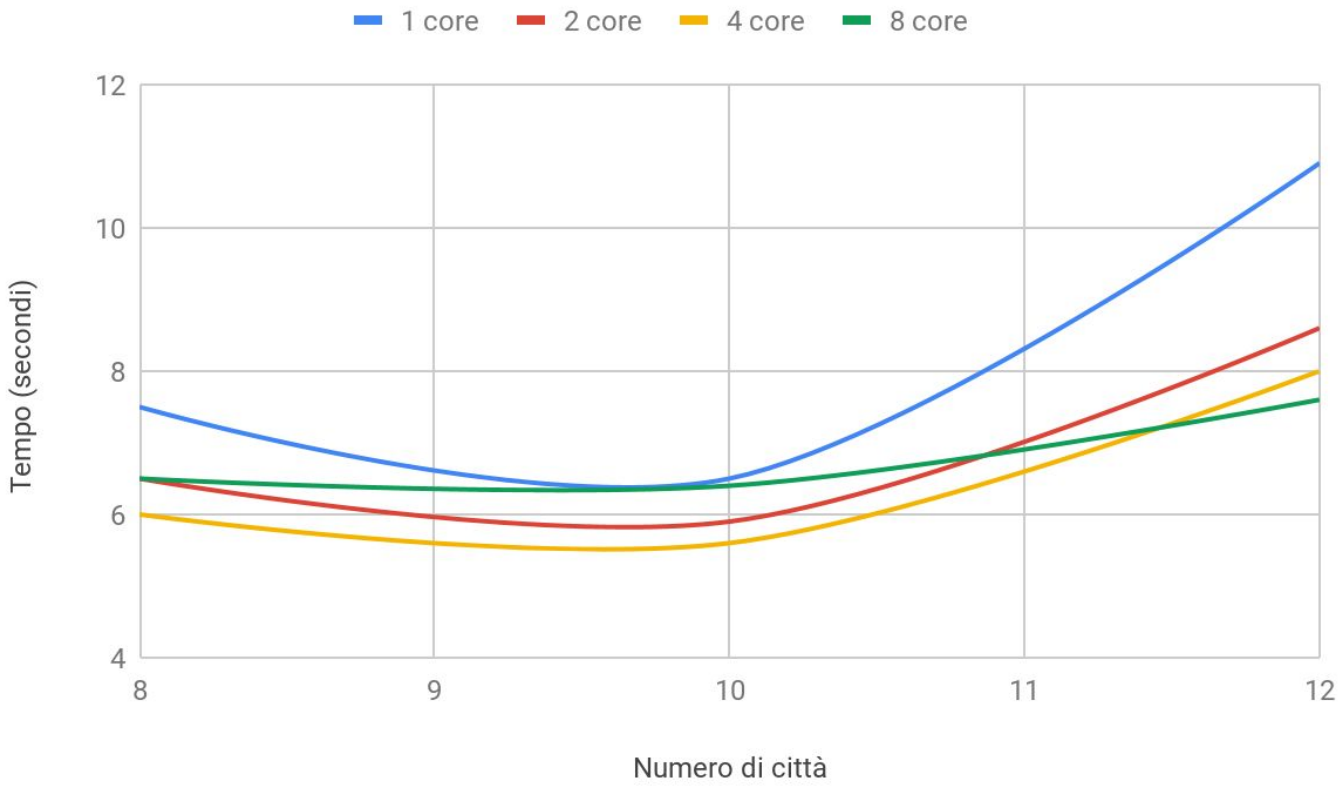
Amazon Web Services

- Il progetto è stato esportato nel formato JAR con l'utility sbt-assembly
- Il file JAR è caricato su AWS attraverso il servizio S3
- Il cluster è avviato utilizzando EMR
  Sono stati utilizzati nodi del tipo x3.large e sono stati creati cluster con 1,2,4 e 8 unità di calcolo.
- Attraverso l'applicazione Putty e utilizzando una chiave è stata fatta la connessione da terminale al cluster
- Il JAR è copiato nel cluster
- Utilizzando il comando *java -jar file.jar* viene eseguito il programma

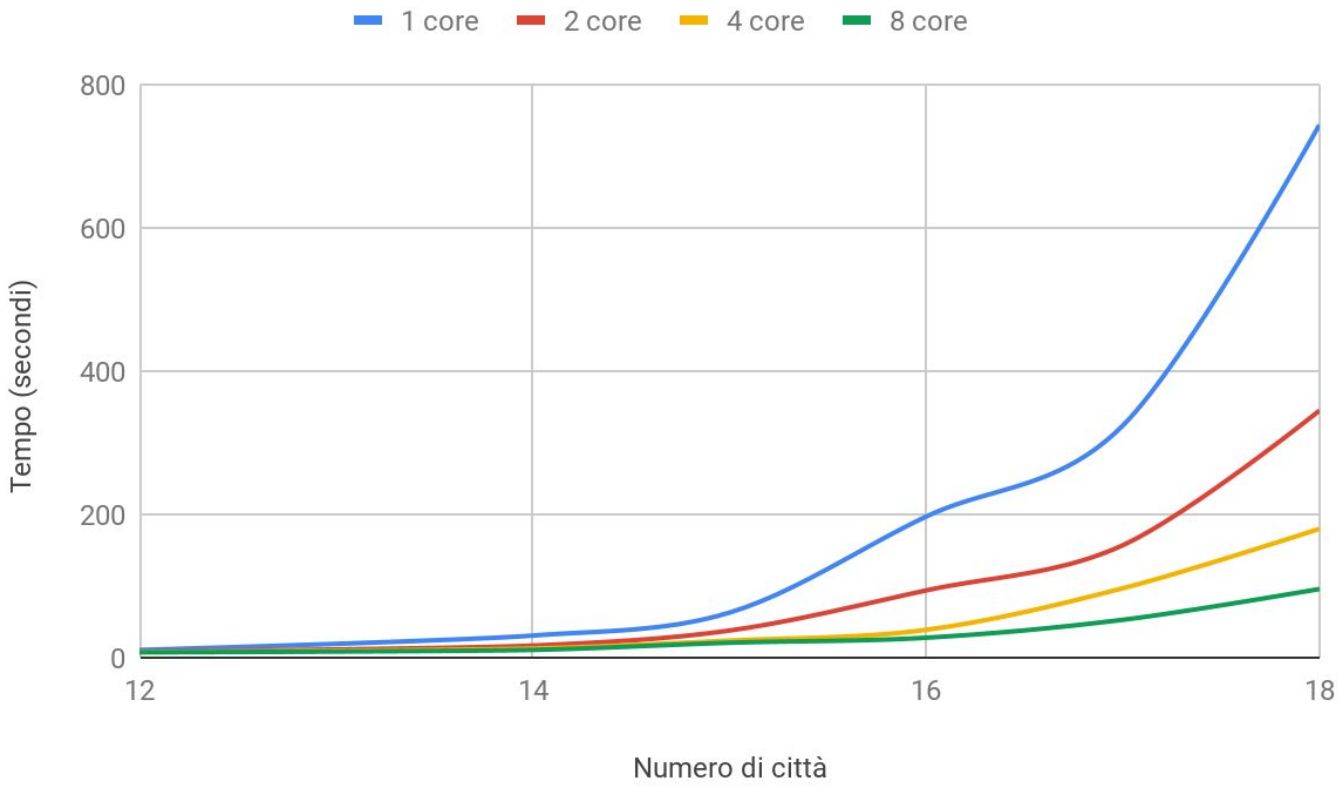
Risultati: tempo di calcolo per 2-10 città con 1-8 core



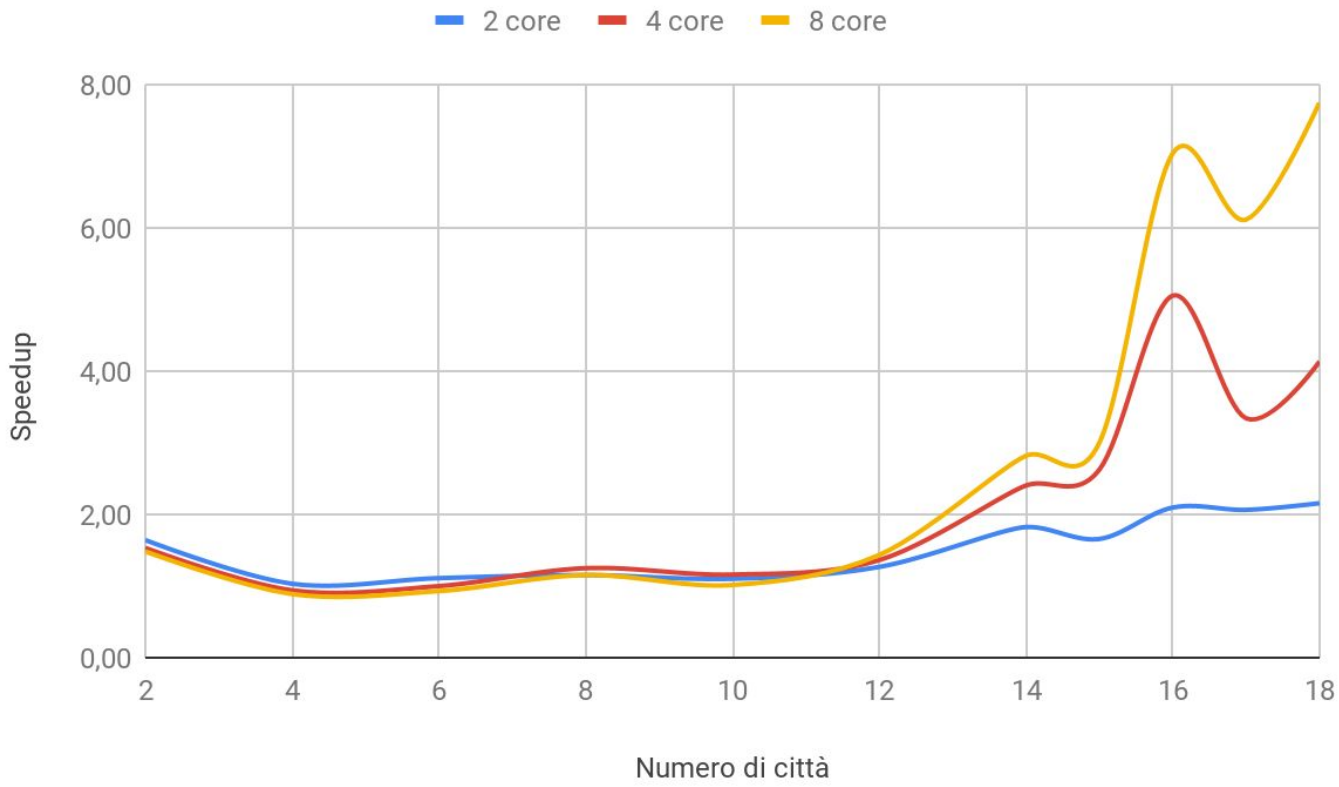
Risultati: tempo di calcolo per 8-12 città con 1-8 core



Risultati: tempo di calcolo per 12-18 città con 1-8 core



Risultati: speedup



Conclusioni

I risultati evidenziano chiaramente un vantaggio in termini di tempo computazionale parallelizzando l'algoritmo TSP branch & bound.

Ai fini del progetto abbiamo testato l'algoritmo su una quantità di dati irrisoria rispetto quello che accade con le applicazioni reali (es. pianificazione di itinerari per la consegna di merci).

Tuttavia già con 16-18 città si evidenzia uno speedup di 6-7 utilizzando 8 core, capace di crescere in modo molto significativo all'aumentare del numero delle città.