DIPARTIMENTO DI INGEGNERIA "ENZO FERRARI"

Automated Decision Making

TSP - Good and Bad Sequences

Maggiorelli Umberto - Pellegrini Daniele

Settembre 2021

Indice

Sommario

Implementazione di un algoritmo greedy le cui scelte delle sequenze per la risoluzione di problemi del TSP sono guidate da uno score risultante da un algoritmo di Machine Learning.

Introduzione

Questo elaborato ha come obiettivo *l'implementazione di un algoritmo greedy* che, guidato da un algoritmo di Machine Learning per valutare delle sequenze di nodi, sia in grado di risolvere i problemi del TSP. Il *Travelling Salesman Problem* (*TSP*), conosciuto in italiano come il Problema del Commesso Viaggiatore, consiste nell'individuare il percorso più breve che attraversi tutte le città (nodi di un grafo) una e una sola volta, ritornando al punto di partenza. Per conseguire l'obiettivo dell'elaborato è necessario utilizzare un algoritmo di Machine Learning che ci permetta di analizzare una sequenza di n nodi ($5 \leq n \leq 8$) e decida se questa sequenza è buona o cattiva. Con il termine 'buona' si fa riferimento a una sequenza che se sostituita all'interno della soluzione ottima genera un errore, sulla lunghezza complessiva minima, inferiore al 4%; viceversa vengono ritenute 'cattive' tutte quelle altre sequenze che generano un errore significativo. Una volta costruito il dataset e allenato l'algoritmo, questo viene utilizzato per guidare l'algoritmo greedy nella scelta delle sequenze da inserire all'interno della soluzione.

Capitolo 1

Costruzione del Dataset

Per la costruzione del dataset, poiché nella libreria TSPLIB [1] non sono presenti sufficienti problemi di 100 nodi, vengono creati 500 grafi costituiti da 100 nodi distribuiti uniformemente nello spazio e di coordinate di diversa grandezza. Su questi grafi viene risolto il problema del TSP utilizzando l'algoritmo LKH, il quale ci permette di ottenere il percorso ottimale per ciascun grafo. Per la costruzione del dataset, si è deciso di prelevare da ogni grafo 4 diversi gruppi di sequenze:

- Sequenze Ottime
- Sequenze Buone
- Sequenze Cattive
- Sequenze Pessime

Ciascun dataset viene costruito in base alla lunghezza della sequenza che si vuole utilizzare nell'algoritmo. Le sequenze definite *ottime* vengono prese direttamente dalla soluzione ottimale conosciuta. Come sequenze *buone* vengono prese quelle nelle quali è stato invertito l'ordine di una coppia di nodi rispetto alla soluzione ottimale, e che pertanto non presentano un errore significativo sulla soluzione. Contrariamente a queste, le sequenze *cattive* sono quelle sequenze su cui vengono effettuati diversi scambi di nodi rispetto alla sequenza corrispondente nella soluzione, e che quindi generano un errore significativo sul percorso conosciuto. Per ultime, come sequenze *pessime*, vengono prese delle sequenze appartenenti a soluzioni generate in maniera casuale. Prendendo lo stesso numero di record per ciascun gruppo di sequenze il dataset risulta bilanciato: le sequenze ottime e quelle buone sono classificate come 1, mentre quelle cattive e quelle pessime come 0.

Capitolo 2

Implementazione dell'Algoritmo Greedy

2.1 Algoritmi di Machine Learning

Per addestrare gli algoritmi di ML si è deciso di dividere il dataset in due parti, nelle quali i gruppi di sequenze sono divisi equamente: l'80% viene impiegato per la fase di training, mentre il 20% per la fase di test. Gli algoritmi di Machine Learning importati da *sklearn* sono:

- AdaBoost
- Decision Tree
- Logistic Regression
- K-Nearest Neighbors
- Naive Bayes
- Random Forest

Per verificare l'accuratezza degli algoritmi nell'individuare sequenze buone o cattive, questi vengono allenati su un dataset diverso per ciascuna sequenza, e testati con delle soluzioni di problemi del TSP create appositamente. Dopo una prima analisi mirata a riconoscere poche sequenze appartenenti a problemi diversi, l'algoritmo KNN risulta complessivamente il migliore, con performance molto simili registrate anche per il Random Forest. Decision Tree e AdaBoost risultano discreti, mentre per quanto riguarda gli algoritmi Logistic Regression e Naive Bayes sono stati registrati i risultati peggiori. Queste prime analisi tuttavia non sono sufficienti a giudicare l'efficacia di un algoritmo: questo infatti deve essere utilizzato per guidare le scelte di un greedy, e quindi testare più sequenze per costruire una soluzione, mentre questa prima analisi è mirata al riconoscimento di sequenze anche sconnesse tra loro, per lo più appartenenti a problemi diversi.

2.2 Funzionamento dell'Algoritmo

L'algoritmo greedy crea una sequenza di n nodi che viene passata per la valutazione all'algoritmo di Machine Learning scelto. La sequenza viene costruita prendendo di volta in volta un nodo casuale nel vicinato dell'ultimo nodo inserito nella stessa. Il vicinato è costituito dal 5% dei nodi più vicini all'ultimo inserito nella sequenza, tra quelli rimasti da inserire all'interno della soluzione. Il nodo di partenza di default è il nodo 1. Una volta costruita la sequenza, si creano tutte le possibili permutazioni dei nodi mantenendo invariato quello di partenza. Le nuove sequenze create vengono quindi ordinate per costo in ordine crescente, per poi essere passate all'algoritmo di Machine Learning. Se una di queste viene valutata positivamente viene immediatamente aggiunta alla soluzione, iterando il procedimento su una nuova sequenza che ha come nodo iniziale l'ultimo nodo della sequenza aggiunta alla soluzione. In caso contrario, si procede alla costruzione di una nuova sequenza. Quando l'algoritmo di Machine Learning non accetta più alcuna sequenza e la soluzione presenta un numero di nodi maggiore al 70%, si aggiunge alla soluzione la permutazione dell'ultima sequenza creata con costo minore. L'ultima sequenza inserita nella soluzione è quella che chiude il percorso e presenta il costo minore.

2.2.1 Implementazione del Timeout

Con il termine *timeout* si fa riferimento a una variabile del codice dell'algoritmo greedy, che funge da contatore per evitare che alcune esecuzioni del programma rimangano bloccate nella ricerca di una sequenza valutata positivamente dall'algoritmo di Machine Learning. Una variabile di questo tipo è necessaria proprio per la difficoltà riscontrata da alcuni algoritmi di trovare una *sequenza buona* con i pochi nodi rimasti all'interno del vicinato. Questo problema talvolta si verifica, senza questa variabile, prima del raggiungimento di un numero di nodi all'interno della soluzione del 70%, facendo rimanere in stallo il programma nella ricerca di una sequenza idonea. Quando il timeout raggiunge il numero prefissato, e quindi l'algoritmo ha già scartato diverse sequenze, il programma prende la permutazione con costo minore dell'ultima sequenza analizzata e la inserisce all'interno della soluzione, per poi continuare con la ricerca della sequenza successiva.

2.3 Configurazione dei Parametri d'Esecuzione

L'algoritmo è stato pensato per funzionare con sequenze di lunghezza diversa, pertanto è possibile eseguirlo configurandone l'esecuzione:

```
python greedy_algorithm.py [-h] [--s LUNGHEZZA_SEQUENZE] [--i NODO_DI_INIZIO]
[--p NUMERO_PROBLEMA] || [--file NOME_FILE_TSPLIB]
```

Capitolo 3

Analisi e Test

L'algoritmo greedy è stato testato con tutti gli algoritmi di Machine Learning citati nel capitolo precedente. Seppur in un primo momento alcuni di questi algoritmi individuassero correttamente singole sequenze, buone e cattive, appartenenti a soluzioni del TSP, dopo diverse esecuzioni dell'algoritmo greedy mirate alla costruzione di soluzioni complete, i risultati sono differenti.

3.1 Performance degli Algoritmi di Machine Learning

Per testare il comportamento dell'algoritmo greedy nel costruire soluzioni ai problemi vengono usati 8 istanze di problemi del TSP. I primi 5 sono grafi generati casualmente, con nodi distribuiti uniformemente nello spazio e di coordinate di grandezza differente. Gli ultimi 3 problemi analizzati sono problemi noti del TSP che hanno un numero di nodi vicino a 100: *ch130*, *ch150*, *eil101*. Si è pensato di testare l'algoritmo greedy anche con problemi di grandezza diversa da 100 nodi per valutarne la capacità di risoluzione. Poiché i nodi all'interno del vicinato vengono scelti in modo casuale, per scongiurare eventuali risultati "fortunati" ottenuti da un algoritmo piuttosto che da un altro, sono state effettuate 3 simulazioni per ogni sequenza di diversa lunghezza (5/6/7/8) per ciascun algoritmo, con seed differenti. In tutto sono state quindi effettuate *3 Simulazioni* per *4 sequenze diverse*, per ciascuno dei *6 Algoritmi*, per un totale di *72 Simulazioni* per problema.

Risultati Nella tabella sottostante viene riportato il numero di vittorie per ciascun algoritmo. Un algoritmo "vince" quando, considerate tutte le 72 simulazioni per ogni problema, è quello che trova la soluzione migliore (rappresentata dal percorso più breve).

AdaBoost	Decsion Tree	KNN	Logistic Regression	Naive Bayes	Random Forest
1	0	1	1	3	2

Tabella 3.1: Numero di Vittorie di un Problema per Algoritmo

Come si nota dalla tabella 3.1 l'algoritmo *Naive Bayes*, impreciso nelle simulazioni atte a decretare la bontà di una sequenza, riesce a trovare alcune delle soluzioni migliori. In questo caso, su 7 di 8 problemi gli algoritmi trovano i percorsi più brevi utilizzando sequenze da 7 nodi.

	AdaBoost	Decsion Tree	KNN	Logistic Regression	Naive Bayes	Random Forest
SEQ. 5	0	1	1	1	5	2
SEQ. 6	2	0	2	3	3	0
SEQ. 7	1	0	2	1	3	1
SEQ. 8	0	1	1	0	3	3
TOT.	3	2	6	5	14	6

Tabella 3.2: Numero di Vittorie degli Algoritmi per Sequenza

Per i risultati mostrati in tabella 3.2 vengono considerati gli algoritmi che ottengono le soluzioni migliori a ciascun problema per ogni sequenza. Anche in questo caso si registrano degli ottimi risultati per gli algoritmi *Naive Bayes*; buone per *KNN* mentre *Logistic Regression* e *Random Forest*, trovano le soluzioni migliori solo per alcune sequenze. Il numero di istanze è talvolta maggiore di 8 perché alcuni algoritmi ottengono lo stesso risultato. Si procede calcolando, per ciascun algoritmo utilizzato nel problema, la media delle *soluzioni* ottenute *per ciascuna sequenza*, decretando come vincitore l'algoritmo con la media minore fra tutte.

AdaBoost	Decsion Tree	KNN	Logistic Regression	Naive Bayes	Random Forest
0	0	2	4	4	0

Tabella 3.3: Numero di Vittorie in media per Algoritmo considerate le Singole Sequenze

Nella tabella 3.3, come per il caso precedente, *Naive Bayes* ha avuto delle ottime performance; allo stesso modo *Logistic Regression* seguita dall'algoritmo *KNN*. In particolare, gli algoritmi trovano le soluzioni in media migliori utilizzando sequenze da 7 e da 8, ma non da 5.

AdaBoost	Decsion Tree	KNN	Logistic Regression	Naive Bayes	Random Forest
0	0	3	1	5	0

Tabella 3.4: Numero di Vittorie in media complessivo per Algoritmo

Nella tabella 3.4 si calcola la media complessiva di *tutte le soluzioni* di ogni algoritmo per ogni problema. Un algoritmo "vince" quando ottiene la media delle soluzioni minore per il problema.

	AdaBoost	Decsion Tree	KNN	Logistic Regression	Naive Bayes	Random Forest
Best SEQ. 5	1	0	0	5	4	0
Best SEQ. 6	0	0	3	2	3	1
Best SEQ. 7	1	0	1	2	4	1
Best SEQ. 8	0	0	4	1	4	0
TOT.	2	0	8	10	15	2

Tabella 3.5: Numero di Vittorie in Media degli Algoritmi per Sequenza

In tabella 3.5 viene riportato, per ciascun algoritmo in ogni problema, il numero di vittorie considerando la media delle *3 simulazioni* con seed differenti fatte per ciascuna sequenza. Anche per quest'ultimo caso gli algoritmi più performanti rimangono gli stessi. Nel complesso, tra gli algoritmi testati, quelli che si prestano meglio ad essere utilizzati per guidare le scelte nell'algoritmo greedy sono *Naive Bayes, Logistic Regression e KNN*: utilizzando questi algoritmi è infatti possibile ottenere le soluzioni migliori che l'algoritmo può raggiungere. Gli algoritmi rimanenti, come *AdaBoost e Random Forest* possono essere comunque utilizzati, ma non assicurano buoni risultati. Per ultimo, l'algoritmo *Decision Tree* è quello che si presta meno ad essere utilizzato all'interno dell'algoritmo greedy, in quanto è con questo algoritmo che il greedy trova le soluzioni peggiori. I risultati ottenuti sono diversi dai precedenti in quanto l'obiettivo è ora quello di *creare un percorso valutando sequenze contigue*, mentre in un primo momento le sequenze in input erano scollegate tra loro e non atte a creare una soluzione completa.

3.2 L'Importanza della Lunghezza della Sequenza

Durante lo svolgimento dei test si è potuto notare un altro parametro che influisce sulle performance dell'algoritmo: *la lunghezza della sequenza* scelta per risolvere il problema del TSP. In particolare, si nota che l'algoritmo agisce meglio se si utilizzano sequenze di 7 nodi. Questo avviene perché utilizzando sequenze con pochi nodi (5/6) l'errore della sequenza (poiché difficilmente risulta essere quella della soluzione ottima) si propaga man mano che l'algoritmo procede nel costruire la soluzione, aumentando quindi l'errore complessivo finale. La sequenza da 7 nodi risulta preferibile alle altre per due motivi: la sequenza, considerando grafi da 100 nodi, introduce un errore minore di quelle da 5 e da 6, mentre risulta preferibile a quella da 8 per come viene costruito il dataset. Infatti, maggiore è il numero della sequenza, minore è il numero di record presi per ciascuno dei 500 grafi utilizzati per la costruzione del dataset. Dunque, un dataset costruito per sequenze da 7 nodi risulta ben bilanciato per il raggiungimento dell'obiettivo di questo elaborato.

Conclusioni

L'obiettivo dell'elaborato è stato raggiunto: l'algoritmo greedy costruito è in grado di trovare soluzioni ai problemi del TSP. Le soluzioni trovate, tuttavia, si discostano dalle soluzioni ottime e dalle performance dei migliori algoritmi utilizzati per risolvere queste tipologie di problema. Ciononostante, è possibile configurare l'algoritmo attraverso dei semplici comandi da terminale per modificarne i parametri di risoluzione adottati e talvolta raggiungere soluzioni subottimali per alcuni problemi (es. *eil101*). Inoltre, uno dei punti di forza dell'algoritmo è la sua capacità di poter trovare soluzione a problemi di dimensione diversa, come si è visto nei risultati ottenuti risolvendo problemi reali della libreria TSPLIB.

Sviluppi Futuri Poiché l'algoritmo non restituisce soluzioni ottimali, esso è migliorabile sotto diversi aspetti. La prima miglioria attuabile sarebbe quella di incrementare la dimensione del dataset, aumentando il numero di grafi utilizzati anche con coordinate diverse e più grandi. In questo modo gli algoritmi vengono allenati su una maggiore quantità di dati ed è possibile ottenere risultati ancora migliori. Sarebbe inoltre possibile introdurre un *buffer* che tenga traccia delle sequenze scartate dall'algoritmo, così che quando questo impiega troppo tempo nella ricerca di una buona sequenza, ricordi, tra quelle scartate, la migliore da inserire nella soluzione. In alternativa sarebbe possibile considerare il grado di affidabilità con cui un algoritmo effettua una scelta su una sequenza; sebbene ora viene inserita la prima sequenza con costo minore accettata dall'algoritmo di Machine Learning, si potrebbero memorizzare tutte le sequenze valutate positivamente e aggiungere alla soluzione una di quelle con costo minore e affidabilità più alta. Agendo in questo modo l'algoritmo potrebbe operare in maniera più efficace e conseguire risultati più attendibili. Ad oggi l'algoritmo greedy viene di default guidato dall'algoritmo di Machine di Learning *Naive Bayes*. Se in seguito a questi miglioramenti anche altri algoritmi risultassero efficienti, si potrebbe dare all'utente la possibilità di scegliere l'algoritmo quando esegue il programma.

Bibliografia

[1] Heidelberg University. Tsplib. http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/index.html.