Università degli studi di Milano - Bicocca

Decision Models

Final Project

Ottimizzazione di percorsi aerei tramite algoritmo genetico

Autori:
Corrado Montoro - 841489 - c.montoro@campus.unimib.it
Luca Lazzati - 850334 - l.lazzati2@campus.unimib.it
Roberto Salerno - 784336- r.salerno4@campus.unimib.it

Abstract

Il progetto vuole ottimizzare le rotte aeree negli Stati Uniti. A tale scopo il progetto si propone di minimizzare i valori massimi di ritardo associati alle rotte e, allo stesso tempo, la distanza totale percorsa dall'aereo nel suo tragitto. Inoltre, si vuole rispondere alla domanda di ricerca se il percorso più breve sia anche quello con la fitness value migliore.
Per arrivare allo scopo si è utilizzata una combinazione di algoritmi: quello di Dijkstra, il Montecarlo e un algoritmo genetico. I dati rappresentano tutti i voli aerei negli Stati Uniti nell'anno 2015.
Il progetto riesce a dimostrare che si possono creare delle rotte che ottimizzano entrambi i parametri presi in considerazione.

Obiettivi

L'algoritmo di Dijkstra viene qui utilizzato con lo scopo di calcolare la sequenza di rotte più breve. In seguito, ci si è posto il problema di cercare una combinazione che minimizzasse allo stesso tempo i valori massimi associati al ritardo sulle rotte e la distanza associata alla rotta. Per raggiungere questo scopo si è utilizzato l'algoritmo Montecarlo, il quale è in grado di trovare i valori massimi associati al ritardo sulle rotte. Si è poi proceduto ad implementare un algoritmo genetico che trovasse nuove combinazioni con minori valori dei massimi ritardi associati alle rotte e minore distanza associata alle rotte.

Formulazione del problema

In questo paper si considerano le rotte (*flight route*) come nodi. Non tutti i nodi sono connessi ad ogni nodo rimanente, perciò il grafo non è completo. Tuttavia, dal momento che esiste almeno un percorso che consente di passare da una rotta ad un'altra il grafo è connesso.

Si è proceduto alla formulazione della funzione obiettivo. La funzione obiettivo è:

$$(1)\ \ min \sum_{i \in y} \sum_{j \in h} d_{ij} x_{ij} \ ,$$

soggetto a $\sum_{i \in y} x_{ij} = 1$ per $j \in h$

x_{ij}= {1 se la rotta j appartiene al percorso i, 0 altrimenti}

$d_{ij} = valore\ massimo\ di\ ritardo\ del\ volo_{ij} + distance_{ij}$

Il motivo per cui d_{ij} è calcolato come la somma delle variabili "*valore massimo di ritardo del volo*" e "*distance*" (e non, per esempio, come il prodotto) è per non annullare il valore di un parametro nel caso l'altro fosse pari a zero.

Datasets

I dati sono stati presi dalla piattaforma Kaggle, alla pagina "2015 Flight Delays and Cancellations", dove sono presenti tre datasets[1]. Quello utilizzato è solo "flights.csv". Le *features* utilizzate sono:

- "Year", "Month", "Day", le quali ci indicano il giorno preciso in cui un determinato volo è stato effettuato negli Stati Uniti nell'anno 2015;
- "Origin_airports", "Destination_Airport", le quale ci indicano l'aeroporto di partenza e di arrivo dell'aereo;
- "Distance", che ci indica, per ogni tratta (dall'aeroporto di partenza all'aeroporto di arrivo), la distanza in miglia;
- "Arrival_delay", che ci indica, in minuti, il ritardo dell'aereo all'arrivo: i numeri negativi indicano un ritardo mentre i numeri positivi indicano che l'aereo è arrivato in anticipo.

Processo di manipolazione dei dati

[1] https://www.kaggle.com/usdot/flight-delays

Il dataset contiene dei valori privi di senso o comunque non interpretabili. Infatti, alcuni aeroporti sono indicati con una serie numerica, ma non viene fornita nessuna informazione che consenta di ricondurre tale serie al nome dell'aeroporto. Questi valori sono stati cancellati. L'informazione del giorno in cui il volo è stata effettuata è divisa in tre colonne ("Year", "Month", "Day"). Per i nostri scopi è stata creata un'unica *feature* 'Date', contenente la data completa.

La percentuale di ritardi è stata calcolata nel seguente modo: sono stati considerati in ritardo tutti i voli che avevano un valore negativo nella colonna "Arrival_delay" e di questi si è calcolato la percentuale sul totale.

"Distance" e "percentage_delay" sono stati normalizzati. E' importante normalizzare queste variabili perché la differenza di grandezza tra la percentuale di ritardi, che ovviamente ha come range i valori da 0 a 1, e la distanza, calcolata in miglia, è notevole. I parametri possono essere ottimizzati con metodi diversi. Un metodo di normalizzazione è noto come *rescaling*. Esso può essere rappresentato come:

$$(2)\ M_j\ (M_o - \min(M_o)\ ) / \max(M_o) - \min(M_o))$$

Dove M_j il valore scalato del parametro, M_o è il singolo valore del parametro. Dopo che i parametri sono stati normalizzati, l'importanza del parametro non sarà modificata da alcun *bias* nella *fitness function*. Tuttavia, nel caso avessimo voluto dare più importanza ad uno dei due parametri (alla distanza o al ritardo), la somma dei *bias* dati al parametro dovrebbe essere stata uguale a 1.

Algoritmi

L'algoritmo di Dijkstra è un algoritmo per trovare i percorsi più brevi tra i nodi in un grafo. In questo caso i nodi rappresentano rotte aeree. Per trovare il percorso più breve abbiamo utilizzato la libreria dijkstra_path[2]. Tale libreria è in grado di calcolare il percorso più breve tra tutti quelli disponibili.

[2] https://networkx.github.io/documentation/networkx-1.10/reference/generated/networkx.algorithms.shortest_paths.weighted.dijkstra_path.html

Dopo aver calcolato la rotta più breve, vengono calcolati anche tutti i percorsi possibili. Per ogni percorso viene calcolato la distanza relativa ad esso e anche la sua percentuale di ritardo. Questi valori vengono poi normalizzati con la funzione (2).

L'algoritmo di Montecarlo è qui usato per trarre stime, attraverso delle simulazioni che si basano sui dati disponibili, di possibili ritardi massimi degli aerei. La simulazione viene eseguita per mille volte. Dopo aver simulato i ritardi massimi per mille volte, si stabilisce il valore massimo del ritardo associato a una certa rotta.

Successivamente, si nomina la colonna dei ritardi massimi "Delay_Series". Ogni percorso avrà quindi un valore corrispondente al massimo ritardo calcolato con l'algoritmo di Montecarlo. Avendo ricavato, col metodo Montecarlo, i valori del ritardo massimo, questi vengono normalizzati secondo la funzione (2) in modo tale da poterli sommare con quelli della distanza già standardizzata in precedenza.

Dopo il Montecarlo, il progetto passa alla costruzione di un algoritmo genetico. Per algoritmo genetico si fa riferimento ad una tecnica di ottimizzazione combinatoria basata sulla selezione naturale e sui principi della genetica. Per ricavare le *fitness value* migliore ci affidiamo al *Genetic Algorithm*. Il principio di funzionamento del *Genetic Algorithm* è il seguente:

1. Inizializzazione della popolazione iniziale
2. Selezione iniziale di alcuni percorsi di rotte, ovvero i "genitori"
3. Calcolo della *Fitness Value*
4. Mutation, ovvero selezione randomica degli individui e scambio di geni
5. Crossover, generazione di nuovi individui, ovvero i "figli"
6. Iterazione sulla nuova popolazione e valutazione delle *fitness value* sui nuovi parametri
7. Aggiunta dei 5 migliori individui alla popolazione iniziale
8. Infine restituzione della migliore soluzione generata dalle iterazioni

Il nostro algoritmo consente tre tipi di selezione: "weighted", "random" e "besthalf". La prima consente di fare un campionamento dove le righe hanno pesi diversi a seconda della loro *fitness function* e gli individui con una *fitness function* migliore sono più facilmente scelti, la seconda è un campionamento casuale semplice, mentre la terza prende la metà degli individui con la *fitness function* migliore.

La prima fase dell'algoritmo genetico è quella di trovare gli individui da sottoporre a trasformazione. Per ottenerli è stata utilizzata la colonna delle rotte, ovvero "Path", nella quale ci sono tutte le diverse rotte con una partenza e un arrivo in comune.
La nostra *fitness function è* calcolata secondo la seguente equazione:

(3) $\rho + \delta$

dove ρ è l'inverso del valore massimo del ritardo (1 - normalized(*valore massimo di ritardo del volo*)), calcolato con l'algoritmo di Montecarlo, e δ è l'inverso del valori normalizzato della distanza(1 - normalized(*valore massimo di ritardo del volo*)). Dal momento che sia il range di ρ che quello di δ sono tra 0 e 1, il range della somma dei loro valori sarà tra 0 e infinito.
Per calcolare la *Fitness Value* si crea un processo che inizia sommando i ritardi standardizzati con le distanze standardizzate per ogni *route* e successivamente creando una colonna 'Probabilità', necessaria ad un ordinamento casuale degli Individui per il calcolo del *Genetic Algorithm.*
Vengono poi effettuate poi le operazione di Mutation e Crossover. La Mutation effettua lo scambio randomico di due individui facenti parte della stessa lista. Questo avviene per tutti gli individui presenti della colonna Path.
Il Crossover, ultima funzione svolta dal processo, scambia due cromosomi di diverse liste, incrociandoli tra loro.
L'intero procedimento viene ripetuto 500 volte in modo che ogni nuovo individuo creato abbia la propria *fitness value*. I vari valori delle *fitness value* ottenuti dal *Genetic Algorithm* vengono comparati ad ogni iterazione e i migliori di questi vengono aggiunti alla popolazione iniziale.

Valutazione dei risultati e conclusioni

Per valutare i risultati si offre un caso concreto. Prendendo in considerazione le rotte possibili tra gli aeroporti "GUM" e "CLD", i risultati ottenuti dall'algoritmo genetico dopo 500 iterazioni sono rappresentati dal grafico seguente:

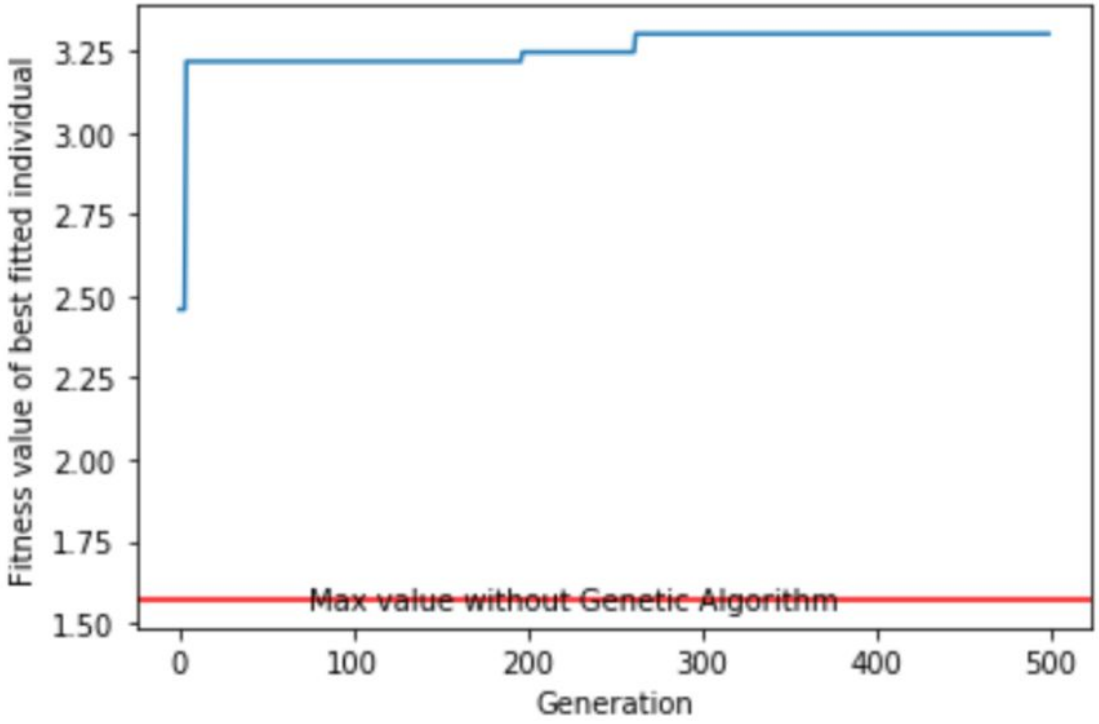

Figura 1: plot della fitness value migliore trovata dall'algoritmo genetico per 500 generazioni.

Come si può vedere dalla figura 1, l'algoritmo genetico riesce a migliorare la fitness function già dopo la prima generazione. La linea rossa rappresenta la fitness value migliore prima dell'applicazione dell'algoritmo genetico. Tuttavia, l'algoritmo genetico non garantisce di trovare il risultato globalmente migliore, ma semplicemente il suo criterio di terminazione è che si sia raggiunta la generazione numero 500.

Per quanto riguarda il caso preso in esame, i migliori venti risultati di fitness value trovati dall'algoritmo genetico sono:

Tabella 1: i migliori venti individui al termine delle 500 iterazioni

individuals	Path_delay_std	Path_distance_std	fitness_values
[(GUM, HNL), (HNL, ANC), (ANC, LAX), (LAX, CLD)]	[0.5679758308157099, 0.7709090909090909, 0.776...	[0.23821039903264818, 0.4445787988714228, 0.53...	3.301903
[(GUM, HNL), (HNL, SLC), (SLC, LAX), (LAX, CLD)]	[0.5679758308157099, 0.8637770897832817, 0.550...	[0.23821039903264818, 0.4008464328899637, 0.88...	3.245882
[(GUM, HNL), (HNL, LIH), (LIH, LAX), (LAX, CLD)]	[0.5679758308157099, 0.5993162393162393, 0.742...	[0.23821039903264818, 0.9836759371221282, 0.47...	3.217191
[(GUM, HNL), (HNL, MSP), (MSP, LAX), (LAX, CLD)]	[0.5679758308157099, 0.8444444444444444, 0.585...	[0.23821039903264818, 0.20374848851269645, 0.6...	3.164179
[(GUM, HNL), (HNL, KOA), (KOA, LAX), (LAX, CLD)]	[0.5679758308157099, 0.5968020065841041, 0.669...	[0.23821039903264818, 0.9713825070536074, 0.49...	3.143932
[(GUM, HNL), (HNL, ITO), (ITO, LAX), (LAX, CLD)]	[0.5679758308157099, 0.5516088591725867, 0.691...	[0.23821039903264818, 0.9607013301088271, 0.51...	3.120897
[(GUM, HNL), (HNL, OGG), (OGG, LAX), (LAX, CLD)]	[0.5679758308157099, 0.5543293718166384, 0.657...	[0.23821039903264818, 0.9840790004030633, 0.50...	3.093537
[(GUM, HNL), (HNL, PDX), (PDX, LAX), (LAX, CLD)]	[0.5679758308157099, 0.5918653576437587, 0.626...	[0.23821039903264818, 0.4796453043127771, 0.83...	3.057494
[(GUM, HNL), (HNL, SAN), (SAN, LAX), (LAX, CLD)]	[0.5679758308157099, 0.6151419558359621, 0.566...	[0.23821039903264818, 0.47742845626763397, 0.9...	3.056526
[(GUM, HNL), (HNL, SFO), (SFO, LAX), (LAX, CLD)]	[0.5679758308157099, 0.6486220472440944, 0.494...	[0.23821039903264818, 0.5209592906086256, 0.93...	3.017156
[(GUM, HNL), (HNL, DFW), (DFW, LAX), (LAX, CLD)]	[0.5679758308157099, 0.6666666666666667, 0.581...	[0.23821039903264818, 0.24163643692059655, 0.7...	3.007446
[(GUM, HNL), (HNL, DEN), (DEN, LAX), (LAX, CLD)]	[0.5679758308157099, 0.6818181818181819, 0.487...	[0.23821039903264818, 0.3260781942765014, 0.83...	2.968059
[(GUM, HNL), (HNL, EWR), (EWR, LAX), (LAX, CLD)]	[0.5679758308157099, 0.7476635514018692, 0.580...	[0.23821039903264818, 0.004232164449818665, 0....	2.966273
[(GUM, HNL), (HNL, IAD), (IAD, LAX), (LAX, CLD)]	[0.5679758308157099, 0.6867469879518072, 0.620...	[0.23821039903264818, 0.0334542523176139, 0.54...	2.960990
[(GUM, HNL), (HNL, SEA), (SEA, LAX), (LAX, CLD)]	[0.5679758308157099, 0.5755478662053056, 0.553...	[0.23821039903264818, 0.4647319629181782, 0.81...	2.958234
[(GUM, HNL), (HNL, ORD), (ORD, LAX), (LAX, CLD)]	[0.5679758308157099, 0.7073170731707317, 0.531...	[0.23821039903264818, 0.14913341394598956, 0.6...	2.949516
[(GUM, HNL), (HNL, LAS), (LAS, LAX), (LAX, CLD)]	[0.5679758308157099, 0.626387176325524, 0.4609...	[0.23821039903264818, 0.4476017734784361, 0.95...	2.948480
[(GUM, HNL), (HNL, PHX), (PHX, LAX), (LAX, CLD)]	[0.5679758308157099, 0.5901785714285714, 0.504...	[0.23821039903264818, 0.4163643692059653, 0.92...	2.941102
[(GUM, HNL), (HNL, ATL), (ATL, LAX), (LAX, CLD)]	[0.5679758308157099, 0.6265060240963856, 0.625...	[0.23821039903264818, 0.0969367190648932, 0.61...	2.939183
[(GUM, HNL), (HNL, OAK), (OAK, LAX), (LAX, CLD)]	[0.5679758308157099, 0.6339113680154143, 0.407...	[0.23821039903264818, 0.5187424425634825, 0.93...	2.915609

Paragonando il percorso più breve e il miglior risultato trovato dopo l'algoritmo genetico, si nota che non sempre il percorso più breve è anche quello con la fitness value migliore.

L'algoritmo genetico trova inoltre dei valori della fitness value che sono, per la maggior parte delle volte, di gran lunga superiori alla fitness value migliore prima dell'applicazione dell'algoritmo genetico.

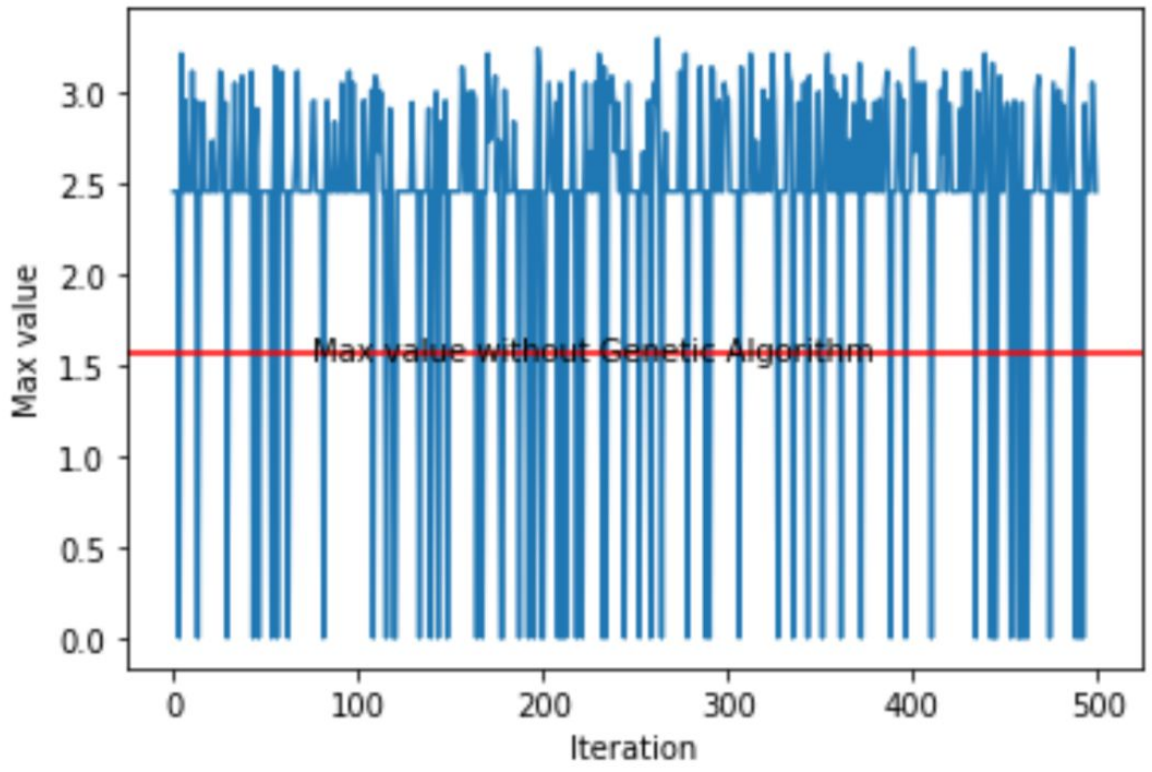

Figura 2: plot della fitness value migliore trovata dall'algoritmo genetico per 500 generazioni.

Miglioramenti

Purtroppo il dataframe non forniva informazione riguardanti i costi dei singoli aerei sulle singole tratte. Tuttavia, si fa notare che sarebbe stato facilmente possibile introdurre questo parametro, standardizzarlo insieme agli altri due e inserirlo nella *fitness function*.

Rifermenti

https://www.kaggle.com/usdot/flight-delays
https://www.python-course.eu/networkx.php
https://scikit-learn.org/stable/