Immagine che contiene testo, schermata, software, Icona del computer

Descrizione generata automaticamente

Rome’s treasures unveiled:  
an insider’s journey

Gruppo di lavoro

* Daniele Guerra, 744399, d.guerra6@studenti.uniba.it
* Nicola Lassandro, 735968, n.lassandro4@studenti.uniba.it

Repository GitHub

<https://github.com/NicolaLassandro/progetto_icon_guerra_lassandro>

AA 2022-2023

Introduzione

Il software lavora nel dominio degli itinerari turistici, in particolare (al momento) è dedicato alla città di Roma.

Esso mette a disposizione dell’utente una panoramica completa di ciò che la città ha da offrire, garantendo una flessibilità che permette di personalizzare la propria esperienza in base alle proprie preferenze ed esigenze.

L’obiettivo del software è quello di individuare il percorso di visita migliore, principalmente sulla base della distanza percorsa, si tiene anche conto dei limiti di tempo e budget impostati, tutto ciò garantendo una qualità di punti di interesse visitati elevata.

Sommario

[Gruppo di lavoro 1](#_Toc137745240)

[Repository GitHub 1](#_Toc137745241)

[Introduzione 2](#_Toc137745242)

[Strutturazione del progetto 4](#_Toc137745243)

[Elenco argomenti di interesse 5](#_Toc137745244)

[Rappresentazione della conoscenza mediante Kb 6](#_Toc137745245)

[Sommario 6](#_Toc137745246)

[Strumenti utilizzati 7](#_Toc137745247)

[Decisioni di progetto 8](#_Toc137745248)

[Valutazioni 8](#_Toc137745249)

[Problema di ricerca su grafo 9](#_Toc137745250)

[Sommario 9](#_Toc137745251)

[Strumenti utilizzati 9](#_Toc137745252)

[Decisioni di progetto 9](#_Toc137745253)

[Valutazioni 9](#_Toc137745254)

[Apprendimento supervisionato (regressione) 10](#_Toc137745255)

[Sommario 10](#_Toc137745256)

[Strumenti utilizzati 10](#_Toc137745257)

[Decisioni di progetto 10](#_Toc137745258)

[Valutazioni 10](#_Toc137745259)

[Conclusioni e sviluppi futuri 11](#_Toc137745260)

[Riferimenti bibliografici 12](#_Toc137745261)

Strutturazione del progetto

* Cartella Knowledge: contiene tutti i file prolog e python relativi alla rappresentazione della conoscenza nella Kb utilizzata nel progetto.
  + Facts.pl
  + KbManager.py
  + Landmark.py
  + Preprocessor.py
  + Rules.pl
  + RuntimeFacts.pl
  + Utility.py
* Cartella Learning: contiene tutti i file python relativi alla fase di apprendimento supervisionato.
  + FeedbackGenerator.py
  + Kfold.py
  + Knn.py
  + ModelInitializer.py
  + PreProcessorLearning.py
  + RegressionTree.py
* Cartella Logs: contiene tutti i file testuali di log utilizzati per supervisionare l’andamento degli script di tutte le altre cartelle.
* Cartella Search: contiene tutti i file python utilizzati nel problema di ricerca su grafo.
  + Cartella Libs: contiene gli algoritmi predefiniti che sono stati utilizzati come base per il problema di ricerca realizzato dal gruppo.
  + ItinerarySearchProblem.py
  + **MainSearch.py**: .
  + NodeGraph.py
* Cartella Storage: contiene tutti i file serializzati in formato pickle.

Elenco argomenti di interesse

Di seguito sono presentati i macro-argomenti trattati durante lo sviluppo dell’applicativo.

* Rappresentazione della conoscenza mediante Kb
* Problema di ricerca su grafo
* Apprendimento supervisionato (regressione)

In fase di scelta dell’idea progettuale e della sua realizzazione si è cercato di includere la più vasta gamma di argomenti trattati durante il corso di Ingegneria della conoscenza.

Rappresentazione della conoscenza mediante Kb

Sommario

La prima operazione svolta durante lo sviluppo dell’applicativo turistico è stata la raccolta dei dati relativi ai punti di interesse della città di Roma, i quali sono stati utilizzati per l’inizializzazione delle istanze della classe di riferimento: Landmark.  
Gli attributi presenti in tale classe sono:

* *placeId, name, address, type, properties, lat, lon, age*.  
  Questi rappresentano le caratteristiche “anagrafiche” del luogo di interesse.
* *rating, ratingCount*.  
  Questi illustrano la popolarità e la qualità del punto di interesse sulla base dei giudizi degli utenti.
* *centreDistance*.  
  E’ la distanza da quello che è stato definito arbitrariamente il punto centrale del turismo della città.
* *tourismRate*, *price*.  
  Questi fanno riferimento alle carattistiche “turistiche”, le quali sono fattori influenzanti della visitabilità del luogo.
* *handicapAccessibility*, *surface*, *height*.  
  Questi sono gli attributi strutturali del punto di interesse.

Tutti gli altri campi presenti verranno popolati solo in un secondo momento.

Successivamente queste istanze sono state memorizzate all’interno di un dizionario e tutte le feature sono state immagazzinate all’interno di una base di conoscenza sotto forma di fatti. La maggior parte di essi seguivano uno stesso formato: feature(landmark\_name, feature\_value).

Di seguito sono presentati alcuni esempi.  
Immagine che contiene testo, schermata, software, Software multimediale

Descrizione generata automaticamente

La feature *distance* fa eccezione dal formato Prolog delle altre, essa è così strutturata: feature(landmark\_1\_name, landmark\_2\_name, feature\_value).  
Immagine che contiene testo, schermata, software, schermo

Descrizione generata automaticamente  
Questa feature è di fondamentale importanza all’interno del progetto in quanto si trova alla base del modulo di ricerca su grafo, nucleo del caso di studio.  
Per questo motivo è stato necessario conservare tutti i fatti ad essa relativi nell’apposito file Prolog, cosa che non è avvenuta per tutte le feature.

La KB, dopo essere stata arricchita con diverse regole, è stata interrogata per generare nuova conoscenza.

Tra le regole realizzate ce ne sono alcune particolarmente importanti:

* *calculateDensity*, che permette di definire la densità turistica del luogo, intesa come media delle distanze da tutti gli altri posti registrati (prelevati precedentemente mediante il predicato findall).  
  Immagine che contiene testo, schermata, software, computer

  Descrizione generata automaticamente
* *calculateTourismPriority*, serve a calcolare la priorità di visità del punto di interesse sulla base delle altre feature che lo caratterizzano.  
  Ognuna di esse influenza in modo diverso il risultato finale, che viene successivamente salvato nel dizionario e utilizzato nel modulo di ricerca su grafo per garantire una soglia minima di qualità dei luoghi previsti dall’itinerario turistico.  
  Immagine che contiene testo, schermata, software, computer

  Descrizione generata automaticamente
* *calculateTimeToVisit*, si occupa di stabilire il tempo necessario alla visita di un luogo, questo è influenzato sia da aspetti strutturali (come le dimensioni) sia da aspetti turistici (all’aumentare del tasso di turismo, il luogo risulta più affollato).  
  Immagine che contiene testo, schermata, software, computer

  Descrizione generata automaticamente

Di seguito è presentato un pezzo di codice python con il quale viene interrogata la base di conoscenza per ciascuno dei punti di interesse registrati, i risultati ricevuti vengono posti in un contenitore (result) e successivemente estratti e assegnati all’omonimo attributo dell’istanza di Landmark corrispondente nel dizionario.

Immagine che contiene elettronica, testo, computer, schermata

Descrizione generata automaticamente

Anche le interrogazioni che restituiscono risultati di diverso tipo (ad esempio booleani) seguono un ragionamento analogo.

Immagine che contiene testo, schermata, software, computer

Descrizione generata automaticamente

Infine, il dizionario contenente tutte le istanze di Landmark è stato aggiornato, con tutti i valori delle nuove feature a disposizione, e serializzato per poter essere utilizzato nei successivi moduli del progetto.

Strumenti utilizzati

La raccolta dei dati utili è stata effettuata mediante l’impiego di una API a pagamento messa a disposizione da Google Places (la piattaforma di Google Maps): **nearbysearch**, essa consente di recuperare una lista di luoghi nelle vicinanze di una determinata posizione geografica.

Per il suo utilizzo è stato necessario generare (gratuitamente) una API-Key della durata di pochi mesi ed è stata impiegata la **libreria Requests** fornita da Python.

Immagine che contiene testo, schermata, software, computer

Descrizione generata automaticamente  
La chiamata richiede diversi parametri per restituire i risultati corretti, di seguito sono citati quelli impiegati nel corrente caso di studio:

* API-Key
* Le coordinate di latitudine e longitudine del centro di ricerca, a partire dal quale sono stati individuati i risultati.
* Il raggio massimo di estensione della ricerca dei luoghi.
* Il tipo dei luoghi, espresso mediante una parola chiave, sulla base del quale viene effettuto il filtraggio, allo scopo di ottenere risultati specifici.

La risposta risultante della chiamata viene restituita in formato json e contiene al suo interno numerosi attributi, tra questi ne sono stati selezionati ed estratti alcuni utili allo scopo dell’applicativo:

* *place\_id*, un identificativo che viene assegnato ad ogni elemento registrato sulla piattaforma Google.
* *user\_ratings\_total*, un intero che rappresenta la quantità di recensioni degli utenti relative all’elemento.
* *name*, il nome del luogo.
* *geometry.location.lat* e *geometry.location.lon*, le coordinate spaziali del luogo in formato decimale.
* *rating*, la media delle votazioni (da 1 a 5) contenute nelle recensioni.

Le restanti caratteristiche dei punti di interesse, come suggerito dal , per via della difficoltà di reperimento, sono state generate casualmente mediante la **libreria random** messa a disposione da python. Ovviamente la generazione casuale ha tenuto conto di intervalli verosimili dei valori delle feature.

Per la rappresentazione della conoscenza e la sua successiva interrogazione si è deciso di utilizzare il **linguaggio Prolog**, un linguaggio di programmazione logico basato su predicati, che rappresentano fatti o relazioni tra oggetti. Un programma Prolog è costituito da un insieme di predicati e regole, queste ultime vengono utilizzate per derivare nuovi fatti o per risolvere interrogazioni sulla base di conoscenza.

Decisioni di progetto

Dal momento che le API di Google restituivano venti risultati per ogni chiamata, è stato necessario ripetere più volte la chiamata API finché non si fossero ottenuti tutti i risultati.

Inoltre, per poter ottenere un numero elevato di luoghi si è deciso di ripetere la chiamate API per quattro volte, fornendo quattro coppie di coordinate diverse a ciascuna chiamata. Il motivo per il quale sono state fornite quattro coppie di coordinate diverse è dipeso dal fatto che si è cercato di massimizzare i risultati ottenibili ripetendo la chiamata API spostandosi di 1 km a nord, sud, est ed ovest del centro turistico scelto.

Per quanto concerne la scelta delle coppie di coordinate da cui far partire la ricerca, si è deciso di scegliere arbitrariamente un punto che potesse contenere un numero consistente di luoghi tale da poter essere definito come centro turistico di Roma.

Infine, la popolazione della base di conoscenza è avvenuta mediante la scrittura su file dei fatti riguardanti i vari luoghi; si è deciso di utilizzare il metodo *write* di Python per scrivere sul file Prolog, al posto del metodo assertz, poiché le informazioni contenute nella base di conoscenza avrebbero dovuto essere consultate anche negli altri moduli del progetto.

Il metodo assertz sarebbe stato più opportuno qualora, nel ciclo di esecuzione del programma, la base di conoscenza avesse richiesto l’aggiunta di fatti in maniera dinamica per poter consentire di effettuare delle interrogazioni su di essa; nel caso di studio in questione ogni modulo equivale ad un’esecuzione separata.

Problema di ricerca su grafo

Sommario

Il fine del caso di studio è quello di rappresentare un punto di riferimento per i turisti che vogliono visitare i luoghi più importanti della città di Roma, fornendo così il miglior itinerario da seguire per sfruttare al meglio le risorse a disposizione (tempo e budget).

Per poter ottenere il miglior percorso, è stato necessario l’impiego di un algoritmo di ricerca su grafo, che potesse tenere in considerazione le necessità dell’utente e che fosse in grado di restituire un percorso contenente i migliori luoghi da visitare.

Scelto dell’algoritmo

Strumenti utilizzati

Lo strumento impiegato per la ricerca su grafo è l’algoritmo A\*, un algoritmo di ricerca euristica. Esso cerca il percorso ottimale tra due nodi, sommando la funzione di costo fino al punto corrente con una funzione euristica che stima il costo dal punto corrente ad un nodo goal:

La libreria utilizzata per l’implementazione di A\* è AIPython, una libreria contenente un insieme di algoritmi di ricerca pre implementati.

Decisioni di progetto

Per l’impiego dell’algoritmo di ricerca, è stato necessario definire una classe che rappresentasse i nodi da inserire nel grafo. I nodi non rappresentano, come si potrebbe banalmente pensare, i luoghi di interesse ma al contrario delle “situazioni”.

Nello specifico, ciascun nodo conterrà le seguenti informazioni:

* *name*, indica il nome del luogo
* *coveredDistance*, indica la distanza percorsa fino al punto corrente
* *remainingBudget*, indica la disponibilità corrente di budget
* *remainingTime*, indica la disponibilità corrente di tempo
* *visitedNodes*, indica i luoghi precedentemente visitati
* *sumVisitedPriority*, indica la somma delle priorità dei luoghi visitati

Gli ultimi quattro attributi del nodo verranno utilizzati in fase di ricerca per effettuare specifici controlli.

Nella fase di definizione del grafo non si conoscono a priori i nodi goal, questo perché la sua generazione avviene dinamicamente a partire dalla posizione corrente dell’utente (nel caso di studio è stata simulata).

Una generazione statica del grafo avrebbe gravato eccessivamente sulle prestazioni dell’applicativo.

Durante il problema di ricerca, è richiesta una funzione di individuazione dei nodi vicini:

Immagine che contiene testo, schermata

Descrizione generata automaticamente

La funzione *neighbors*, in primo luogo, interroga la base di conoscenza per ottenere la lista dei luoghi vicini a quello indicato nel nodo corrente, sfruttando l’apposita regola Prolog:

Immagine che contiene testo, Carattere, schermata

Descrizione generata automaticamente

Successivamente, per ognuno di essi verifica che non sia stato già visitato, in caso affermativo procede recuperando dalla base di conoscenza le informazioni utili a calcolare la nuova distanza percorsa, il nuovo budget rimanente e il nuovo tempo rimanente.

Tutti questi aspetti vengono utilizzati per la definizione dell’istanza *NodeGraph* relativa ad ogni nodo vicino, questo avviene solo se non vengono sforate le soglie di budget e tempo rimanenti.

Infine, si aggiunge il nodo e l’arco corrispondente al grafo mediante la funzione *arcs.append*.

Dal momento che, la generazione del grafo avviene dinamicamente è stato necessario predisporre una funzione che controllasse man mano se il nodo in analisi fosse un nodo obbiettivo:

Immagine che contiene testo, schermata, software

Descrizione generata automaticamente

La funzione *is\_goal*, analogamente alla precedente, interroga la base di conoscenza per ottenere una lista dei luoghi vicini a quello corrente, successivamente viene verificato che il nuovo budget rimanente e il nuovo tempo rimanente non siano negativi, se questa condizione è rispettata per tutti i punti di interesse vicini il nodo corrente è considerato candidato obbiettivo.

Esso viene confermato solo nel caso in cui la media aritmetica delle priorità dei luoghi visitati, lungo il percorso, non sia inferiore ad una determinata soglia (arbitrariamente è stato scelto 4).

La caratteristica principale dell’algoritmo A\* è la presenza di una funzione euristica, che, nel caso di studio, è stata definita come segue:

Immagine che contiene testo, schermata, software, Software multimediale

Descrizione generata automaticamente

La funzione *heuristic* ha lo scopo di assegnare un valore stimato di costo per raggiungere un nodo obbiettivo, di conseguenza viene subito controllato se il nodo in questione sia obbiettivo o meno, in caso positivo il valore assegnato è 0.

In caso contrario, mediante interrogazioni Prolog, ottiene la minima distanza possibile tra due luoghi registrati, la priorità del luogo associato al nodo in input e il tempo massimo di visita e il costo massimo richiesti da un punto di interesse (nel calcolo successivo si terrà conto di quello minore).

Il valore restituito dall’euristica deriva dal prodotto di tre fattori:

1. Il rapporto tra il tempo rimanente del nodo e il massimo tempo di visita oppure tra il budget rimanente e il costo massimo.  
   L’aver messo il valore massimo al denominatore minimizza il valore del fattore.
2. La distanza minima registrata.
3. Un valore via via decrescente all’aumentare della priorità.  
   Questo fattore funge da discriminante in caso di valori molto simili del prodotto dei due fattori precedenti (se vicini o uguali la preferenza è diretta verso il nodo associato al luogo con priorità maggiore, così la sua euristica restituirà un valore più basso).

Minimizzare i fattori è un requisito fondamentale per far si che l’euristica restituisca valori che siano sempre sottostima del reale costo, permettendo così il corretto funzionamento dell’algoritmo.

Valutazioni

Di seguito, si osserva la differenza tra il percorso individuato da un generico searcher:

Immagine che contiene testo, schermata, Carattere, nero

Descrizione generata automaticamente

e quello individuato dall’algoritmo A\* adattato:

Immagine che contiene testo, schermata, Carattere, nero

Descrizione generata automaticamente

Si può notare come in entrambi i casi si arrivi ad un punto di terminazione dove il tempo o il budget rimanenti impediscano di proseguire (in entrambi gli esempi la risorsa terminata è il tempo, con un altro passo si andrebbe sotto lo zero), anche la condizione sulla priorità media è rispettata: nel primo caso risulta 25 / 6 = 4,17 >= 4, nel secondo caso risulta 20 / 5 = 4 >= 4.

La differenza sostanziale dettata dall’algoritmo utilizzato sta nella distanza percorsa, che nel primo caso risulta essere molto più elevata (2022 m) in relazione alla distanza percorsa nel secondo caso (333 m).

Osservazione sull’impiego di un csp

Apprendimento supervisionato (regressione)

Sommario

Strumenti utilizzati

Decisioni di progetto

Valutazioni

Conclusioni e sviluppi futuri

Riferimenti bibliografici

<https://developers.google.com/maps/documentation/places/web-service/search-nearby?hl=it>

<https://artint.info/AIPython/>