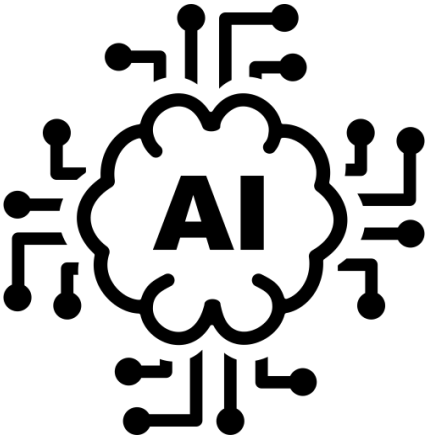
PROGETTO FUNDAMENTALS OF ARTIFICIAL INTELLIGENCE



Sviluppo di un Sistema Esperto nell'ambito del turismo

Andrea Basile **matricola 762676**
Roberto Lorusso **matricola 765447**

Sommario

Introduzione

Idea Generale

Questo documento è stato realizzato per offrire una descrizione del progetto riguardante lo sviluppo di una **Knowledge Base per un Sistema Esperto relativo al dominio del turismo.**

L'obiettivo principale del Sistema Esperto è quello di **pianificare** una sequenza di punti di interesse da visitare, sulla base di vincoli imposti dall'utente.

In particolare si vogliono gestire i percorsi turistici in base a determinati interessi del turista che possono essere:

1) **Categorie** (Movimento artistico, epoca).
2) **Posizione dei punti di interesse** (Posizione geografica, posizione in una determinata città o in un tipo di costruzione).
3) **Autore.**
4) **Prezzo.**
5) **Distanza Massima.**

La strategia è fortemente influenzata anche da altri aspetti come, ad esempio, la vicinanza tra un punto di interesse ed un altro, in modo da evitare che il turista intraprenda un percorso di visita "inefficiente", effettuando così una **selezione** della scelta migliore.

Ogni attrazione è situata in un luogo chiamato "Punto di interesse" (esempio: Piazza, Museo, Teatro…) ed ogni Punto di interesse risiede in un luogo geografico (esempio: firenze)

I risultati saranno essenzialmente tre:

- Lista dei punti di interesse in ordine di percorso.
- Lista delle attrazioni presenti nei punti di interesse.
- Percorsi alternativi.

Di seguito si riporta la rete semantica relativa a questo progetto:

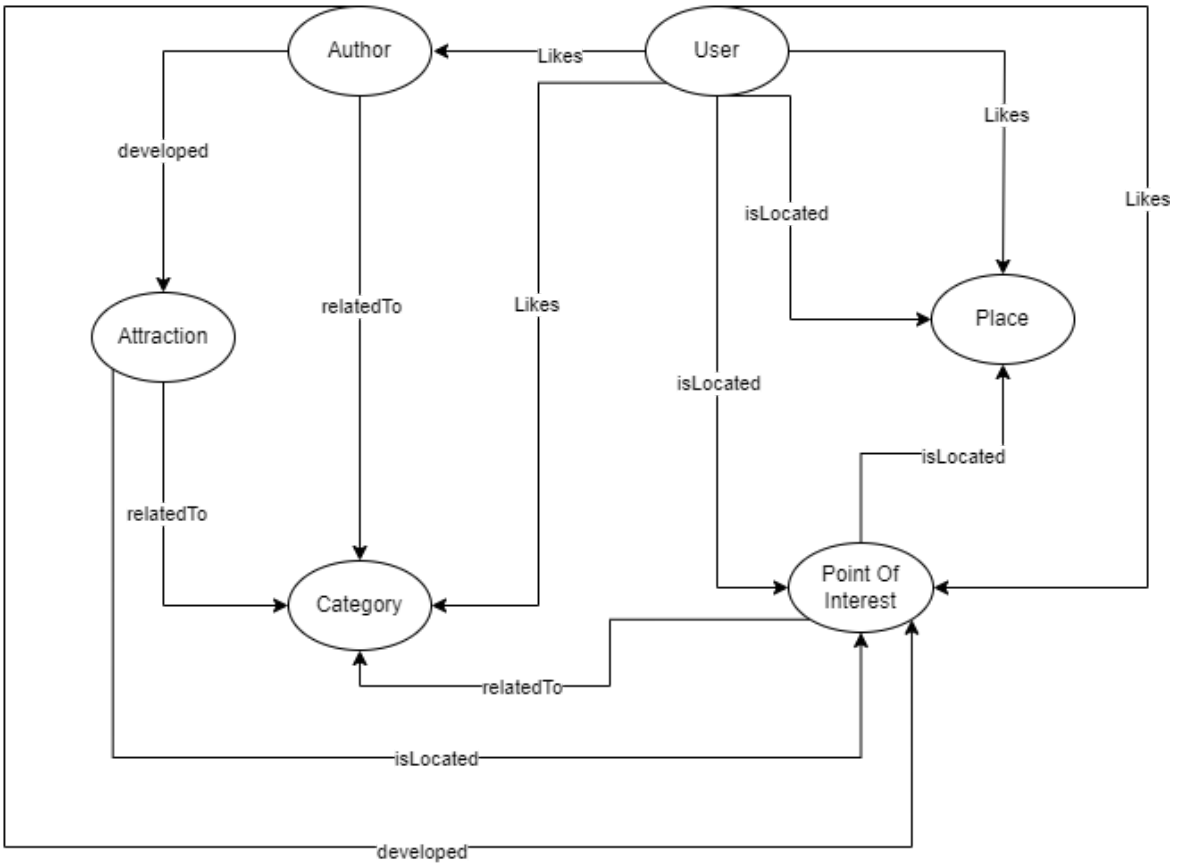


Figura 1 - Rete Semantica delle Entità

Analisi del caso di studio

Come già preannunciato, lo scopo del caso di studio in questione è quello di fornire una "Strategia di visita" all'utente in base alle sue esigenze. Per poter perseguire questo obiettivo è necessaria una knowledge base che raccolga tutte le informazioni del dominio quali:

- **Attrazioni**
  - Sculture
  - Opere
  - Dipinti
- **Luoghi**
- **User**
- **Autore**
- **Punti di interesse**
  - Musei
  - Teatri
  - Gallerie

- **Categorie**
    - Epoca
    - Movimento artistico

Si noti che le tipologie per ognuna di queste entità possono essere estese ulteriormente durante la realizzazione della knowledge base.

Le strategie per le sequenze si baseranno su:

1) **Interessi dell'utente**: Categorie, Autori e tipologia dei Punti di interesse.
2) **Prezzo:** l'intero percorso deve poter essere effettuato dall'utente con un determinato budget.
3) **Distanza Massima**: l'intero percorso deve coprire non più dei metri decisi dall'utente.

Uno dei motivi per cui alcuni punti di interesse potrebbero essere non incluse nella sequenza (oltre al fatto che non rispecchiano gli interessi dell'utente) è la distanza entro cui cercare i punti di interesse da visitare ed il limite di budget dell'utente in quanto alcuni punti di interesse potrebbero avere un costo.

Esempio: L'utente potrebbe essere interessato a visitare i punti di interesse con un budget di 50€ in un raggio di 2 km.

Componenti del Sistema Esperto

Ci sono una serie di funzioni comunemente utilizzate nei sistemi esperti. Alcune SHELL forniscono la maggior parte di queste funzionalità e altre solo alcune. Le SHELL personalizzate forniscono le caratteristiche più adatte al problema specifico. Le principali caratteristiche trattate in questo caso di studio sono:

- **Ragionamento ibrido** (backward-chaining & forward-chaining)
- **Rappresentazione dei dati**: il modo in cui sono rappresentati i dati specifici riguardo al problema, la loro memorizzazione all'interno della KB e il modo in cui vengono recuperati;
- **Interfaccia utente** – è la componente che permette di interagire con l'utente nelle operazioni di Input e di Output;
- **Spiegazioni** – la capacità del sistema di spiegare il processo di ragionamento con cui è giunto alla conclusione richiesta.

In particolare, relativamente a questo scenario, abbiamo ritenuto non necessaria la gestione della certezza. Infatti l'utente potrà decidere se gradisce un punto di interesse, autore, categoria e un posto o meno senza una gradualità. Tuttavia non escludiamo che questa feature potrebbe essere sviluppata in futuro.

I punti di interesse verranno poi trovati con formule che misurano la distanza quindi l'output sarà oggettivamente certo e non necessita di un grado di certezza.

In generale un sistema esperto possiede la seguente struttura:

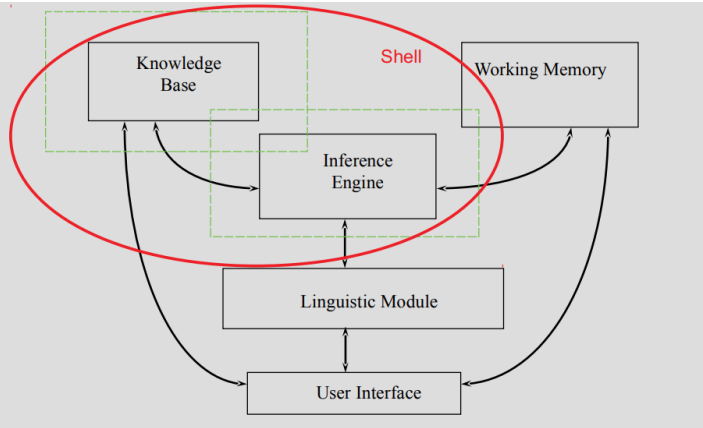


Figura 2 – Struttura di un sistema esperto

Si definiscono, di seguito, le feature che verranno utilizzate:

1) **Inference Engine:** Si intende sviluppare un sistema basato sull'uso del Backward-Chaining e Forward-Chaining (Hybrid) usando Prolog.
2) **Knowledge Base:** La conoscenza del sistema esperto risiede in due file separati, uno che conterrà i fatti e uno che conterrà le regole.
3) **User Interface:** Come interfaccia utente si è scelto di utilizzare la linea di comando di Prolog.
4) **Working Memory:** È la memoria ausiliare di Prolog, utilizzata tramite opportuni metodi (assert) per registrare le informazioni utili a perseguire il goal della costruzione del path di visita ottimale.
5) **Linguistic Module:** Modulo che conterrà i vari tipi domande/risposte in linguaggio naturale per rendere più user-friendly l'interazione con il sistema in questione.

Regole fondamentali per la costruzione del percorso

Di seguito sono riportate le regole principali al fine di ottenere un sistema esperto nella ricerca dei percorsi di visita:

1) Trovare i punti di interesse relativi agli interessi dell'utente.
2) Trovare i punti di interesse che hanno un prezzo che rispetta il budget dell'utente
3) Trovare i punti di interesse che rispettano la distanza dell'utente
4) Trovare il punto di interesse più vicino all'utente o ad un altro punto di interesse.
5) Trovare tutte le attrazioni relative ai punti di interesse per l'utente.
6) Trovare percorsi alternativi.

Realizzazione del Sistema Esperto

1. Concettualizzazione

La concettualizzazione è formalmente definita come una tripla < U, F, R > dove U è l'universo del discorso, F rappresenta le funzioni usate dagli oggetti del discorso mentre R rappresenta l'insieme delle relazioni tra questi ultimi.

In questa sezione si intende concretizzare ciò che è stato ideato nella fase introduttiva. Come primo passo si è deciso di descrivere l'Universo del discorso e quindi tutti gli elementi con i quali esprimiamo la nostra conoscenza.

1.1 Classi

Di seguito si riportano le classi che formano l'ontologia relativa al caso di studio, riportando una descrizione generale del significato assunto da ogni classe, ed i relativi attributi coinvolti dalle operazioni utili al perseguimento dell'obiettivo.

Nome	Descrizione	Attributi
User	Entità che rappresenta una User che può intraprendere una sequenza di visita. Ogni User è identificato da un UID e possiede nome e cognome.	UID: integer surname: string name: string
Attraction	Entità che rappresenta un'attrazione che si trova all'interno di un PointOfInterest. Ad ogni Attraction è possibile associare più di una categoria.	id: integer name: string type: string description: string date: date technique: string material: string
Place	Entità che rappresenta una città collocata all'interno di una regione e di uno Stato.	name: string region: string state: string
PointOfInterest	Entità relativa ai luoghi di interesse per il turista e che contiene delle attrazioni. Questa entità viene utilizzata per costruire le sequenze di visita da proporre allo User in base alle sue preferenze. Ad ogni PointOfInterest è possibile	id: integer name: string type: string description: string date: date price: real

Nome	Descrizione	Attributi
	associare più di una categoria.	
Category	Entità che rappresenta una categoria di riferimento per un'altra entità.	type:string name: string
Author	Entità che rappresenta un autore che ha sviluppato un PointOfInterest o una Attraction. Possono esserci più autori per ognuna di queste entità. Ad ogni autore può essere associata più di una categoria.	id: integer name: string birthDate: date deathDate: date birthPlace: entity deathPlace: entity nationality: string job: string

Ulteriori dettagli circa le relazioni tra queste entità ed i vincoli di esistenza

2. Relazioni tra le entità

Di seguito si riportano le relazioni tra le entità sopra descritte, grazie a cui è possibile derivare nuova conoscenza o verificare, ed eventualmente ritrattare, la conoscenza pregressa presente all'interno della knowledge base.

2.1 Relazioni

In questa sezione si riportano in maniera schematica le relazioni descritte dalla *Figura 1, Bozza della rete semantica,* a cui si aggiunge la presenza di eventuali attributi.

Nome: **isLocated**

Descrizione: Relazione che identifica l'inclusione fisica di un'entità all'interno di un'altra, permette di localizzare il soggetto per mezzo degli attributi Latitudine e Longitudine.

Attributi:

- Latitudine: real
- Longitudine: real

Soggetto	Oggetto
PointOfInterest	Place
User	Place
User	PointOfInterest

Soggetto	Oggetto
Attraction	PointOfInterest

Nome: **relatedTo**

Descrizione: Relazione che collega il soggetto ad una categoria di interesse. È possibile assegnare più di una categoria ad un soggetto.

Soggetto	Oggetto
Attraction	Category
PointOfInterest	Category

Nome: **developed**

Descrizione: Relazione che collega un autore con le sue creazioni.

Soggetto	Oggetto
Author	PointOfInterest
Author	Attraction

Nome: **likes**

Descrizione: Relazione che permette di stabilire cosa piace ad un dato utente.

Soggetto	Oggetto
User	Category
User	PointOfInterest
User	Place
User	Author

2.2 Assiomi

Di seguito sono riportati gli assiomi relativi alle classi parent:

- $\forall a \in PointOfInterest: \exists p \in Place: isLocated(a,\ p)$.
- $\forall a \in Attraction: \exists c \in Author:\ developed(a,\ c)$.
- $\forall a \in Attraction: \exists c \in PointOfInterest:\ isLocated(a,\ c)$.

3. Metodo di Ragionamento, Control Strategy e Search Strategy

3.1 Metodo di ragionamento

Per la realizzazione dell'obiettivo di questo caso di studio è stato utilizzato un **approccio ibrido** (Forward/Backward). Il sistema esperto utilizza i dati di partenza circa le preferenze dell'utente per risalire, in backward, ai punti di interesse che rispettano tali vincoli ed effettuare una ulteriore selezione basata sul prezzo e sulla distanza. Il risultato ultimo di questa selezione, che consiste nel punto di interesse più vicino a quello precedente, viene aggiunto alla sequenza di visita e, a partire da questo, l'algoritmo viene applicato ricorsivamente finché non viene raggiunta una condizione di stop, così come descritto nel paragrafo che segue.

Per questo motivo il nostro goal non è quello di dimostrare che un dato path, già conosciuto in partenza, sia quello ottimale, bensì consiste nel costruire il percorso ottimale mano a mano che i punti di interesse più vicini vengono *'scoperti'* ed aggiunti alla sequenza. Così facendo, la nuova conoscenza dedotta, nonchè il punto di interesse più vicino a quello precedente, viene utilizzata per dedurre quella futura, senza che sia noto in partenza il goal che si vuole dimostrare.

Pertanto l'inference engine di Prolog ed il backward chaining sono utili alla deduzione di informazioni che è possibile dimostrare risalendo direttamente ai fatti presenti nella knowledge base e che rappresentano il punto di partenza per la deduzione di ulteriore conoscenza, applicando regole ricorsive tipicamente backward.

Si rimanda ai paragrafi successivi per una visione più completa del problema e della relativa soluzione proposta in questo caso di studio.

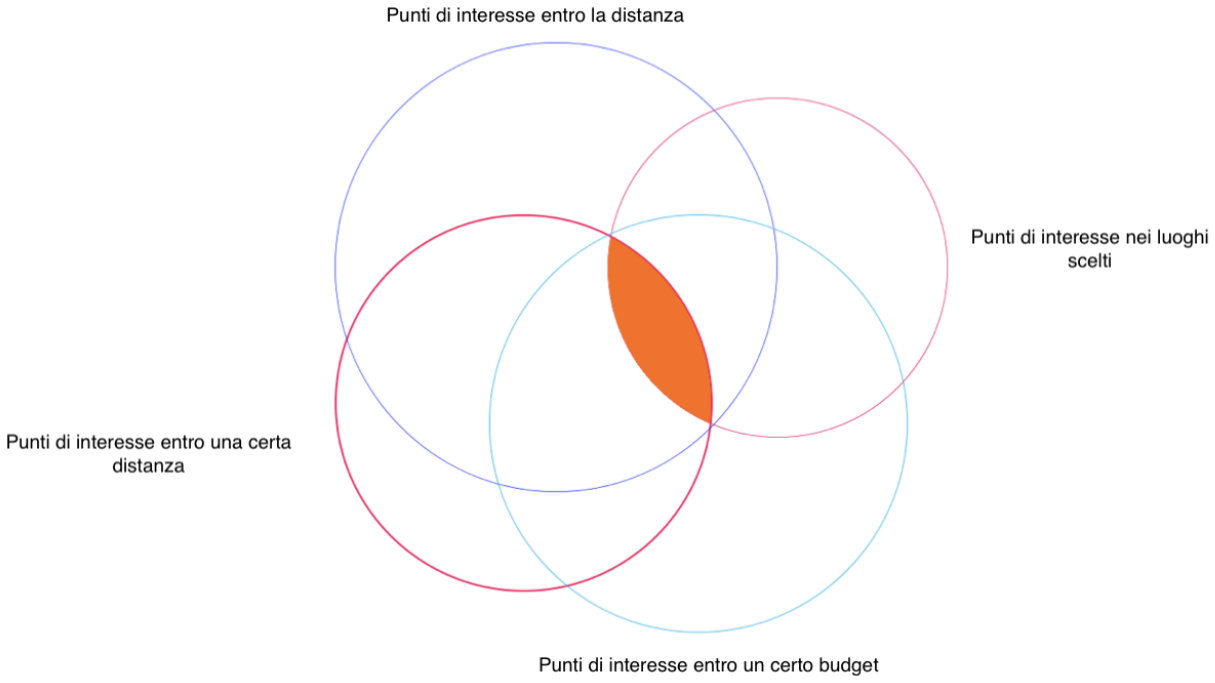
3.2 Algoritmo di costruzione del percorso ottimale

L'algoritmo di costruzione del percorso ottimale è al centro della discussione di questo caso di studio. Pertanto si intende porre particolare attenzione a questo argomento, evidenziando il ruolo e la funzione di ogni componente all'interno dell'algoritmo.

Nel paragrafo successivo, viene descritto l'algoritmo di costruzione dei percorsi alternativi che, affiancato all'obiettivo principale del caso di studio, permette di proporre all'utente tutti i possibili percorsi di visita che egli può intraprendere entro i vincoli da lui imposti.

Come primo step il sistema esperto raccoglie le informazioni dell'utente chiedendo:

1) Se preferisce delle categorie in particolare. (opzionale)
2) Se preferisce degli autori. (opzionale)
3) Se preferisce dei luoghi in particolare. (opzionale)
4) Se preferisce dei punti di interesse. (opzionale)
5) In che luogo si trova.
6) Il suo budget.
7) Distanza massima che vuole percorrere.



In questa figura è possibile osservare tutti i punti di interesse che rispettano i "quattro vincoli" e a partire dai quali verrà costruito il path ottimale. Poiché l'intersezione dei diversi insiemi può risultare in un insieme vuoto, l'intersezione viene effettuata solo tra insiemi non vuoti.

L'algoritmo di ricerca del path ottimale recupera la posizione dell'utente e cerca tutti i punti di interesse nel raggio di una distanza da lui definita, filtrando per budget ed eventualmente per luogo ed interessi.

Nella figura successiva, l'utente è descritto dal cerchio rosso mentre la freccia arancione indica la distanza massima che egli vorrà intraprendere.

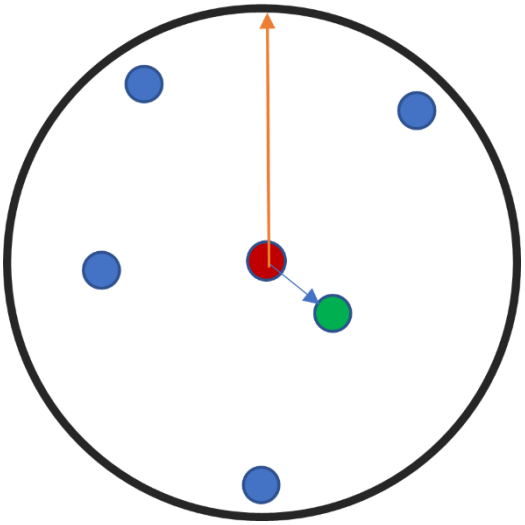


Figura 1 POI filtrate nel raggio d'azione dell'utente

Per calcolare la distanza tra due punti viene impiegata la formula dell'**emisenoverso**:

$$
\begin{aligned}
d &= 2r \arcsin\left(\sqrt{\operatorname{hav}(\varphi_2 - \varphi_1) + (1 - \operatorname{hav}(\varphi_1 - \varphi_2) - \operatorname{hav}(\varphi_1 + \varphi_2)) \cdot \operatorname{hav}(\lambda_2 - \lambda_1)}\right) \\
&= 2r \arcsin\left(\sqrt{\sin^2\left(\frac{\varphi_2 - \varphi_1}{2}\right) + \left(1 - \sin^2\left(\frac{\varphi_2 - \varphi_1}{2}\right) - \sin^2\left(\frac{\varphi_2 + \varphi_1}{2}\right)\right) \cdot \sin^2\left(\frac{\lambda_2 - \lambda_1}{2}\right)}\right) \\
&= 2r \arcsin\left(\sqrt{\sin^2\left(\frac{\varphi_2 - \varphi_1}{2}\right) + \cos\varphi_1 \cdot \cos\varphi_2 \cdot \sin^2\left(\frac{\lambda_2 - \lambda_1}{2}\right)}\right).
\end{aligned}
$$

[9]

- φ_1, φ_2 sono le latitudini dei due punti.
- λ_1, λ_2 sono le longitudini dei due punti.

Questa formula permette di calcolare la distanza in metri tra due punti sulla superficie terrestre a partire da due coppie di coordinate nella forma di Latitudine e Longitudine.

Come si evince dalla *Figura 1,* in questo raggio vi sono 5 punti di interesse che rientrano negli interessi dell'utente, tra queste viene selezionato il più vicino nello spazio come punto iniziale per l'algoritmo di ricerca del percorso ottimale.

Come passo seguente,l'algoritmo identifica il centro del "raggio d'azione" con il punto di interesse trovato in precedenza e il "raggio d'azione", nonchè la distanza residua che l'utente intende percorrere, viene aggiornato sottraendo alla distanza massima iniziale (freccia arancione in *Figura 1*) la distanza con il punto di interesse più vicino (freccia blu in *Figura 1*).

$$maximumDistance = maximumDistance - minimumDistance$$

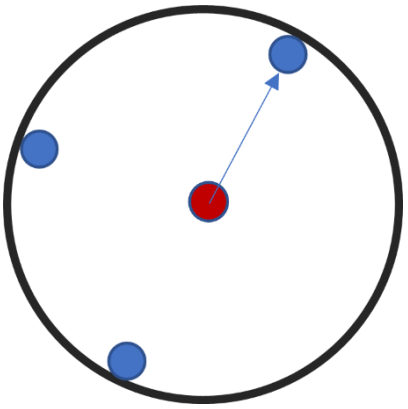


Figura 2 Point of interest nel raggio d'azione di un'attrazione

L'algoritmo procede in maniera ricorsiva, aggiungendo al percorso finale il punto di interesse più vicino ad ogni nuovo centro. Al fine di evitare la presenza di cicli, l'algoritmo esclude dalla ricerca i punti di interesse già presenti nel percorso ottimale.

La condizione di stop è dettata dall'assenza di punti di interesse, che rispettino i vincoli imposti dell'utente, all'interno del raggio d'azione di un centro.

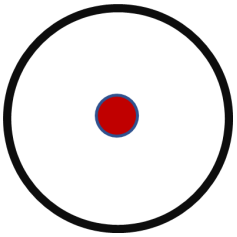


Figura 3 Stop Condition: nessun altra attrazione nelle vicinanze.

In *Figura 3* è possibile notare che nonostante ci sia ancora della distanza residua che l'utente potrebbe impiegare per il percorso, non vi sono altri punti di interesse nelle vicinanze del nuovo centro (condizione di stop) e l'algoritmo termina la ricerca.

Il risultato sarà una lista ordinata di pointOfInterest **FinalPath** = $[p_1, p_2, \dots, p_n]$, dove:

- p_1 è il punto di interesse più vicino all'utente,
- p_k, $1 < k \leq n$, è il punto di interesse più vicino a quello precedente p_{k-1}

3.1.1 Algoritmo di costruzione dei percorsi alternativi

L'algoritmo descritto fino a questo momento ottimizza il percorso basandosi principalmente sulla distanza, non permettendo all'utente di esplorare tutti gli altri possibili percorsi (non ottimali) che rispettano i vincoli da lui imposti. Per risolvere questa criticità si è deciso di implementare un algoritmo di ricerca dei percorsi non ottimali partendo dall'insieme iniziale di attrazioni vicine all'utente in un determinato raggio (*Figura 1*).

Si rimanda alla sezione 4.4.2 per visualizzare il risultato di questo algoritmo.

3.3 Pseudocodice - Path Ottimale

Di seguito è riportato lo pseudocodice dell'algoritmo riguardante il path ottimale per mostrare in maniera più dettagliata e ad alto livello, il suo funzionamento.

La relativa implementazione in Prolog è descritta al Paragrafo 4.

```
costruisciPercorsoOttimale(dismax, pos, categoria, autore, budget):
      posizione = pos;
      Percorso [] = null;
      distanzaMax = dismax;
      while c == true :
          poi [] = trovaPOIVicini(posizione, distanzaMax);
         PoiFiltrate[] = filtraPOI(poi [], budget, autore, categoria);
           if(length(POIFiltrati[]) > 0 && distanzaMax > 0) {
                poi = POIPiuVicino(posizione, PoiFiltrate[]);
                 distanzaMinima = dist(posizione, poi.pos);
                percorso.add(poi);
                budget = budget - poi.price;
                distanzaMax = distanzaMax - distanzaMinima;
                posizione = poi.position;
          }
          Else
               c = false;
return Percorso.


trovaPOIVicini(posizione, distanzaMax):
       poi = getAllPOIFromDB();
       poiVicini [] = null;
```

```
        foreach poi in attrazioni:
                if( distanza(posizione, poi.pos) <= distanzaMax )
                        poiVicini.add(poi);
        return poiVicini[];



filtraPOI(poi[], budget, autore, categoria):
        poiFiltrati = poi[];
        if (budget) {
                poiFiltrati = filtraBudget(poiFiltrati);
        }
        if (autore) {
                poiFiltrati = filtraAutore(poiFiltrati);
        }
        if (categoria) {
                poiFiltrati = filtraCategoria(poiFiltrati);
        }
        return poiFiltrati[];



POIPiuVicino(userPosition, poiFiltrati[]):
distList = []
foreach attr in poiFiltrati[]
        distList.add( distance(userPosition, poi ));
 }
return poiFiltrati[argmin(distList)];
```

argmin/1 restituisce l'indice dell'elemento con valore minimo all'interno del vettore dato.

distance/2 calcola la distanza tra due punti nello spazio.

4. Implementazione in Prolog

In questo Capitolo si riporta l'implementazione Prolog di quanto descritto fin'ora: le classi coinvolte, le relazioni fra di esse e le operazioni che manipolano questi dati, con l'obiettivo di consigliare all'utente il percorso ottimale e tutti i possibili percorsi che soddisfano i suoi interessi ed i vincoli di distanza massima e budget.

I moduli importati per l'utilizzo di regole di utility all'interno della knowledge base sono:

- :- use_module(library(lists)).
- :- use_module(library(assoc)).
- :- use_module(library(ordsets)).

4.1 Descrizione dei fatti

Di seguito si descrive la struttura dei fatti utilizzati all'interno della Knowledge Base per questo progetto.

Classi

- pointOfInterest(ID, NOME, TIPOLOGIA, DATA, DESCRIZIONE, _, PRICE( )).
- place(NOME, REGIONE, STATO).
- attraction(ID, NOME, TIPOLOGIA, DESCRIZIONE, DATA, TECNICA, MATERIALE).
- category(TIPOLOGIA, NOME_CATEGORIA).
- latlon(LATITUDINE, LONGITUDINE).
- user(ID, NOME, COGNOME).
- author(ID, NOME, NASCITA, MORTE, CITTA_NATALE, CITTA_MORTE, NAZIONE, LAVORO).

Relazioni

- developed
  - developed(author(-) , pointOfInterest(-)).
  - developed(author(-), attraction(-)).
- isLocated
  - isLocated(pointOfInterest(-), place(-), latlon(-))
  - isLocated(attraction(-), pointOfInterest(-), lat (-))
- relatedTo(pointOfInterest(-), category (-)).
- likes
  - likes(ID_USER, author(-)).
  - likes(ID_USER, category(-)).
  - likes(ID_USER, Place(-)).
  - likes(ID_USER, pointOfInterest(-)).

4.2 Regole e funzioni

L'intera implementazione è allegata al documento. In questa sezione vengono descritte le regole e le funzioni usate per soddisfare il goal principale della ricerca del path ottimale tra i punti di interesse.

La regola **main** è il punto di entrata del programma e si occupa di gestire le chiamate alle altre regole e di coordinare i parametri di input e di output tra di esse. Il **main** ha inizio con una interazione con l'utente che permette di raccogliere i dati relativi alle sue preferenze; si rimanda alla sezione *Interazione con l'utente per ulteriori dettagli*.

4.2.1 filteredPOI/3

Questa regola permette di filtrare tutti i punti di interesse all'interno della knowledge base, utilizzando le preferenze dell'utente riguardo alle categorie, autori, tipo di punto di interesse e prezzo.

INPUT:

- UID: integer, identificativo dell'utente che esegue il programma
- Price: real, budget a disposizione dell'utente per effettuare la visita

OUTPUT:

- Result: List, lista di punti di interesse che rientrano negli interessi dell'utente.

```
filteredPOI(UID,Price,Result):-
    interestedIn(1,ListCat,ListAuthor,ListPoi,ListPlace),
    findPOIByCat(ListC, ListCat),
    findPOIByAuth(ListA, ListAuthor),
    findPOIByType(Listpoi, ListPoi),
    findPOIByPlace(ListP,ListPlace),
    findPOIByPrice(Price,ListPr),
    selectPOI(ListC,ListA,ListP,Listpoi,ListPr,Result),
    assertRule(['\n\nHo filtrato le punti di interesse in base ai tuoi interessi tramite filteredPOI:\n', Result]).
```

4.2.2 interestedIn/5

Questa regola recupera le informazioni relative alle preferenze dell'utente, restituendo liste di interessi per ogni tipologia di preferenza (Categoria, Autore, Città, Tipo di attrazione).

INPUT

- UID: integer, rappresenta l'identificativo dell'utente di cui recuperare le preferenze.

OUTPUT

- ListCategory: List, rappresenta la lista di categorie che piacciono all'utente
- ListAuthor: List, rappresenta la lista di autori che piacciono all'utente
- ListPlace: List, rappresenta la lista di città che piacciono all'utente
- ListPOI: List, rappresenta la lista di tipologie di punti di interesse che piacciono all'utente

```
%MODIFIED
interestedIn(UID,ListCat,ListAuthor,ListPoi,ListPlace) :-
    findall( C1,(likes(UID,category(_,C1)), nonvar(C1)),ListCat),
    findall( NameAuthor,(likes(UID,author(_,NameAuthor,_,_,_,_,_,_)),nonvar(NameAuthor)),ListAuthor),
    findall( CatPoi,(likes(UID,pointOfInterest(_,X,CatPoi,_,_,_,_)),nonvar(CatPoi)),ListPoi),
    findall( NamePlace,(likes(UID,place(NamePlace,_,_)), nonvar(NamePlace)),ListPlace),
    assertRule(['\n\nHo trovato tutti i nomi delle punti di interesse collegate ai tuoi interessi tramite la regola interestedIn: \n']).
```

4.2.3 findPOIbyCat/2

Questa regola ricorsiva restituisce tutti i punti di interesse che rispettano le preferenze dell'utente riguardo le categorie

INPUT

- ListCat: List, rappresenta la lista contenente i nomi delle categorie che piacciono all'utente

OUTPUT

- Result: List, rappresenta la lista di tutti i punti di interesse che rispettano le preferenze dell'utente riguardo le categorie

```
findPOIByCat([],[]).

findPOIByCat(ListAtt, [H|T]):-
    findall(X,(pointOfInterest(_,X,_,_,_,_,_), relatedTo(pointOfInterest(_,X,_,_,_,_,_),category(_,H))), List ),
    findPOIByCat(L1,T),
    append(L1,List,ListAtt).
```

4.2.4 findPOIbyAuth/2

Regola ricorsiva utile a trovare tutti i punti di interesse che fanno riferimento agli autori preferiti dell'utente.

INPUT

- ListAuthr:List, rappresenta la lista contenente i nomi degli autori che piacciono all'utente

OUTPUT

- Result: List, rappresenta la lista di tutti i punti di interesse che rispettano le preferenze dell'utente riguardo gli autori

```
findPOIByAuth([],[]).


%MODIFIED
findPOIByAuth(ListAtt, [H|T]):-
    findall(X,(developed(author(_, H, _, _, _, _, _, _), pointOfInterest(_, X, _, _, _ , _,_) )), List1 ),
    findall(X,(developed(author(_, H, _, _, _, _, _, _), attraction(_, X, _, _, _ , _,_) )), List2 ),
    findPOIFromAttr(ListPOI,List2),
    merge(ListPOI,List1,List),
    findPOIByAuth(L1,T),
    append(L1,List,ListAtt).
```

4.2.5 findPOIbyType/2

Regola ricorsiva utile a trovare tutti i punti di interesse relativi "tipo" di queste ultime (esempio: l'utente preferisce musei e teatri).

INPUT

- ListPOI: List, rappresenta la lista contenente i nomi delle tipologie di punti di interesse.

OUTPUT

- Result: List, rappresenta la lista di tutti i punti di interesse che rispettano le preferenze dell'utente riguardo la tipologia dei punti di interesse.

```
findPOIByType([],[]).

%MODIFIED
findPOIByType(ListAtt, [H|T]):-

    findall(X,pointOfInterest(_,X,H,_,_,_,_) , List ),
    findPOIByType(L1,T),
    append(L1,List,ListAtt).
```

4.2.6 findPOIbyPlace/2

Regola ricorsiva utile a trovare tra tutti i punti di interesse, quelli che fanno parte di luoghi geografici definiti dall'utente (esempio: Bari, Firenze).

INPUT

- ListPlace: List, rappresenta la lista contenente i nomi delle città che piacciono all'utente

OUTPUT

- Result: List, rappresenta la lista di tutti i punti di interesse che rispettano le preferenze dell'utente riguardo alle città.

```
findPOIByPlace([],[]).

findPOIByPlace(ListAtt, [H|T]):-
    findall(X,(isLocated(pointOfInterest(_,X,_,_,_,_,_),place(H,_,_), latlon(_,_)) ), List ),
    findPOIByPlace(L1,T),
    append(L1,List,ListAtt).
```

4.2.7 findPOIbyPrice/2

Regola utile al filtraggio dei punti di interesse in base al prezzo, utilizzando come soglia il budget residuo dell'utente.

INPUT

- ListPlace: List, rappresenta la lista contenente i nomi delle città che piacciono all'utente

OUTPUT

- Result: List, rappresenta la lista di tutti i punti di interesse che rispettano le preferenze dell'utente riguardo alle città.

```
findPOIByPrice(Price,List):-
    findall(X,(pointOfInterest(_,X,_,_,_,_,price(P)), P =< Price ),List).
```

4.2.8 selectPOI/6

Regola che seleziona i punti di interesse prendendo in input le liste prodotte dalle regole ed esegue i seguenti step:

1) Effettua un merge tra tutte le liste dei punti di interesse relativi all'autore, tipologia, categoria.
2) Effettua un merge tra la lista risultante dal punto 1) e la lista delle attrazioni in base ad un certo prezzo. Questo significa che nella lista risultante vi saranno POI che rispettano gli interessi e POI che rispettano solamente il budget.
3) La lista allo step 2) viene intersecata con la lista di tutti i punti di interesse sotto un certo budget.
4) La lista allo step 3) viene intersecata con la lista delle attrazioni nei luoghi geografici scelti dall'utente (nel caso in cui siano stati esplicitati).
5) la lista ottenuta allo step 4) viene intersecata con la lista ottenuta al punto 1) in modo da ottenere i punti di interesse che soddisfano pienamente l'utente.

```
selectPOI(LCat,LAuth,LPlace,LPOI,LPrice,Result):-
    merge(LCat,LAuth,R1),
    merge(R1,LPOI,R3),
    merge(R3,LPrice,R4),
    remove_duplicates(R4,R5),
    intersection(R5,LPrice,R6),
    intersectPlace(R6,LPlace,ResultPlace),
    intersectAll(ResultPlace,R3,Result).
```

4.2.8.1 intersectPlace/3

Questa regola prende in input la lista ottenuta dall'unione dei nomi dei punti di interesse che rispettano le preferenze dell'utente e la interseca con la lista ottenuta da **findPOIByPlace**, contenente tutti i nomi dei punti di interesse presenti all'interno delle città che piacciono all'utente, così da ottenere come risultato una nuova lista che contiene solamente i punti di interesse che possono piacere all'utente in una determinata città.

INPUT

- ListInput: List, rappresenta la lista contenente i nomi dei punti di interesse che rispettano le preferenze dell'utente.
- LPlace: List, rappresenta la lista contenente i nomi dei punti di interesse presenti all'interno delle città scelte dall'utente. Questa lista agisce da *'filtro'* per la lista **ListInput**.

OUTPUT

- Result: List, rappresenta la lista ottenuta filtrando la lista di input **ListInput** con la lista LPlace.

```
intersectPlace(ListInput, LPlace, Result):-

    (length(LPlace,L1 ), L1 > 0
    ->
    (subtract(ListInput, LPlace,Sub1),
    subtract(ListInput,Sub1,Result))
    ;
    Result = ListInput).
```

4.2.8.2 intersectAll/3

Questa regola prende in input la lista ottenuta in output da **intersectPlace** e la '*filtra*', ottenendo in output una nuova lista che considera solamente i punti di interesse per le città selezionate e che rispettano le preferenze dell'utente in termini di categorie, autori e tipologie dei punti di interesse stessi.

INPUT

- ResultPlace: List, rappresenta la lista contenente i nomi dei punti di interesse che rispettano le preferenze dell'utente.
- R3: List, rappresenta la lista contenente i nomi dei punti di interesse presenti all'interno delle città scelte dall'utente. Questa lista agisce da *'filtro'* per la lista **ListInput**.

OUTPUT

- Result: List, rappresenta la lista ottenuta filtrando la lista di input **ListInput** con la lista LPlace.

```
intersectAll(ListInput, ListAll, Result):-

    (length(ListAll,L1 ), L1 > 0
    ->
    (subtract(ListInput, ListAll,Sub1),
    subtract(ListInput,Sub1,Result))
    ;
    Result = ListInput).
```

4.2.9 aroundUser/5

Regola ricorsiva che trova tutti i punti di interesse significativi attorno all'utente, così come descritto nel paragrafo 3.1. Questa regola fornisce in output due liste: una contiene le distanze tra l'utente e i punti di interesse entro la distanza residua, e l'altra i nomi dei punti di interesse corrispondenti.

INPUT

- UID: integer, rappresenta l'identificativo dell'utente di cui recuperare le preferenze.
- ListFiltered: List, rappresenta la lista contenente i nomi dei punti di interesse filtrati da **selectPOI**
- MaxDist: real, rappresenta la distanza massima da percorrere nella sequenza di visita.

OUTPUT

- ListDist: List, rappresenta la lista delle distanze
- ListNames: List, rappresenta la lista ottenuta filtrando la lista di input **ListInput** con la lista LPlace

```
aroundUser(_,[],_,[],[]).

aroundUser(UID, ListFiltered, MaxDist, ListDist,ListNames):-
    remove_duplicates(ListFiltered,List),
    reverse(List,List1),
    subtract(List1,[[]],ListPOI), %%rimuoviamo lista vuota per fare in modo che calcDist non cerchi una attrazione con nome [] ==> impediamo il fallimento di calcDist
    calcDistUser(UID, ListPOI, ListD),

    findall(El, (select(poidist(_,El),ListD,Residuo), El < MaxDist), ListDist),
    findall(N, (select(poidist(N,El),ListD,Residuo), El < MaxDist), ListNames),

    assertRule(['\n\nHo trovato le punti di interesse più vicine a te con la regola aroundUser: \n',ListNames,\n,'Le relative distanze sono: \n',ListDist]).
```

.

4.2.10 calcDistUser/3

Funzione ricorsiva che calcola la distanza tra l'utente e tutti i punti di interesse forniti in input, utilizzando la formula dell'emisenoverso.

INPUT

- UID: integer, rappresenta l'identificativo dell'utente
- ListPOI: List, rappresenta la lista di punti di interesse su cui effettuare il calcolo della distanza

OUTPUT

- ListDist: List, rappresenta la lista delle distanze tra i punti di interesse e l'utente

```
calcDistUser(_,[],[]).


calcDistUser(UID,[H|T],ListDist):-
    getDist(user(UID,_,_),pointOfInterest(_,H,_,_,_,_,_), Dist),
    calcDistUser(UID,T,LD1),
    append(LD1,[poidist(H,Dist)],ListDist).
```

4.2.11 findPath/5

Questa regola ricorsiva calcola il path ottimale partendo dal punto di interesse più vicino all'utente e che rispetta tutti i vincoli imposti da esso, proseguendo fino ad una condizione di stop secondo la strategia descritta nel *paragrafo 3.1*.

INPUT

- UID: integer, rappresenta l'identificativo dell'utente
- Budget: real, rappresenta il budget residuo per la ricerca dei punti di interesse
- ResidualDist: real, rappresenta la distanza residua per la ricerca dei punti di interesse
- CenterPOI: string, rappresenta il nome del punto di interesse utilizzato come centro per la ricerca dei vicini entro il raggio definito da **ResidualDist.**

OUTPUT

- FinalPath: List, rappresenta la lista ordinata contenente il path di visita ottimale per la configurazione scelta dall'utente.

```
findPath(_,[],_,_,[H|T]):-
    !,
    subtract([H|T], [[]], Res),
    assertRule(['\n\nAlla fine, utilizzando la regola findPath, il percorso ottimale per le tue preferenze è: \n', Res]),
    expandAttractions(Res,List),
    assertRule(['\n\n Allo interno di questi punti di interesse puoi ammirare queste punti di interesse:\n\n', List]).


findPath(UID, CenterPOI, ResidualDist, Budget, FinalPath ):-

    filteredPOI(UID, Budget, ResFil ),
    subtract(ResFil, [CenterPOI], ResFiltered),

    aroundPOI(CenterPOI, ResFiltered, ResidualDist, ListDist,ListNames,FinalPath),
    nearestPOI(ListDist,ListNames, NewCenter,NewMin),

    pointOfInterest(_,NewCenter,_,_,_,_,price(PricePOI)),

    NewBudget is Budget - PricePOI,
    NewDist is ResidualDist - NewMin,

    append(FinalPath, [NewCenter], Result),

    findPath(UID, NewCenter, NewDist, NewBudget, Result).
```

4.2.12 aroundPOI/6

Questa regola ricorsiva viene utilizzata da **findPath** per capire quali sono i punti di interesse attorno ad un altro, entro il raggio definito da MaxDist, escludendo quelli già presenti all'interno del path ottimale per evitare che si presentino cicli.

INPUT

- POIName: string, rappresenta il nome del punto di interesse utilizzato come centro per la ricerca di altri punti di interesse entro il raggio definito da **MaxDist**
- ListFiltered: List, rappresenta la lista di attrazioni filtrate secondo le preferenze dell'utente; tra queste sono selezionate solo quelle che rientrano nel raggio d'azione specificato.
- MaxDist: real, rappresenta il raggio di azione in cui effettuare la ricerca dei punti di interesse.
- FinalPath: List, rappresenta la lista contenente il path ottimale elaborato fino a quel momento. Gli elementi di questa lista sono esclusi dalla ricerca delle attrazioni intorno al centro per evitare che si creino cicli.

OUTPUT

- ListDist: List, rappresenta la lista di distanze tra i punti di interesse ed il centro **POIName,** in un raggio specificato da **MaxDist.**
- ListNames : List, rappresenta la lista di nomi corrispondenti alle distanze presenti nella lista **ListDist**

```
aroundPOI([],_,_,[],[],[]).

%%passiamo il final path per eslcuderne gli elementi ed evitare che ci siano cicli

aroundPOI(POIName, ListFiltered, MaxDist, ListDist,ListNames,FinalPath):-
    remove_duplicates(ListFiltered,List),
    reverse(List,List1),
    subtract(List1,[[]],ListPOI), %%rimuoviamo lista vuota per fare in modo che calcDist non cerchi una attrazione con nome [] ==> impediamo il fallimento di calcDist
    calcDistPOI(POIName, ListPOI, ListD),
    findall(El, (select(poidist(N,El),ListD,Residuo), El < MaxDist ,\+member(N,FinalPath) ), ListDist),
    findall(N, (select(poidist(N,El),ListD,Residuo), El < MaxDist, \+member(N,FinalPath) ), ListNames),
    assertRule(['\n\nTramite aroundPOI, ho trovato le punti di interesse intorno a ', POIName, ' nel raggio di ', MaxDist, '.\n
    Le punti di interesse sono: \n',ListNames,\n,'Le relative distanze sono: \n',ListDist]).
```

4.2.13 nearestPOI/4

Questa funzione ricorsiva, utilizzata da **findPath**, seleziona dalla lista **ListDist** l'elemento con distanza minima, utilizzando il suo indice

INPUT

- ListDist: List, rappresenta la lista delle distanze tra l'attuale centro ed i punti di interesse
- ListNames: List rappresenta la lista dei nomi delle attrazioni corrispondenti alle distanze

OUTPUT

- NewCenter
- NewMin.

```
nearestPOI([],[],[],0.0).


nearestPOI(ListDist,ListNames,NearestPOI,Min):-
    min_list(ListDist,Min),
    nth1(Nth, ListDist,Min),
    nth1(Nth,ListNames,NearestPOI),
    assertRule(['\n\nHo utilizzato la regola nearestPOI per trovare la più vicina' ,NearestPOI, ' con distanza:', Min,'\n']).
```

4.2.14 alternativePath/5

regola che costruisce, in maniera ricorsiva, tutti i path alternativi che vengono costruiti rimuovendo, dall'OptimalPath, la testa e cercando tra i punti di interesse rimanenti, quello più vicino che verrà utilizzato per il percorso alternativo.

INPUT

- UID: integer, identificativo univoco dell'utente
- Price: real, Budget residuo per la visita
- MaxDist: real, distanza residua per la costruzione della sequenza di visita

- AltDist: List. Lista che contiene le distanze relative ai punti di interesse
- AltPoi: List. Lista che contiene i nomi dei punti di interesse relativi alle distanze

```
alternativePath(_,_,_,[],[]).

alternativePath(UID,Price,MaxDist,[D|TD],[N|TN]):-

    nearestPOI([D|TD],[N|TN], InitialPOI,Min),

    subtract([D|TD],[Min],AltDist),
    subtract([N|TN],[InitialPOI],AltAttr),

    pointOfInterest(_,InitialPOI,_,_,_,_,price(PricePOI)),

    InitPrice is Price - PricePOI,
    InitDist is MaxDist-Min,

    append([],InitialPOI,FinalPath),

    findPath(UID,InitialPOI,InitDist, InitPrice,[FinalPath]),


    alternativePath(IUD,Price,MaxDist,AltDist,AltAttr).
```

4.2.15 expandAttractions/2

Questa regola ricorsiva viene utilizzata dopo aver trovato il path ottimale con lo scopo di mostrare all'utente tutte le attrazioni contenute all'interno dei punti di interesse facenti parte del percorso.

INPUT

- FinalPath: List, rappresenta la lista contenente il path ottimale finale.

OUTPUT

- Res: List, rappresenta la lista che contiene tutte le attrazioni contenute all'interno dei punti di interesse facenti parte di **FinalPath.**

Esempio:

- **FinalPath:** [quadreria,teatro petruzzelli,teatro margherita,museo civico]
- **Res:** [collezione fondo tanzi,collezione fondo menotti,collezione fondo antonelli,collezione di armi,collezione campagna di africa]

```
expandAttractions([],[]).

expandAttractions([H|T],Res):-
    findall(X,isLocated(attraction(_,X,_,_,_,_,_), pointOfInterest(_,H,_,_,_,_,_),_),List),
    expandAttractions(T,L1),
    append(L1,List,Res).
```

4.2.16 emisenoverso/5

Questa funzione permette di calcolare la distanza tra due punti utilizzando come unità di misura i metri e prendendo in considerazione il piano sferico della terra.

INPUT

- Lat1, Lon1, Lat2, Lon2: Latitudine e Longitudine dei due rispettivi punti

OUTPUT

- Dist: distanza in metri tra i due punti

```
emisenoverso(Lat1,Lat2,Lon1,Lon2,Dist):-
    R is 6371*10**3,
    Phi1 is Lat1*pi/180,
    Phi2 is Lat2*pi/180,
    Delta_phi is (Lat2-Lat1)*pi/180,
    Delta_lambda is (Lon2-Lon1)*pi/180,
    A is (sin(Delta_phi/2)*sin(Delta_phi/2) + cos(Phi1)*cos(Phi2)*sin(Delta_lambda/2)*sin(Delta_lambda/2)),
    C is 2*atan2(sqrt(A),sqrt(1-A)),
    Dist is R*C.
```

4.3 Interazione con l'utente

Le regole qui descritte memorizzano all'interno della working memory e del file 'new_fact.pl' le preferenze dell'utente utilizzando il seguente predicato: ***likes(UID, Something).***

Dove:

- UID: integer, corrisponde all'identificativo univoco dell'utente.
- Something: Entity, corrisponde al tipo di entità che piace all'utente.

Esempio: Per memorizzare il fatto che all'utente con UID = 1 piace la categoria *'rinascimento'* il predicato likes sarà di questa forma: ***likes(1, category(period,rinascimento)).***

4.3.1 menuCat e askCategory

Utile per chiedere all'utente *n* categorie che preferisce per i punti di interesse.

4.3.3 menuPoi e askPOI

Utile per chiedere all'utente *n* tipologie che preferisce per i punti di interesse.

4.3.5 menuPlace e askPlace

Utile per chiedere all'utente *n* luoghi geografici che preferisce per i punti di interesse.

4.3.7 menuAuthor e askAuthor

Utile per chiedere all'utente *n* autori che preferisce per i punti di interesse.

```
menuCat :-
    write('Vuoi selezionare delle categorie di tuo interesse?:[[y/n]]'), nl,
    read(y)
    -> askCategory
    ;true.

menuPlace :-

    write('Vuoi selezionare delle citta di tuo interesse?:[[y/n]]'), nl,
    read(y)
    -> askPlace
    ;true.



menuPoi:-
    write('Vuoi selezionare specifiche tipologie di luoghi?:[y/n]]'), nl,
    read(y)
    -> askPOI
    ;true.


menuAuthor:-
    write('Vuoi selezionare un autore di tuo interesse?:[[y/n]]'), nl,
    read(y)
    -> askAuthor
    ;true.
```

4.3.9 askBudget

Utile per chiedere all'utente il suo budget disponibile per l'intera visita.

4.3.10 askRadius

Utile per chiedere all'utente la distanza che vuole ricoprire con il suo percorso di visita.

Di seguito sono riportate le regole relative agli ask:

```
askCategory :-
    repeat,
    writeln('Seleziona una categoria di tuo interesse:'), nl,
    findall(N,relatedTo(Entity, category(_,N)), ML),
    remove_duplicates(ML,MenuList),
    write(MenuList),nl,
    read(X),
    (   member(X,MenuList)
    ->  write('Hai selezionato: '), write(X), assertLikes(1,category(period,X)), nl, !
    ;   write('Immetti un nome valido'), nl, fail
    ),
    writeln('Vuoi selezionare altro?: [y/n]'),
    (read(y)
    -> askCategory
    ; true).



askPlace :-
    repeat,
    writeln('Seleziona una citta:'), nl,
    findall(N, (isLocated(_, place(N,_,_),_),nonvar(N)), ML),
    remove_duplicates(ML,MenuList),
    write(MenuList),nl,
    read(X),
    (   member(X,MenuList)
    ->  write('Hai selezionato: '), write(X), assertLikes(1,place(X,_,_)), nl, !
    ;   write('Immetti un nome valido'), nl, fail
    ),
    writeln('Vuoi selezionare altro?: [y/n]'),
    (read(y)
    -> askPlace
    ; true).
```

```
askPOI :-
    repeat,
    writeln('Seleziona una tipologia di punto di interesse:'), nl,
    findall(N, (pointOfInterest(_,_,N,_,_,_,price(_)) , nonvar(N)), ML),
    remove_duplicates(ML,MenuList),
    write(MenuList),nl,
    read(X),
    write('Hai selezionato: '), write(X), assertLikes(1,pointOfInterest(_,_,X,_,_,_,_)), nl, !,
    writeln('Vuoi selezionare altro?: [y/n]'),
    (read(y)
    *-> askPOI
    ; true).

askAuthor :-
    repeat,
    writeln('Seleziona un autore:'), nl,
    findall(N, ( developed(author(_,N, _, _, _, _, _, _),_),nonvar(N)), ML),
    remove_duplicates(ML,MenuList),
    write(MenuList),nl,
    read(X),
    write('Hai selezionato: '), write(X), assertLikes(1,author(_,X,_,_,_,_,_,_)), nl, !,
    writeln('Vuoi selezionare altro?: [y/n]'),
    (read(y)
    *-> askAuthor
    ; true).



askBudget :-
    repeat,
    writeln('Inserisci un costo massimo per la tua visita:'), nl,
    read(X),
    ( number(X)
    ->  write('Hai selezionato: '), write(X), assert(budget(X)), nl, !
    ;   write('Immetti un valore valido'), nl, fail
    ).



askRadius :-
    repeat,
    writeln('Inserisci un raggio di azione per la ricerca delle punti di interesse (in metri):'), nl,
    read(X),
    ( number(X)
    ->  write('Hai selezionato: '), write(X), assert(radius(X)), nl, !
    ;   write('Immetti un valore valido'), nl, fail
    ).
```

4.4 Explainability

Tutte le spiegazioni riguardo il processo seguito dal sistema esperto verranno salvate nel file *explanation.txt* per mezzo della regola **assertRule.** La regola assertRule prende in input una lista di stringhe e la scrive all'interno dell'output corrente. Il file presenta, in linguaggio naturale, tutte le regole che sono state coinvolte al fine di produrre i path, mostrando anche distanze e point of interest ottenuti ad ogni passo dell'algoritmo ricorsivo. Nella parte finale, invece, è possibile trovare il percorso ottimale sotto forma di lista.

Il file *alternative-path.txt* invece, contiene i path alternativi che l'utente può seguire e le relative spiegazioni.

4.4.1 Esempio explainability per il path ottimale

Di seguito si riporta il risultato ottenuto dall'algoritmo dopo aver scelto di visitare punti di interesse di Bari, di categoria 'Contemporaneo', nel raggio di 1500 metri e con un budget di 76 Euro.

Ho trovato tutti i nomi dei punti di interesse collegate ai tuoi interessi tramite la regola **interestedIn:**

Ho filtrato i punti di interesse in base ai tuoi interessi tramite **filteredPOI:**

[museo civico,museo nuova era,museo nicolaiano,teatro margherita,teatro petruzzelli,kunsthalle,quadreria,galleria arcieri]

Ho trovato i punti di interesse più vicine a te con la regola **aroundUser:**

[museo civico,museo nuova era,museo nicolaiano,teatro margherita,teatro petruzzelli,kunsthalle,quadreria,galleria arcieri]

Le relative distanze sono:

[857.303415104891,1075.39630595537,1243.18196949954,975.612884008533,818.227777558972,1048.93933772602,483.147966670544,1042.16582573515]

Ho utilizzato la regola **nearestPOI** per trovare la più vicina quadreria con distanza:483.147966670544

Inizio il calcolo del percorso ottimale con prezzo iniziale di 76 e distanza rimanente di 1016.85203332946.

Ho trovato tutti i nomi dei punti di interesse collegate ai tuoi interessi tramite la regola **interestedIn:**

Ho filtrato i punti di interesse in base ai tuoi interessi tramite **filteredPOI:**

[museo civico,museo nicolaiano,teatro margherita,teatro petruzzelli,galleria arcieri]

Tramite **aroundPOI,** ho trovato i punti di interesse intorno a quadreria nel raggio di 1016.85203332946.

Le punti di interesse sono:

[museo civico,museo nicolaiano,teatro margherita,teatro petruzzelli,galleria arcieri]

Le relative distanze sono:

[538.689351750685,930.965562532439,538.414356795986,335.082907172544,895.790769330561]

Ho utilizzato la regola **nearestPOI** per trovare la più vicina teatro petruzzelli con distanza:335.082907172544

Ho trovato tutti i nomi dei punti di interesse collegate ai tuoi interessi tramite la regola interestedIn:

Ho filtrato i punti di interesse in base ai tuoi interessi tramite **filteredPOI:**

[museo civico,museo nicolaiano,teatro margherita,teatro petruzzelli,galleria arcieri]

Tramite **aroundPOI,** ho trovato i punti di interesse intorno a teatro petruzzelli nel raggio di 681.769126156912.

I punti di interesse sono:

[museo civico,teatro margherita]

Le relative distanze sono:

[504.026619156083,322.256232409747]

Ho utilizzato la regola **nearestPOI** per trovare la più vicinateatro margherita con distanza:322.256232409747

Ho trovato tutti i nomi dei punti di interesse collegate ai tuoi interessi tramite la regola interestedIn:

Ho filtrato i punti di interesse in base ai tuoi interessi tramite **filteredPOI**:

[museo civico,teatro margherita,teatro petruzzelli,galleria arcieri]

Tramite **aroundPOI**, ho trovato i punti di interesse intorno a teatro margherita nel raggio di 359.512893747165.

Le punti di interesse sono:

[museo civico]

Le relative distanze sono:

[292.301341353526]

Ho utilizzato la regola **nearestPOI** per trovare la più vicinamuseo civico con distanza:292.301341353526

Ho trovato tutti i nomi dei punti di interesse collegate ai tuoi interessi tramite la regola interestedIn:

Ho filtrato i punti di interesse in base ai tuoi interessi tramite **filteredPOI**:

[museo civico,teatro margherita,teatro petruzzelli,galleria arcieri]

Tramite **aroundPOI**, ho trovato i punti di interesse intorno a museo civico nel raggio di 67.2115523936394.

I punti di interesse sono:

[]

Le relative distanze sono:

[]

Alla fine, utilizzando la regola findPath, il percorso ottimale per le tue preferenze è:

[quadreria,teatro petruzzelli,teatro margherita,museo civico]

All'interno di questi punti di interesse puoi ammirare queste attrazioni:

[collezione fondo tanzi,collezione fondo menotti,collezione fondo antonelli,collezione di armi,collezione campagna di africa]

L'algoritmo trova la sequenza di visita ottimale corrispondente a **[quadreria,teatro petruzzelli,teatro margherita,museo civico]** ed in seguito per ognuno di questi punti di interesse mostra le attrazioni che è possibile vedere al loro interno.

Come è possibile notare dalle ultime righe del testo, l'algoritmo termina quando la regola **aroundPOI** riporta una lista vuota di punti di interesse intorno al centro considerato.

4.4.2 Esempio explainability per i path alternativi

Poichè l'expainability per i path alternativi presenta la stessa struttura di quella del path ottimale, si è deciso di omettere l'intero output del file *'alternative-paths'* e, per brevità, si riportano solamente i path alternativi (non ottimali) ottenuti dalla regola **alternativePath.**

Di seguito si riportano i path alternativi ottenuti utilizzando la stessa configurazione precedente, scegliendo come punti di interesse quelli di Bari, di categoria 'Contemporaneo', nel raggio di 1500 metri e con un budget di 76 Euro.

1) **[teatro petruzzelli,teatro margherita,museo civico]**
2) **[museo civico,museo nuova era,teatro margherita]**
3) **[teatro margherita,museo civico,museo nuova era]**
4) **[galleria arcieri]**
5) **[kunsthalle,teatro petruzzelli]**
6) **[museo nuova era,museo civico]**
7) **[museo nicolaiano]**

Come è possibile notare, l'algoritmo ottiene ben sette percorsi alternativi che l'utente può intraprendere, sulla base delle sue preferenze. Può capitare che alcuni dei percorsi alternativi, come nel caso dei percorsi **2)** e **3),** consistano degli stessi punti di interesse ma in ordine diverso.

5. Risultati

Nell'immagine sottostante è rappresentato il percorso suggerito dal nostro sistema esperto immettendo come latitudine e longitudine: 41.12374194325008, 16.867298368953858,

Le attrazioni sono state trovate rispettando il budget e la distanza massima (in questo caso non più di 2 km) e cercando tutte quelle attrazioni che possono essere di interesse per l'utente.

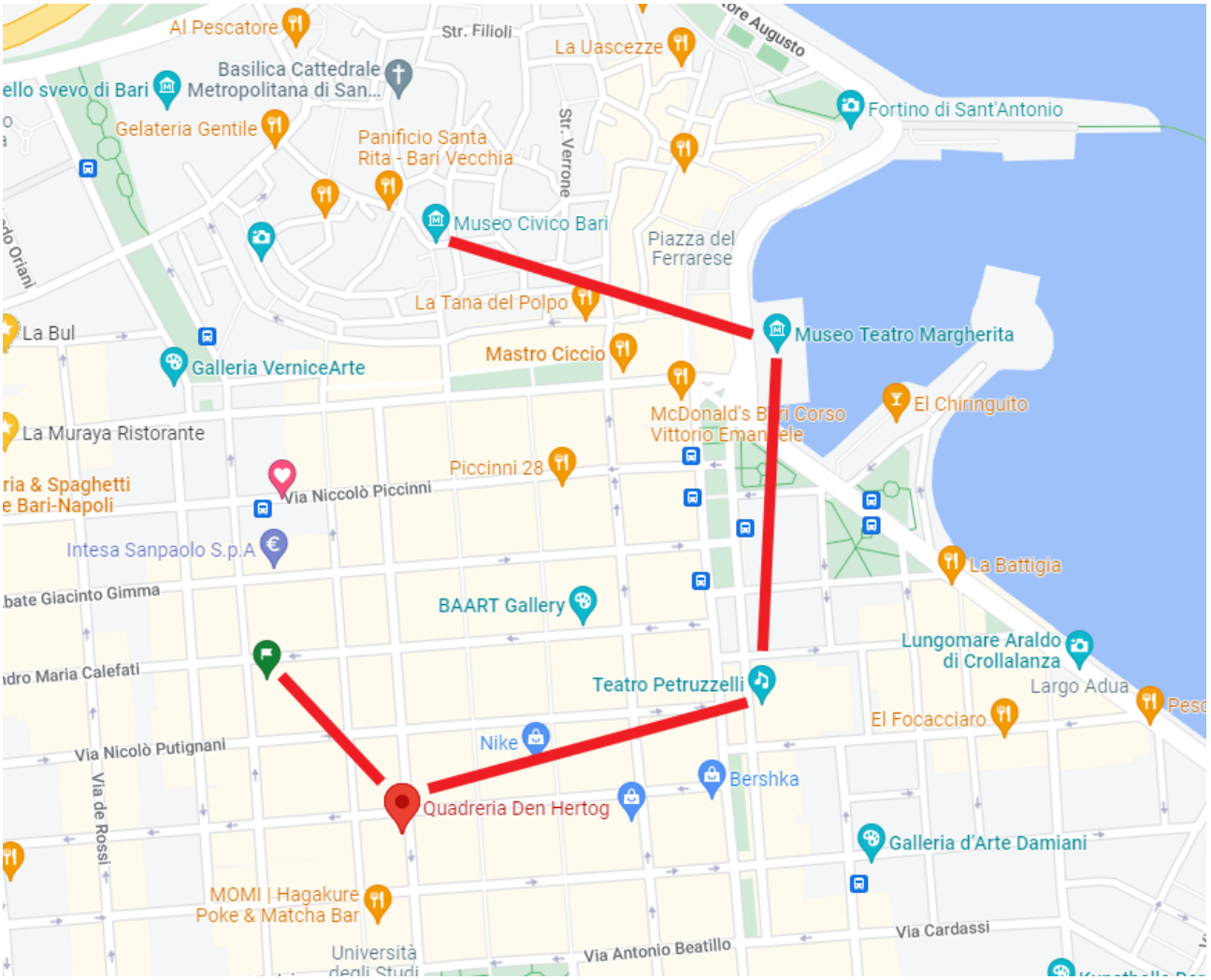
```
   ?- main.
   ____        _   _                 _   _____      _   _
  / __       | | (_)               | | |  __     | | | |
 | |  | |_ __ | |_ _ _ __ ___   __ _| | | |__) |_ _| |_| |__
 | |  | |  _ | __| |  _ ` _  / _` | | |  ___/ _` | __|  _
 | |__| | |_) | |_| | | | | | | (_| | | | |  | (_| | |_| | | |
  ____/| .__/ __|_|_| |_| |_|__,_|_|_| |_|   __,_|__|_|_| |_|
       | |
       |_|
Progetto FoAI 2022 - Universita degli studi di Bari Aldo Moro
Autori: Andrea Basile - Roberto Lorusso
Benvenuto!
Per cominciare inserisci la tua posizione...
Inserisci Latitudine:
   |: 41.12374194325008.
Inserisci Longitudine:
   |:  16.867298368953858.
Vuoi selezionare delle categorie di tuo interesse?:[[y/n]]
   |: n.
Vuoi selezionare un autore di tuo interesse?:[[y/n]]
   |: n.
Vuoi selezionare specifiche tipologie di luoghi?:[y/n]]
   |: n.
Vuoi selezionare delle citta di tuo interesse?:[[y/n]]
   |: y.
Seleziona una citta:

[bari]
   |: bari.
Hai selezionato: bari
Vuoi selezionare altro?: [y/n]
   |: n.
Inserisci un costo massimo per la tua visita:
   |: 80.

Hai selezionato: 80
Inserisci un raggio di azione per la ricerca delle attrazioni (in metri):
   |: 2000.

Hai selezionato: 2000
[quadreria,teatropetruzzelli,museoteatromargherita,museocivicobari,[]]
yes
```

Di seguito si riporta una figura che mostra il percorso ottenuto dall'algoritmo. Si noti che il percorso di visita, a partire dalla posizione dell'utente, è effettivamente ottimizzato rispetto alla distanza da percorrere.



6. Utilizzo

Per utilizzare il sistema esperto è necessario avviare **YAP 6.2.2** e consultare la knowledge-base.

Scrivendo “main” nella console il sistema richiede all’utente dei menù dai quali è possibile specificare le proprie preferenze per la sequenza di visita.

Una volta inserite tutte le informazioni, verrà visualizzato il path con l’ordine di visita nel file *explanation.txt* mentre i percorsi alternativi, nel file *alternative-path.txt.*

7. Sviluppi futuri

Sviluppo di un'interfaccia grafica che permetta all'utente di navigare la knowledge base utilizzando dei filtri sulle categorie, autori, ecc… e che permetta un utilizzo più user-friendly del programma, possibilmente visualizzando il percorso su una mappa e mostrando la descrizione dei punti di interesse/attrazioni.

Sviluppi futuri possono comprendere la possibilità di gestire il grado di certezza con cui un punto di interesse o una particolare attrazione possa piacere all'utente, basandosi sui gradi certezza di fatti già noti. Così facendo il sistema potrà stabilire e prevedere in maniera più accurata le preferenze dell'utente senza che queste gli vengano chieste esplicitamente, utilizzando i gradi di certezza come 'pesi' per includere punti di interesse e/o attrazioni all'interno della sequenza di visita.

Si prevede l'inserimento di nuove entità, come eventi e posti geografici più generali rispetto alle città. Attualmente il sistema offre la possibilità di scegliere una o più città all'interno delle quali costruire la sequenza di visita; sviluppi futuri possono far scegliere all'utente regioni geografiche più macroscopiche entro cui costruire il percorso ottimale.

Infine, ulteriori sviluppi prevedono la possibilità di scegliere il livello di dettaglio per la sequenza di visita e cioè se la sequenza debba comprendere attrazioni piuttosto che punti di interesse che le contengono. Si prevede anche la gestione delle date in cui è possibile ammirare un'attrazione in un punto di interesse o all'interno di nuove entità come gli eventi.