UNIVERSITÀ DEGLI STUDI DI BARI ALDO MORO

DIPARTIMENTO DI INFORMATICA

Documentazione Caso di Studio
Ingegneria della Conoscenza
A.A. 2023/24
Nome progetto: Drone Delivery

Gruppo di lavoro

- Vito Mascolo 668874, v.mascolo4@studenti.uniba.it
- Mauro Gaetano Allegretta 676581 m.allegretta6@studenti.uniba.it

Sommario

Introduzione

Il nostro obiettivo è un software per ottimizzare la consegna a domicilio utilizzando delle tecniche di intelligenza artificiale e analisi dei dati.

L'obiettivo principale del software è quello di decidere il percorso migliore per ridurre il più possibile i tempi di attesa della consegna e migliorare il servizio riducendo inquinamento (drone elettrico) ed evitare quindi possibili ritardi dovuti a traffico.

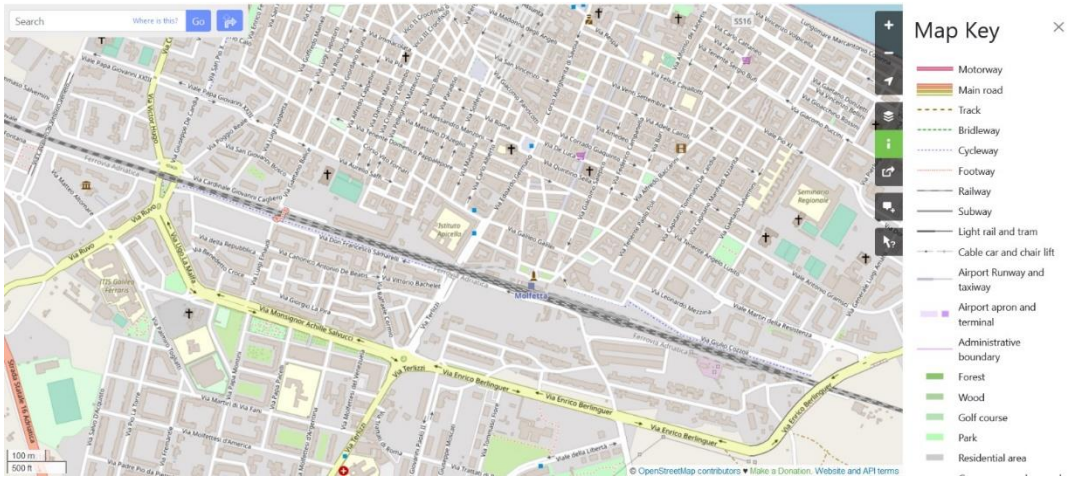
Ciò si ottiene utilizzando algoritmi di AI per l'analisi del meteo e adattare il percorso del drone in base alle consegne sfruttando la tecnica del path finding A*

Ontologie

OpenStreetMap (OSM) è un progetto open source che consente agli utenti di creare e condividere dati geografici, tra cui informazioni stradali (sia percorribili a piedi che non), edifici e punti di interesse. I dati di OSM sono disponibili in diversi formati, tra cui XML, che è un formato di testo utilizzato per la trasmissione e la rappresentazione dei dati su internet.

Le clausole in Prolog che abbiamo generato sono state ricavate dai dati ottenuti da OpenStreetMap in formato XML.

Prolog è un linguaggio di programmazione che può essere utilizzato per rappresentare conoscenza e interrogare una base di conoscenza. I dati di OpenStreetMap in formato XML sono stati convertiti in un formato adatto per la rappresentazione in Prolog, come una serie di clausole e classi che descrivono le relazioni tra gli elementi della mappa. La struttura di un file XML di OpenStreetMap è composta da un insieme di elementi che descrivono gli oggetti presenti nella mappa, come strade, edifici, punti di interesse e così via. Il file XML inizia con un elemento radice <osm> che contiene tutti gli altri elementi del file.



Gli elementi principali sono:

- <node>: rappresenta un punto geografico sulla mappa, con un set di attributi come le coordinate latitudine e longitudine.
- <way>: rappresenta una strada o una via, come una via principale o un sentiero, composto da una serie di nodi.
- <relation>: rappresenta una relazione tra gli elementi della mappa, come una relazione tra una strada e un edificio.
- <tag>: contiene i metadati per un elemento, come il nome della strada, il tipo di edificio o il tipo di sagnale stradale.

I dati di OSM sono organizzati come un grafo, dove ogni elemento è identificato da un identificativo univoco (id) e gli elementi sono collegati tra di loro mediante relazioni. Questo consente di navigare facilmente i dati e di estrarre informazioni rilevanti attraverso query sulla base di conoscenza rappresentata dai dati. La struttura a grafo rende possibile una facile navigazione e relazioni tra gli oggetti, permettendo di accedere facilmente alle informazioni richieste.

Knowledge Base (KB)

Un insieme di conoscenze organizzate tali che possano essere utilizzate da un sistema o da un programma per prendere decisioni o per rispondere a domande costituisce una Knowledge Base. La programmazione logica di prolog utilizza la KB che abbiamo implementato per descrivere un modello di sistema di navigazione del drone. Il modello utilizza le classi per rappresentare strade, edifici e movimenti del drone con proprietà specifiche per ogni classe. Prop viene usato per definire le relazioni tra le classi.

Fatti e regole della KB

Classe drone:

- Nome: nome del drone
- Latitudine: indica la latitudine del drone
- Longitudine: indica la longitudine del drone
- Altitudine: indica l'altitudine del drone
- Velocità: velocità in movimento del drone
- Batteria: autonomia del drone prima di una ricarica

La classe drone descrive le proprietà comuni dei droni come il nome, la velocità massima, la batteria ecc…

NON utilizzare il drone in condizioni meteorologiche avverse, come venti con velocità superiori a 12,8 m/s, neve, pioggia, nebbia, grandine o fulmini.

Le sue funzioni di movimento vengono definite nella knoledge base attraverso le funzioni salta, atterra e sposta.

```
/* Classe drone
*
* Contiene i seguenti attributi:
* - nome: nome del drone
* - latitudine: indica la latitudine del drone
* - longitudine: indica la longitudine del drone
* - altitudine: indica l'altitudine del drone
* - velocita': velocita' massima drone
* - batteria: tempo di durata del drone
*/

drone(posizione(X, Y), altitudine(Z)).


% Regole per il movimento verticale del drone

salta(Drone, NuovaAltitudine) :-

    drone(Posizione, AltitudineAttuale),

    NuovaAltitudine is AltitudineAttuale + 1,

    retract(drone(Posizione, AltitudineAttuale)),
```

```
    assert(drone(Posizione, NuovaAltitudine)).

atterra(Drone) :-
    drone(Posizione, _),
    retract(drone(Posizione, _)),
    assert(drone(Posizione, 0)).

% Regola per il movimento orizzontale del drone
sposta(Drone, NuovaPosizione) :-
    drone(PosizioneAttuale, Altitudine),
    NuovaPosizione = posizione(X, Y),
    retract(drone(PosizioneAttuale, Altitudine)),
    assert(drone(NuovaPosizione, Altitudine)).
```

Classe edificio:

- Latitudine: indica la latitudine del drone
- Longitudine: indica la longitudine del drone
- Altezza: indica l'altezza dell'edificio

La classe edificio descrive le proprietà comuni di un edificio, latitudine, longitudine e altezza. La funzione prop viene utilizzata per specificare che un edificio abbia i determinati valori per i suoi attributi.

Classe nodo: La classe nodo ha i seguenti attributi:

- Id: identificativo del nodo
- Latitudine: indica la latitudine del nodo
- Longitudine indica la longitudine del nodo

Query Knowledge Base (KB)

Le query su una KB in Prolog sono operazioni che servono per interrogare la KB e per ottenere e/o recuperare informazioni specifiche. In Prolog, si utilizza la sintassi della programmazione logica per esprimere le query ed esse sono usate per effettuare ricerche basate sui fatti e le regole presenti nella KB.

Calcola la distanza tra due nodi

```
/**
 * Calcola la distanza tra due nodi X e Y
 *
 * @param X: primo nodo
 * @param Y: secondo nodo
 * @param S: distanza tra i due nodi (viene restituito il risultato)
 */
distanza_nodi(X, Y, S) :- prop(X, latitudine, L1), prop(Y, latitudine, L2),
                          prop(X, longitudine, G1), prop(Y, longitudine, G2),
                          S is abs(L1 - L2 + G1 - G2).
```

La query distanza_nodi(X, Y, S) in Prolog calcola la distanza tra due nodi X e Y, e la assegna alla variabile S. Nello specifico, la query effettua le seguenti operazioni:

1. Il predicato **prop(X, latitudine, L1)** recupera la latitudine del nodo X e la assegna alla variabile L1.
2. Il predicato **prop(Y, latitudine, L2)** recupera la latitudine del nodo Y e la assegna alla variabile L2.
3. Il predicato **prop(X, longitudine, G1)** recupera la longitudine del nodo X e la assegna alla variabile G1.
4. Il predicato **prop(Y, longitudine, G2)** recupera la longitudine del nodo Y e la assegna alla variabile G2.
5. Il predicato **S is abs(L1 – L2 + G1 – G2)** calcola la distanza tra i due nodi come la somma della differenza tra le latitudini e la differenza tra le longitudini e la assegna al risultato S.

Query Per Recuperare Latitudine e Longitudine

La query **lat_lon(X, latitudine, longitudine)** in Prolog è utilizzata per recuperare la latitudine e la longitudine di un oggetto X specifico.

In particolare, la query effettua le seguenti operazioni:

1. Il predicato **prop(X, latitudine, latitudine)** recupera la latitudine dell'oggetto X specificato e la assegna alla variabile latitudine.
2. Il predicato **prop(X, longitudine, longitudine)** recupera la longitudine dell'oggetto X specificato e la assegna alla variabile longitudine.

```
/**
 * Restituisce la latitudine e longitudine di un nodo passato in input
 *
 * @param X: nodo di cui si vogliono conoscere le coordinate
 * @param L: latitudine
 * @param G: latitudine
 *
 */
lat_lon(X, Latitudine, Longitudine) :- prop(X, latitudine, Latitudine),
                                       prop(X, longitudine, Longitudine).
```

La query **controllo_altitudine(Altitudine, Altezza)** in prolog controlla che il drone sia più in alto dell'edificio, se non lo è dice che il drone deve saltare:

1. **regola_and(Altitudine, Altezza)** controlla che l'altitudine del drone non sia minore o uguale dell'altezza dell'edificio.
2. **salta(drone, Altezza)** nel caso che la regola_and è TRUE allora deve far alzare il drone.

```
/**
* Controlla che l'altezza dell'edificio non sia superiore all'altitudine del drone
* @param Altitudine: attuale altitudine del nodo
* @param Altezza: altezza dell'edificio
*/
controllo_altitudine(Altitudine, Altezza):- regola_and(Altitudine, Altezza),
salta(drone, Altezza).
```

La query **controllo_arrivo(X, Y)** in Prolog controlla che il drone sia arrivato alla destinazione. Nel caso sia arrivato, dice che il drone deve atterrare.

1. **lat_lon(X, latitudine, longitudine)** è utilizzata per recuperare la latitudine e la longitudine di un oggetto X specifico.
2. **atterra(Drone)** fa atterrare il drone.

```
/*
* Controlla se il drone e' arrivato alla posizione di arrivo, se si il drone attera
in quel punto
* @param X: posizione drone
* @param Y: posizione di arrivo
*/
```

```
controllo_arrivo(X, Y):- lat_lon(X, Latitudine, Longitudine),
                         lat_lon(Y, latitudine, longitudine),
                         Latitudine=latitudine, Longitudine=longitudine,
                         atterra(drone).
```

La query **vicini_edificio(Edificio, Vicini)** restituisce gli edifici immediatamente vicini dell'edificio passato in input. I due edifici sono vicini se hanno una strada in comune.

Nello specifico la query effettuerà le seguenti operazioni:

1. il predicato **prop(Edificio, type, edificio)** verifica che l'oggetto passato come primo elemento sia un edificio.
2. Il predicato **prop(Edificio, strade, Strade)** recupera le strade che attraversano l'"Edificio" e gli assegna alla variabile strada.
3. La query **vicini_strade_edificio(Edificio, Strade, Vicini)** utilizza la lista delle strade che si incrociano con gli edifici e restituisce una lista degli edifici vicini agli edifici presi in input e li assegna alla variabile "Vicini".

In questo modo la query principale **vicini_edificio(Edificio, Vicini)** utilizza la proprietà edificio per trovare gli edifici vicini.

La query **vicini_strade_edificio,** la prima serve come passo base perché la lista delle strade è vuota, la seconda serve per trovare gli edifici vicini ad un edificio specifico, usando la lista delle strade.

```
/**
 * Restituisce gli edifici immediatamente vicini dell'edificio passato in input.
 * Due edifici sono immediatamente vicini se collegati dalla strada in input.
 *
 * @param Edificio: Edificio di cui si vogliono conoscere i vicini
 * @param Vicini: lista di edifici vicini (viene restituito il risultato)
 */
vicini_edificio(Edificio, Vicini) :- prop(Edificio, type, edificio),
                                     prop(Edificio, strade, Strade),
                                     vicini_strade_edificio(Edificio, Strade,
Vicini).

vicini_strade_edificio(Edificio, [], Vicini) :- prop(Edificio, type, edificio),
Vicini = [].
vicini_strade_edificio(Edificio, [S1|S2], Vicini) :- prop(S1, nodi, N1),
                                                     vicini_strade_edificio(Edifici
o, S2, Vicini3),
                                                     append(Vicini3,
[Vicino1|Vicino2], Vicini).
```

La query **ritorno_altezza_edifici(Edificio, Altezza)** fa ritornare l'altezza degli edifici.

```
/**
* Restituisce l'altezza dell'edificio passato in input.
* Verifichiamo se quello passato e' un edificio.
*
* @param Edificio: identifica il nodo che vogliamo controllare e di cui vogliamo
sapere l'altezza nel caso.
* @param Altezza: identifica l'altezza dell'edificio che vogliamo restituire.
*/
ritorno_altezza_edifici(Edificio, Altezza):- lat_lon(Edificio, Latitudine,
Longitudine),
                                             prop(Edificio, type, edificio),
                                             prop(Edificio, altezza, Altezza).
```

Apprendimento Supervisionato

Il modello utilizzato ha il compito di predire come sarà il meteo in un determinato giorno e in determinati orari. Questo indice servirà per decidere quando far partire il drone per le consegne o quando farlo tornare. Per poter trovare questo indice assumiamo come condizioni di maltempo precipitazioni e temperatura (volevamo usare la velocità del vento ma a causa di mancanza di valori nel dataset abbiamo preso la temperatura per simulare che un'alta temperatura potesse rovinare il drone). Il dataset, preso dall'arpa della puglia, era molto limitante, in quanto molti dati non erano misurati e mancavano alcune condizioni di maltempo che ci interessavano, quindi in mancanza di alternative, abbiamo usato questo.

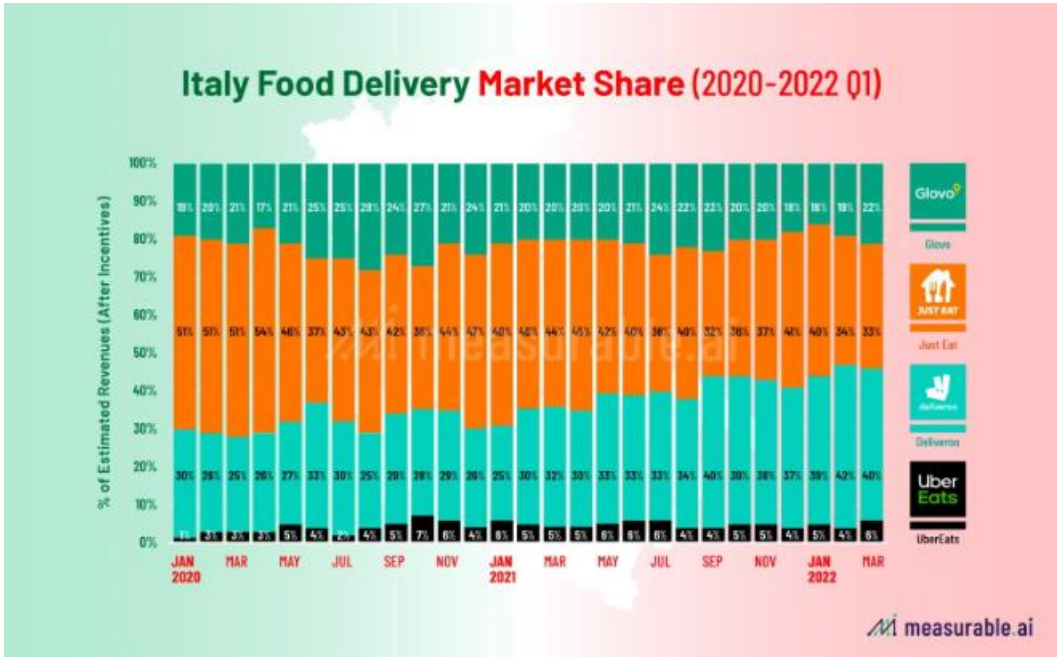
Data Cleaning

Il dataset iniziale "stazione_molfetta_anno_2023" contiene alcuni valori metereologici con intervalli di un'ora, a cui ogni data e ora ha associato i suoi valori metereologici odierni.

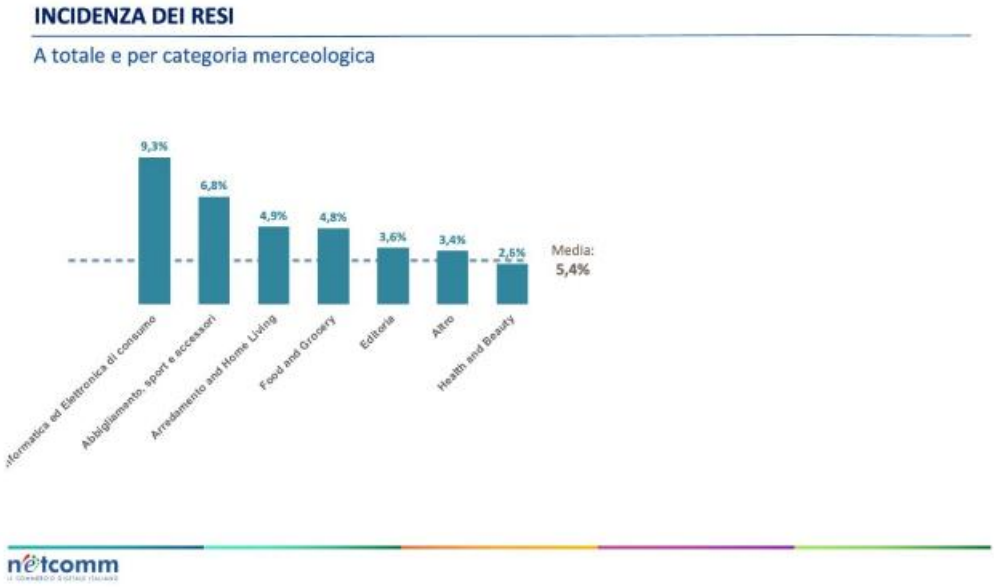
```
data;temperatura;flag_t;umr;flag_umr;precipitazione;flag_prec;vvento;flag_vv;dvento;flag_dv;radsolare;flag_rad;pressione;flag_pres;
2022-01-01 00:00:00;10.24;1;73.74;1;0.00;1;NaN;6;NaN;6;0.00;1;NaN;6;
2022-01-01 01:00:00;10.16;1;77.98;1;0.00;1;NaN;6;NaN;6;0.00;1;NaN;6;
2022-01-01 02:00:00;10.66;1;82.97;1;0.00;1;NaN;6;NaN;6;0.00;5;NaN;6;
2022-01-01 03:00:00;9.52;1;91.94;1;0.00;1;NaN;6;NaN;6;0.00;1;NaN;6;
2022-01-01 04:00:00;8.54;1;94.72;1;0.00;1;NaN;6;NaN;6;0.00;1;NaN;6;
```

Il dataset originario contiene dati a partire dall'01-01-2022 al 30-11-2023. Tuttavia le registrazioni relative a questi anni sono state prese in considerazione per capire l'attualità e l'usabilità del drone per fare delle consegne.

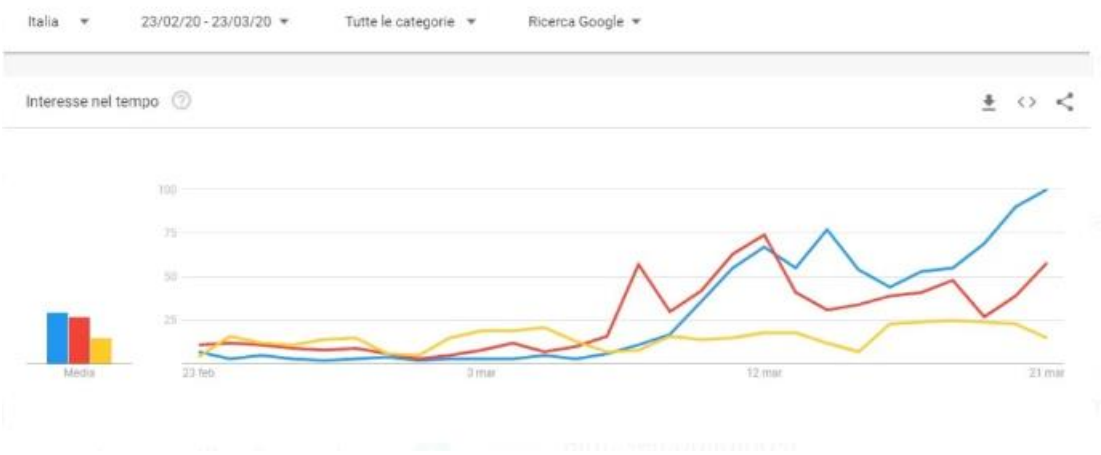
I seguenti grafici mostrano l'andamento delle consegne avvenute in questi ultimi anni in Italia (quelli specifici per molfetta non li abbiamo trovati)



Il primo grafico è su un ipotetico drone di consegne di cibo (quindi ci sarebbero da fare altri controlli).



Il secondo invece è più generale e serve per vedere come l'e-commerce sia più influente in alcuni ambiti più che in altri in modo da capire quali vincoli sarebbero più adatti da implementare in un drone per le consegne generali.



Il terzo grafico è stato preso durante gli anni della pandemia di Covid per dimostrare come determinate situazioni potessero causare un aumento o una diminuzione di consegne.

Scelta del modello

Sommario

Per poter predire il tempo in relazione alla data e orario rispetto alla città selezionata sono state analizzate le prestazioni del Random Forest in termini di accuratezza e tempi di risposta. Quindi possiamo definire il nostro task di regressione.

Il Random Forest è un metodo di apprendimento supervisionato che risolve un task di regressione, combinando diversi alberi decisionali per migliorare la precisione e la robustezza della previsione. In una random forest, viene addestrato un insieme di alberi decisionali indipendenti tra loro, ognuno su un sottoinsieme casuale dei dati di addestramento e utilizzando un sottoinsieme casuale delle feature. Durante la fase di previsione, ciascun albero fornisce una previsione e la classe maggioritaria tra tutte le previsioni degli alberi costituisce la previsione finale della random forest. Questo approccio aiuta a ridurre l'overfitting e migliorare le prestazioni del modello, rendendo le random forest particolarmente adatte a una vasta gamma di problemi di classificazione e regressione.

Strumenti utilizzati

```
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
```

Queste due righe servono per importare e utilizzare le librerie “sklearn.model_selection” e “sklearn.esemble” per implementare il nostro modello di Random Forest.

Decisioni di Progetto

Classification:

I parametri usati per definire il nostro random forest sono stati i valori di maltempo, ogni albero decisionale costruito con random forest avrebbe ottenuto un sottoinsieme casuale dei dati di addestramento e delle caratteristiche. I dati per addestrare il random forest sono stati i parametri del maltempo da predire e il valore binario di cui 1 indicherà la presenza di un determinato tipo di maltempo e 0 la sua assenza.

Constraint Satisfaction Problems (CSP)

Sommario

Abbiamo deciso di utilizzare il CSP per indicare dei vincoli ai problemi che ha il nostro drone, suddividendoli in Hard e Soft:

- Vincolo di consegna: verificare che sia stato consegnato il pacco
- Vincolo dell'orario: il pacco non dovrà essere consegnato in determinati orari, abbiamo presupposto dall'1 di notte alle 8 di mattina; abbiamo valutato quest'orario perché in caso di lavoro notturno sarà comunque possibile consegnare entro l'1.
- Vincolo di una sola consegna: inserito perché il drone dovrebbe fare tutte le consegne di un edificio una sola volta senza doverci ripassare.
- Vincolo con numero consegne: è usato per massimizzare il numero di consegne e massificarne l'efficacia.
- Vincolo di batteria: serve a controllare che il percorso trovato che dovrebbe percorrere il drone sia nei limiti prefissati della batteria.

Strumenti utilizzati

```
from constraint import Problem, AllDifferentConstraint
from KB.CSP.lib.CSP_problem import Variable
from KB.CSP.lib.CSP_SLS import SLSearcher
from datetime import datetime
```

Abbiamo deciso di utilizzare queste librerie per l'implementazione del nostro CSP. Alcune sono state ottenute dalla rete (datetime, per ottenere data e orario), altre invece sono state prese prendendo spunto da progetti precedenti.

Nel CSP abbiamo passato come parametri:

- max_battery: indica la durata della batteria del drone.
- max_deliveries_per_day: indica il numero massimo di consegne attuali del drone.

Decisioni di progetto

Per quanto riguarda le decisioni per il progetto abbiamo inserito nell'algoritmo di Knowledge_Base all'interno della funzione distanza_nodi_tempo il vincolo di controllo del CSP total_delivery_time_within_battery. Questo controllava che il percorso venisse svolto in tempo, adeguato alla batteria del drone in modo tale che avesse il tempo per ritornare indietro.

```
def distanza_nodi_tempo(self, X, Y, tempo_partenza=0, in_tempo=True):
        '''
        Metodo distanza_nodi
        --------------------
        Dati di input
        --------------
        X: primo nodo
        Y: secondo nodo
        tempo_partenza: tempo trascorso dall'inizio del percorso
        in_tempo: se True aggiunge il tempo al drone

        Dati di output
        --------------
```

```
        distanza: distanza tra i due nodi
        '''
        distanza = 0
        velocita = self.velocita
        radius = 6371

        query = "lat_lon("+X+", L, G)"
        for atom in self.prolog.query(query):
            lat1 = atom["L"]
            lon1 = atom["G"]

        query = "lat_lon("+Y+", L, G)"
        for atom in self.prolog.query(query):
            lat2 = atom["L"]
            lon2 = atom["G"]


        dlat = radians(lat2 - lat1)
        dlon = radians(lon2 - lon1)
        a = (sin(dlat / 2) * sin(dlat / 2) + cos(radians(lat1)) *
cos(radians(lat2)) * sin(dlon / 2) * sin(dlon / 2))
        c = 2 * atan2(sqrt(a), sqrt(1 - a))
        distanza = radius * c * 1000


        # converte km in secondi
        m_s = velocita / 3.6
        tempo = distanza / m_s

        distanza = self.total_delivery_time_within_battery(tempo, in_tempo)

        seconds_from_start += tempo

        if in_tempo:
            return distanza, tempo
        else:
            in_tempo = False
            return distanza, 0
```

Valutazione

Il nostro drone ha voluto valutare l'efficacia del nostro drone rispetto ai vincoli passati e abbiamo cercato di ottenere la soluzione migliore attraverso il nostro solve_CSP.

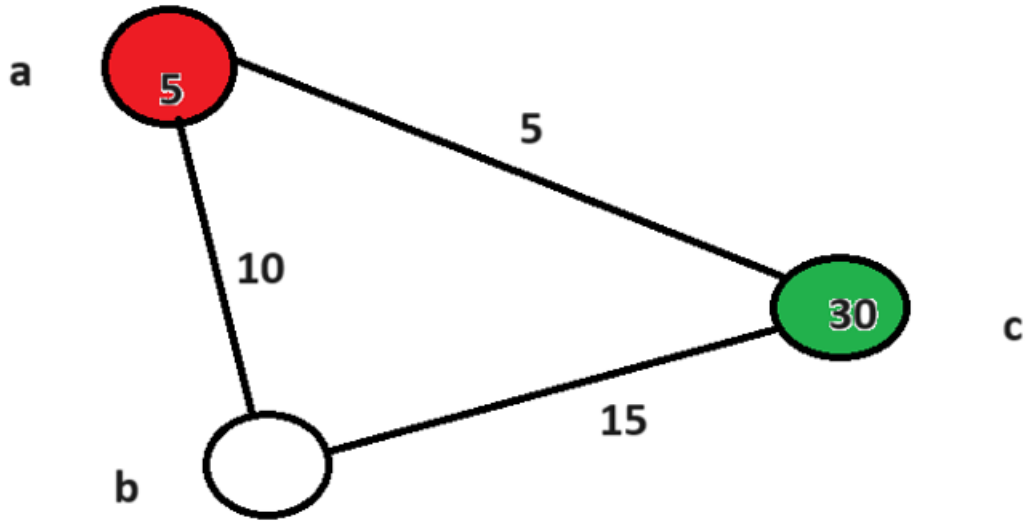
Ovviamente per avere un'efficacia maggiore il nostro drone dovrebbe rispettare tutti i vincoli prefissati; l'efficacia andrà diminuendo quando alcuni vincoli soft non saranno rispettati.

Il nostro drone restituirà distanza infinita nel caso in cui il tempo della batteria sia superiore al limite della batteria.

Algoritmo di path finding con A*

Sommario

Algoritmo di path finding A* viene utilizzato per calcolare il percorso ottimizzato per passare da un nodo di partenza ad un nodo di arrivo in un grafo per minimizzare il tempo totale per eseguire il percorso che verrà indicato dal costo dell'arco più il costo dei due nodi attraversati.



Il costo del percorso per andare da a fino a c può variare in base alle consegne da fare.

Strumenti utilizzati

L'algoritmo di path finding utilizzato è la sua formula base senza aggiunta di variabili, perché essendo un drone volante si evita attese ai semafori o intoppi di qualsiasi tipo che possono avvenire per strada.

f(p) = cost(p) + h(p)

cost(p) costo effettivo di p, con p → percorso del drone

h(p) stima del costo del cammino del drone da un nodo all'altro

Prolog offre diverse librerie e implementazioni per il calcolo del percorso (pathfinding) A*. Un'implementazione comune si trova nella libreria library(heaps) e può essere utilizzata per implementare l'algoritmo A*.

Decisioni di Progetto

L'algoritmo comunica direttamente solo con la base di conoscenza, quindi per determinare il costo del percorso del drone si suppone che esso segua esattamente la via più breve, e che quindi arrivi esattamente al nodo di arrivo.

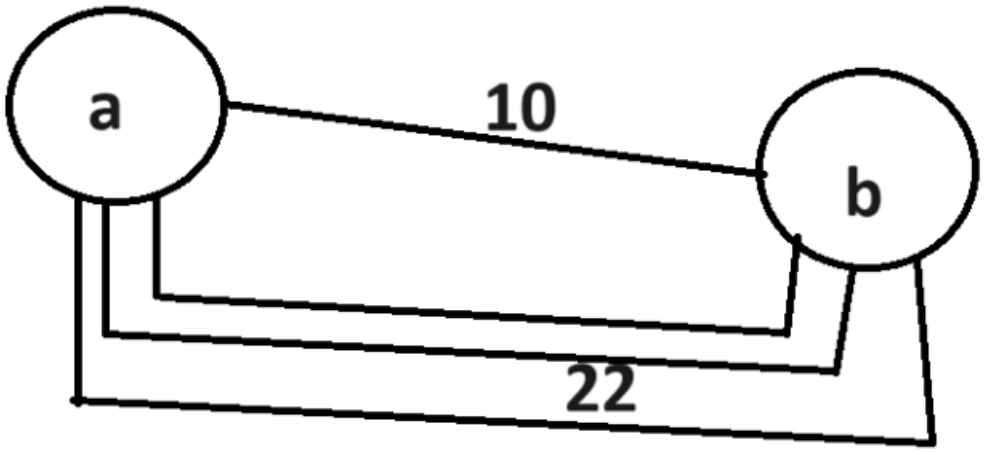
(Il grado di incertezza relativo alla consegna del drone è invece demandato al CSP.)

Prima di tutto si è definita la funzione euristica h(n): questa restituisce il tempo che si prevede di impiegare se il percorso più breve fosse in linea retta tra il nodo n attuale e il nodo d'arrivo. In questo modo h(p) è sicuramente ammissibile perché non sovrastima il costo reale, per i seguenti motivi:

- La distanza reale tra il nodo attuale e il nodo obiettivo non può essere più corta della distanza in linea retta tra i due nodi, ma al massimo uguale. Per calcolare la distanza prevista, si interroga la base di conoscenza per prelevare la latitudine e longitudine di entrambi i nodi per poi convertirli in metri tramite la formula della distanza euclidea.

Quindi, al massimo l'euristica utilizzata è uguale al costo reale ma mai minore, il che la rende ammissibile. Un'ulteriore caratteristica implementata nell'algoritmo è la rimozione dalla frontiera dei percorsi che presentano cicli. Di per sé, l'algoritmo A* è in grado di trovare la soluzione ottimale anche in presenza di cicli, tuttavia per una questione di efficienza in complessità computazionale si è deciso di effettuare questa scelta anche perché nella ricerca di un percorso (come in un navigatore) non si può far passare il drone due volte dallo stesso nodo, controllo rimandato al CSP.

La costruzione del grafo si è scelto di farla in modo dinamico, richiamando ogni volta dalla base di conoscenza la lista di nodi vicini al nodo attuale, ovvero l'ultimo presente nel percorso selezionato dalla frontiera (passando anche i secondi trascorsi dall'inizio del percorso). Si è deciso di far questo perché sarebbe stato impossibile avere i costi di ciascun nodo prima ancora di far partire la ricerca.



Da questo grafico si può vedere che il drone seguendo la strada avrebbe impiegato più tempo rispetto ad andare da A fino a B volando.

Valutazione

Abbiamo visto che il path finding A* e IDA* possono andare entrambi bene, abbiamo fatto un confronto su qual è meglio usare, avevamo questo dubbio perché IDA è più efficace nel caso di profondità di soluzione ottimale sconosciuta, avendo la mappa presa dalle ontologie, ponendo che la parte di mappa caricata comprenda la soluzione, abbiamo deciso di usare l'A* dato che presentava più vantaggi sulla consegna da parte del drone.

Usiamo algoritmo path finding A* (rientrando nei parametri batteria, questo controllo è stato fatto nel CSP) consegne effettuate entro limite di tempo e poi ricarica.

Conclusione

Il nostro software può essere ampliato in diversi modi in base all'esigenza di ciò che potrebbe consegnare, come visto sul grafico, per esempio controllo del peso nel caso di carichi pesanti, oppure per la fragilità del prodotto.

Tecnologie che si possono invece usare per ampliarlo sono:

- GPS, per sapere dov'è il drone e a che punto è la consegna;
- Una telecamera wireless, per il riconoscimento di un documento;
- Uno schermo touchscreen per la firma digitale;
- Un termometro sia per valutare la temperatura all'interno del drone per prevenire congelamento o surriscaldamento di alcune parti (fortemente consigliato in zone climatiche eccessivamente calde o eccessivamente fredde).