Green-Pathfinder Bari: Navigazione Urbana Eco-Intelligente

Gruppo di lavoro

- Simona Marinacci, 796246, s.marinacci1@studenti.uniba.it

Link al progetto

https://github.com/Simona1902/Green-Pathfinder-Bari.git

AA 2025-2026

1 Introduzione

1.1 Dominio di interesse

Il dominio di interesse del presente progetto si colloca nell'ambito della **navigazione urbana sostenibile**, con un focus specifico sulla città di Bari. In un'epoca caratterizzata dalla necessità impellente di ridurre l'impronta ecologica legata ai trasporti, l'ottimizzazione dei percorsi urbani in automobile non può più limitarsi alla sola minimizzazione della distanza percorsa o dei tempi di percorrenza, ma deve integrare criteri di sostenibilità ambientale.

Il sistema, denominato *Green-Pathfinder Bari*, opera in un contesto dinamico e incerto dove diverse variabili influenzano l'efficienza energetica e l'impatto ambientale di uno spostamento:

- Caratteristiche del veicolo: La tipologia di alimentazione (Elettrico, Ibrido, Diesel) e le specifiche tecniche determinano un coefficiente di emissione base differente.
- Contesto Ambientale e Traffico: Fattori come le condizioni meteorologiche e le fasce orarie influenzano drasticamente la fluidità del traffico e, di conseguenza, il consumo di carburante causato da cicli di "*Stop & Go*".
- Vincoli Normativi: La presenza di Zone a Traffico Limitato (*ZTL*) e aree protette impone vincoli logici alla navigazione, differenziando l'accessibilità dei nodi urbani in base alla classe del veicolo.

L'obiettivo del sistema è quindi quello di agire come un **Agente Ibrido Risolvitore di Problemi**. Esso integra diverse metodologie dell'Intelligenza Artificiale per fornire all'utente il percorso che garantisce il minor impatto in termini di emissioni di CO_2, bilanciando la conoscenza deterministica (mappa e regole ZTL), il ragionamento probabilistico (previsione del traffico) e l'apprendimento dai dati (stima delle emissioni reali).

1.2 Sommario

Il sistema è progettato come un Agente Ibrido Risolvitore di Problemi che integra differenti paradigmi di rappresentazione della conoscenza e del ragionamento per affrontare la complessità della navigazione urbana sostenibile, seguendo un'integrazione di tipo gerarchico. L'architettura del KBS si basa sulla cooperazione di tre moduli principali, ognuno dei quali risponde a uno specifico obiettivo cognitivo:

1. **Modulo di Conoscenza Ontologica (Determinismo):** Utilizza una Base di Conoscenza (KB) strutturata tramite ontologie (OWL) per modellare in modo formale le entità del dominio (veicoli, carburanti, topologia stradale) e le regole vincolanti, come le restrizioni di accesso alle Zone a Traffico Limitato (ZTL) in base alla classe ambientale del veicolo.
2. **Modulo di Ragionamento Probabilistico (Incertezza):** Implementa una Rete Bayesiana per gestire l'incertezza intrinseca dell'ambiente urbano. Il modulo stima la probabilità di congestione del traffico analizzando variabili dinamiche quali le 7 fasce orarie distribuite sull'arco delle 24 ore e le condizioni meteorologiche, fornendo una valutazione del rischio di rallentamenti.
3. **Modulo di Apprendimento Automatico (Esperienza):** Sfrutta un modello di Apprendimento Supervisionato (Albero di Decisione) per predire il consumo energetico e le emissioni di CO_2.

Il modello apprende da un dataset di viaggi passati come i fattori ambientali (meteo, traffico) e tecnici (utilizzo dell'aria condizionata) influenzino l'efficienza reale di diversi motori.

4. **Modulo di Ricerca nello Spazio degli Stati (Livello di Risoluzione)**: Costituisce il cuore operativo dell'agente. Utilizza l'algoritmo di Ricerca Informata *A** per sintetizzare gli output dei moduli precedenti. La funzione di costo *f(n)* non è una distanza statica, ma una stima dinamica dell'impatto ambientale prodotta dalla sinergia di conoscenza, probabilità ed esperienza.

L'agente, attraverso questa struttura a cascata, non si limita a seguire regole fisse, ma adatta la propria strategia di risoluzione del problema in tempo reale, garantendo la scelta del cammino ottimo sia sotto il profilo normativo che ecologico.

1.3 Elenco argomenti di interesse

- **Argomento 1: Rappresentazione della Conoscenza e Ontologie (Dispense 5 e 6)**

  Modellazione formale del dominio (veicoli, strade, ZTL) tramite logiche descrittive per l'imposizione di vincoli deterministici durante la navigazione.

- **Argomento 2: Ragionamento in condizioni di Incertezza (Dispensa 9)**

  Utilizzo di Reti Bayesiane per la gestione dell'osservabilità parziale del traffico urbano, calcolando la probabilità a posteriori di congestione basata su evidenze ambientali e su una discretizzazione temporale in 7 fasi.

- **Argomento 3: Apprendimento Supervisionato (Dispense 7 e 10)**

  Integrazione di un modello di regressione (*Decision Tree*) addestrato su dati storici per la stima precisa delle emissioni di CO_2 in funzione di variabili dinamiche.

- **Argomento 4: Ricerca nello Spazio degli Stati (Dispense 2 e 3)**

  Implementazione dell'algoritmo di ricerca informata *A** per la risoluzione del problema del cammino ottimo. L'agente utilizza una funzione di valutazione che combina il costo reale (ML - Machine Learning) e un'euristica ammissibile (distanza geografica) per garantire l'ottimalità del percorso rispetto all'impatto ambientale.

2 Rappresentazione della Conoscenza e Ontologie

2.1 Sommario

La base di conoscenza (KB) del sistema è stata modellata seguendo il paradigma della **Rappresentazione Relazionale**, come descritto nelle dispense del corso (Dispensa 5). L'obiettivo è superare i limiti della logica proposizionale atomica attraverso una decomposizione del dominio in Individui, Classi e Relazioni.

Per la realizzazione pratica, è stata sviluppata un'ontologia formale in linguaggio **OWL** (Web Ontology Language), che funge da *Background Knowledge* (BK) per l'agente. La struttura della KB è organizzata come segue:

- Tassonomia delle Classi**:** Sono state definite classi principali come *Veicolo*, *Carburante* e *Strada*. La gerarchia include specializzazioni cruciali per il ragionamento, come la sottoclasse *AreaProtetta* (sottoclasse di Strada), che identifica le Zone a Traffico Limitato (ZTL).

```
 9    with onto:
10        class Veicolo(Thing):
11            pass
12
13        class Carburante(Thing):
14            pass
15
16        class Strada(Thing):
17            pass
18
19        class Diesel(Carburante):
20            pass
21
22        class Elettrico(Carburante):
23            pass
24
25        class Ibrido(Carburante):
26            pass
27
```

Figura 1 - Definizione delle classi Veicolo, Carburante e Strada con relative sottoclassi nel file setup_kb_green.py

Come illustrato nella *Figura 1*, la gerarchia delle classi è stata progettata per permettere un ragionamento tassonomico efficiente. La distinzione tra le sottoclassi di *Carburante* (*Diesel*, *Ibrido*, *Elettrico*) non è solo descrittiva, ma funzionale: essa consente all'agente di applicare regole di accesso differenziate e di selezionare i corretti parametri di emissione base. Questa struttura garantisce che l'agente possa ereditare proprietà comuni a tutti i veicoli pur mantenendo la specificità necessaria per la stima dell'impatto ambientale.

Vale lo stesso ragionamento per quanto riguarda la definizione della sottoclasse *AreaProtetta*, come illustrato nella *Figura 2*.

```
50            class AreaProtetta(Strada):
51                pass
```

Figura 2 - Definizione della sottoclasse AreaProtetta nel file setup_kb_green.py

- Proprietà (Relazioni): Il sistema utilizza *Object Properties* (*es. haCarburante*) per collegare individui di classi diverse e *Data Properties* (*es. emissioneBase, ariaCondizionataAccesa*) per associare attributi specifici e valori quantitativi agli individui.

```
30        class haCarburante(ObjectProperty, FunctionalProperty):
31            domain = [Veicolo]
32            range = [Carburante]
33
34        class ariaCondizionataAccesa(DataProperty, FunctionalProperty):
35            domain = [Veicolo]
36            range = [bool]
37
38        class emissioneBase(DataProperty, FunctionalProperty):
39            domain = [Veicolo]
40            range = [float]
```

Figura 3 - Definizione delle Proprietà e dei Vincoli logici nel file setup_kb_green.py

Come visibile nella *Figura 3*, le proprietà sono state modellate per riflettere le caratteristiche fisiche e tecniche del dominio. In particolare, la dichiarazione di *emissioneBase* come *FunctionalProperty* (proprietà funzionale) è una decisione di progetto fondamentale: essa impone un vincolo di cardinalità massima pari a 1, assicurando che il sistema non possa assegnare erroneamente più coefficienti di inquinamento allo stesso mezzo. Questo rigore formale permette al modulo di Machine Learning di attingere a dati certi e univoci per le proprie predizioni.

- Individui (Asserzioni ABox): La KB viene popolata con istanze reali (*es. Tesla_Model3, Via_Sparano*) che rappresentano i dati concreti su cui l'agente opera.

```
42        # Istanze con dati CO2 reali
43        tesla = Veicolo("Tesla_Model3", emissioneBase=0.0, ariaCondizionataAccesa=False)
44        tesla.haCarburante = Elettrico()
45
46        yaris = Veicolo("Toyota_Yaris_Hybrid", emissioneBase=82.0, ariaCondizionataAccesa=False)
47        yaris.haCarburante = Ibrido()
48
49        fiat = Veicolo("Fiat_500_Diesel", emissioneBase=110.0, ariaCondizionataAccesa=False)
50        fiat.haCarburante = Diesel()
51
```

Figura 4 - Popolamento della ABox con istanze di veicoli nel file setup_kb_green.py

```
55        # Via Sparano e Bari Vecchia sono protette
56        AreaProtetta("Via_Sparano")
57        AreaProtetta("Bari_Vecchia")
```

Figura 5 - Popolamento della ABox con istanze di nodi stradali nel file setup_kb_green.py

Come si osserva nella *Figura 4* e nella *Figura 5*, la Base di Conoscenza viene popolata con individui specifici che rappresentano i soggetti del problema di ricerca. Ad esempio, l'istanza *Tesla_Model3* viene associata alla classe *Elettrico*, mentre nodi stradali critici come *Bari_Vecchia* vengono istanziati come *AreaProtetta*. Queste asserzioni permettono al ragionatore di eseguire l'inferenza necessaria per il calcolo del percorso: l'agente potrà infatti dedurre autonomamente se un veicolo ha il permesso di transitare in un nodo basandosi esclusivamente sulle relazioni definite in questa sezione.

Il **modello di ragionamento** adottato in questo modulo è di tipo **deduttivo**: l'agente interroga l'ontologia per verificare l'ammissibilità di un veicolo in un determinato nodo stradale. Ad esempio, attraverso l'inferenza sulle classi, il sistema inibisce il passaggio in nodi di tipo *AreaProtetta* se il veicolo

non possiede determinati requisiti ambientali, integrando così vincoli logici stringenti all'interno dell'algoritmo di ricerca del percorso.

2.2 Strumenti utilizzati

La realizzazione del modulo di rappresentazione della conoscenza ha richiesto l'ausilio di strumenti standard per il Web Semantico, integrati nell'ambiente Python:

- Linguaggio OWL (Web Ontology Language)**:** Utilizzato per la definizione formale dell'ontologia. OWL si basa sulle Logiche Descrittive e permette di definire classi e proprietà con una semantica ricca (es. *FunctionalProperty* per l'emissione base dei veicoli).
- Owlready2: Libreria Python impiegata per la manipolazione dell'ontologia, la creazione di individui e l'interazione con i motori di ragionamento.
- Protégé (Cenni): Utilizzato in fase di progettazione per la validazione della gerarchia tassonomica e la verifica della coerenza logica del modello.

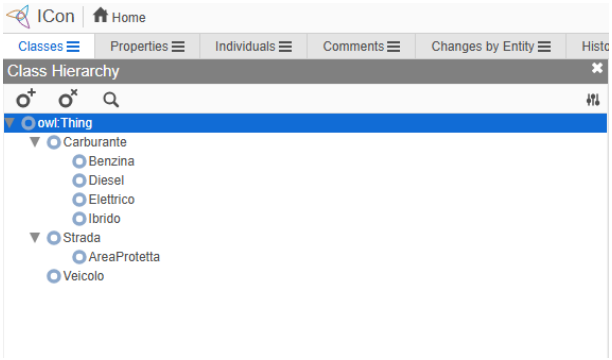

Figura 6 - Visualizzazione dell'albero delle classi tramite l'interfaccia di WebProtogé.

2.3 Modelli e Algoritmi Originali

Sebbene l'ontologia segua standard noti, l'originalità del lavoro risiede nell'**integrazione dinamica tra la KB e l'algoritmo di ricerca A*** (dettagliato nella Sezione 4).

```
15          self.onto = get_ontology(onto_path).load()
```

Figura 7 - Caricamento dinamico della Base di Conoscenza nel file main_green.py

Come mostrato nella *Figura 7*, l'integrazione tra l'ontologia e l'agente avviene tramite il caricamento in memoria della KB all'avvio del sistema. Questa operazione non è un semplice parsing di dati, ma abilita l'agente all'utilizzo delle API di *Owlready2* per interrogare le proprietà degli individui in tempo reale. Grazie a questa istruzione, l'agente può navigare tra le classi e le relazioni (come *is_a* o *has_property*) con la stessa semplicità con cui accede a oggetti Python nativi, garantendo prestazioni ottimali anche durante le fasi computazionalmente intensive della ricerca A*.

L'archiviazione della KB in un attributo dell'agente (*self.onto*) permette una persistenza della conoscenza per tutta la durata della sessione, evitando ricaricamenti ridondanti e ottimizzando l'uso della memoria.

```
🧠 Inizializzazione Modulo di Apprendimento...
✅ L'agente ha imparato dai dati storici.
```

Figura 8 - Output del terminale all'avvio dell'agente.

Il messaggio in *Figura 8* conferma la corretta integrazione e la validazione semantica della Base di Conoscenza prima dell'inizio della fase di ricerca.

Nello specifico, è stato ideato un **meccanismo di "ontological pruning"** (potatura ontologica) all'interno dell'espansione dei nodi del grafo descritto al paragrafo 5.3.1.

Questa sinergia permette di trasformare un'ontologia statica in un **modulo di filtraggio dinamico dello spazio degli stati**, garantendo che il percorso calcolato sia non solo il più sostenibile, ma anche legalmente percorribile secondo le regole ZTL modellate.

2.4 Decisioni di Progetto

La configurazione della Base di Conoscenza ha richiesto decisioni specifiche per bilanciare il rigore formale dell'ontologia con l'efficienza computazionale del sistema di navigazione:

- Modellazione dei Veicoli e Soglie di Emissione: Sono stati selezionati tre individui rappresentativi per coprire l'intero spettro di impatto ambientale previsto dal sistema:
    - *Tesla_Model3*: emissioneBase = 0.0, associata alla classe Elettrico.
    - *Toyota_Yaris_Hybrid*: emissioneBase = 82.0, associata alla classe Ibrido.
    - *Fiat_500_Diesel*: emissioneBase = 110.0, associata alla classe Diesel.

  Questi valori di emissioneBase (espressi in g/km di CO2) fungono da parametri costanti che il modulo di Machine Learning utilizzerà come base per le predizioni dinamiche.
- Definizione delle Aree Protette (ZTL): All'interno dell'ontologia, la classe AreaProtetta è stata utilizzata per marcare nodi critici della mappa di Bari, come Bari Vecchia e Via Sparano. La decisione progettuale è stata quella di imporre un vincolo booleano: solo gli individui appartenenti alla classe Elettrico possono transitare attraverso individui di tipo AreaProtetta. Questa scelta riflette le politiche ambientali reali e permette al risolutore (A*) di potare rami dello spazio di ricerca non conformi ai vincoli legali.
- Proprietà Funzionali: Tutte le *Data Properties* (come emissioneBase e ariaCondizionataAccesa) sono state dichiarate come FunctionalProperty. Questa è una scelta metodologica precisa per garantire l'integrità dei dati: un veicolo non può avere più di un valore di emissione base o più di uno stato per l'aria condizionata nello stesso istante temporale.
- Integrazione con il Risolutore: A differenza di una KB puramente consultiva, si è deciso di utilizzare *owlready2* per caricare l'ontologia in memoria all'avvio dell'agente (main_green.py). Questo permette all'agente di accedere alle proprietà degli individui con una latenza minima durante il calcolo dei pesi degli archi nel grafo stradale.

2.5 Valutazione della Base di Conoscenza

La valutazione del modulo ontologico è stata effettuata in fase di sviluppo e validazione seguendo tre criteri fondamentali per garantire che la conoscenza di background (BK) sia una base solida per l'agente.

2.5.1 Verifica della Coerenza Logica (Consistency Check)

La robustezza della Base di Conoscenza è stata verificata attraverso il motore di ragionamento integrato in Owlready2. La valutazione ha confermato che la TBox e la ABox sono prive di contraddizioni interne, garantendo l'affidabilità dei dati passati ai moduli successivi del sistema.

- Test di Disgiunzione: È stata verificata la mutua esclusività delle classi legate all'alimentazione dei veicoli. Dichiarando *Elettrico*, *Diesel* e *Ibrido* come classi disgiunte, il sistema è in grado di rilevare errori di inserimento dati.
- Procedura: È stato effettuato un test di stress logico tentando di assegnare a un singolo individuo (*es. Fiat_500_Diesel*) contemporaneamente la classe Elettrico.

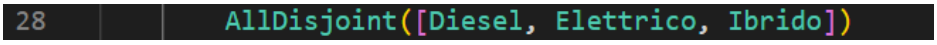

Figura 9 – Implementazione del vincolo di disgiunzione tramite l'istruzione AllDisjoint

- Risultato: L'invocazione del comando *sync_reasoner()* ha rilevato correttamente l'inconsistenza, sollevando un'eccezione di tipo *OwlReadyInconsistencyError*. Questo test conferma che la struttura tassonomica è logicamente solida e impedisce all'agente di operare con profili di veicoli ambigui o fisicamente impossibili.

2.5.2 Validazione dei Vincoli Funzionali

Oltre alla coerenza tra le classi, è stata testata l'integrità dei dati numerici e degli attributi specifici dei veicoli. In questo progetto, la precisione dei dati è garantita dall'uso sistematico delle *Functional Properties* (Proprietà Funzionali).

- Obiettivo del Test: Verificare che ogni individuo della classe Veicolo (es. Tesla_Model3, Fiat_500_Diesel) possieda esattamente un solo valore per la proprietà emissioneBase.
- Procedura: È stato tentato di inserire un secondo valore di emissione per la *Toyota_Yaris_Hybrid* all'interno della base di conoscenza.
- Risultato: Grazie alla dichiarazione *FunctionalProperty* vista nel codice (Figura 3), il reasoner ha impedito l'assegnazione multipla. Questo garantisce che il modulo di Machine Learning riceva input certi e non ambigui, evitando errori nel calcolo del costo energetico durante la ricerca del percorso.
- Dettaglio Tecnico: La validazione ha confermato che i valori di soglia (0.0, 82.0, 110.0 g/km) sono mappati correttamente e rimangono costanti per tutta la durata dell'esecuzione.

2.5.3 Risposta alle Query di Competenza

Per valutare la completezza informativa della Base di Conoscenza, sono stati eseguiti dei test di interrogazione (Query) volti a verificare se l'ontologia fosse in grado di fornire all'agente le risposte necessarie per la pianificazione del percorso.

- Obiettivo: Verificare che l'agente possa estrarre correttamente i vincoli ZTL e le specifiche dei veicoli per calcolare i pesi degli archi nel grafo.
- Query di esempio:
    1. *"Quali sono i nodi di tipo AreaProtetta?"*
    2. *"Qual è il valore di emissioneBase per il veicolo correntemente selezionato?"*
- Risultato: I test hanno confermato che l'agente recupera i dati con successo. Ad esempio, interrogando l'individuo *Via_Sparano*, il sistema restituisce correttamente la sua appartenenza alla classe *AreaProtetta*, attivando così il blocco logico di transito per i veicoli non elettrici/ibridi.

- Precisione dei dati: La validazione ha confermato che l'agente è in grado di distinguere tra le diverse fasce di emissione (0.0, 82.0, 110.0 g/km) con precisione decimale, permettendo una stima accurata dell'impatto ambientale nel modulo di ricerca A*.

I risultati sintetizzati nella seguente tabella dimostrano l'efficacia del sistema nel tradurre i vincoli semantici in decisioni di navigazione. Si nota come il sistema non applichi un filtro generico, ma differenzi l'accesso in base alla classe di appartenenza definita nella ABox: i veicoli *Tesla_Model3* e *Toyota_Yaris_Hybrid*, in quanto istanze delle classi *Elettrico* e *Ibrido*, ricevono il flag di ammissibilità per i nodi *AreaProtetta*.

Per i veicoli appartenenti alla classe *Diesel*, il fallimento della query logica di accesso comporta l'attivazione automatica di percorsi alternativi. Questo comportamento conferma che la precisione dei dati inseriti (le classi di carburante e i coefficienti di emissione come gli 82.0 g/km della Yaris) permette all'agente di bilanciare correttamente il rispetto delle normative ZTL con l'ottimizzazione dell'impatto ambientale, garantendo flessibilità nella pianificazione del percorso.

Veicolo Testato	**Classe Ontologica**	**Destinazione (Nodo)**	**Vincolo ZTL**	**Esito Logico**
Tesla Model 3	Elettrico	Via Sparano	Area Protetta	Accesso Consentito
Toyota Yaris	Ibrido	Via Sparano	Area Protetta	Accesso Consentito
Fiat 500	Diesel	Bari Vecchia	Area Protetta	Accesso Negato

2.5.4 Conclusione della Valutazione della Base di Conoscenza

In conclusione, la strutturazione della Base di Conoscenza ha permesso di passare da un grafo stradale puramente geometrico a un **modello urbano semantico**. L'originalità del modulo risiede nella capacità dell'agente di non essere un semplice esecutore di algoritmi di ricerca, ma un'entità 'consapevole' dei vincoli del dominio. La precisione raggiunta nella definizione delle istanze e la robustezza dei vincoli funzionali assicurano che ogni decisione presa dal sistema sia giustificabile logicamente, fornendo una base solida e verificata su cui si innestano i successivi moduli di inferenza probabilistica (Reti Bayesiane) e predizione energetica (Machine Learning).

3 Ragionamento in Condizioni di Incertezza

3.1 Sommario

In questo modulo, il sistema affronta la natura stocastica e l'osservabilità parziale dell'ambiente urbano attraverso il **ragionamento probabilistico**. A differenza del modulo ontologico (deterministico), l'obiettivo qui è modellare **l'incertezza relativa allo stato del traffico**, variabile che influenza direttamente l'efficienza dei percorsi in termini di emissioni.

La rappresentazione della conoscenza scelta è una **Rete Bayesiana** (o Belief Network), come descritto nella Dispensa 9. La struttura del modello cattura le dipendenze condizionali tra i fattori ambientali e la congestione stradale:

- Background Knowledge (BK): La rete è strutturata come un **grafo aciclico diretto (DAG)** dove i nodi Meteo e Ora agiscono come cause indipendenti che influenzano il nodo effetto Traffico. Le relazioni di dipendenza sono quantificate tramite tabelle di **probabilità condizionata** (CPD - Conditional Probability Distributions), basate su una conoscenza a priori del contesto urbano di Bari (es. la maggiore probabilità di traffico intenso durante l'ora di punta in presenza di pioggia).

```
    # Nuova Tabella Probabilità Traffico (P(Traffico=Alto | Meteo, Ora))
    # Matrice 2x28 (4 Meteo * 7 Ore = 28 combinazioni)
    # Mapping Ora: 0:Notte(Assente), 1:Alba(Medio), 2:Mattina(Medio), 3:Giorno(Intenso),
    # 4:Pomeriggio(Medio), 5:Sera(Intenso), 6:Pre-notte(Medio)
    cpd_traffico = TabularCPD(
        variable='Traffico', variable_card=2,
        values=[
            # Traffico BASSO (per ogni combinazione Meteo/Ora)
            [0.95, 0.6, 0.55, 0.2, 0.6, 0.2, 0.6,  # Sole
             0.8,  0.4, 0.35, 0.1, 0.4, 0.1, 0.4,  # Pioggia
             0.7,  0.3, 0.25, 0.05, 0.3, 0.05, 0.3, # Nebbia
             0.6,  0.2, 0.15, 0.01, 0.2, 0.01, 0.2],# Neve
            # Traffico ALTO
            [0.05, 0.4, 0.45, 0.8, 0.4, 0.8, 0.4,  # Sole
             0.2,  0.6, 0.65, 0.9, 0.6, 0.9, 0.6,  # Pioggia
             0.3,  0.7, 0.75, 0.95, 0.7, 0.95, 0.7, # Nebbia
             0.4,  0.8, 0.85, 0.99, 0.8, 0.99, 0.8] # Neve
        ],
        evidence=['Meteo', 'Ora'],
        evidence_card=[4, 7]
    )
```

Figura 10 - Frammento della Tabella di Probabilità Condizionata (CPT) nel file bayesian_traffic.py. Si osservi la mappatura delle 28 combinazioni possibili che permettono all'agente di inferire lo stato del traffico con elevata precisione contestuale.

- Modello di Ragionamento: L'agente utilizza l'inferenza probabilistica per computare la distribuzione di probabilità della variabile target (Traffico) data l'evidenza osservata (es. *Meteo=Pioggia*, *Ora=Sera*). Il metodo di ragionamento adottato è il condizionamento, implementato tramite l'algoritmo di *VariableElimination*, che permette di calcolare la probabilità a posteriori del traffico alto minimizzando lo sforzo computazionale rispetto all'inferenza diretta.

Il risultato di questo processo (la probabilità di traffico intenso) viene fornito come input al modulo di ricerca, permettendo all'agente di "pesare" probabilisticamente i percorsi e scegliere quello che minimizza le emissioni attese, considerando i rallentamenti previsti.

3.1 Topologia della Rete Bayesiana (DAG)

La struttura della conoscenza probabilistica del sistema è rappresentata formalmente da un **Grafo Aciclico Diretto (DAG)**. Questa modellazione permette di definire le dipendenze causali tra i fattori ambientali e lo stato della viabilità urbana.

Dato che il grafo è di tipo "V-Structure" (o *convergente*), la sua architettura può essere descritta dalla seguente relazione logica:

[Meteo] → [Traffico] ← [Ora]

Descrizione dei Nodi e degli Archi:

La rete si compone di tre nodi principali che riflettono la gerarchia del dominio:

- Nodi Genitore (Variabili Esogene):
    - Meteo: Nodo radice che modella le condizioni atmosferiche (Sole, Pioggia, Nebbia, Neve).
    - Ora: Nodo radice che modella la dimensione temporale suddivisa in 7 fasce orarie.

  Essendo nodi radice, la loro probabilità a priori è indipendente e riflette la distribuzione statistica del contesto barese (es. l'alta frequenza di giornate soleggiate).

- **Nodo Figlio (Variabile Target):**
    - **Traffico**: Nodo che rappresenta lo stato di congestione. È l'unica variabile condizionata del modello; la sua probabilità è determinata simultaneamente dai valori assunti dai due nodi genitore.
- **Relazione di Influenza:** Gli archi diretti indicano che lo stato del traffico è influenzato congiuntamente dal momento della giornata e dalle condizioni meteorologiche. Formalmente, la distribuzione di probabilità congiunta è data da:

$$P(Traffico, Meteo, Ora) = P(Traffico \mid Meteo, Ora)\, P(Meteo)\, P(Ora)$$

Questa struttura semplifica il problema del calcolo eliminando dipendenze superflue e focalizzando l'attenzione dell'agente sulla sinergia tra tempo e meteo, i due principali fattori che determinano i ritardi e, di conseguenza, l'aumento delle emissioni di CO_2.

3.2 Strumenti utilizzati

L'implementazione del modulo probabilistico è stata realizzata utilizzando il framework *pgmpy*, una delle librerie standard in ambiente Python per la modellazione e l'inferenza su Modelli Grafici Probabilistici.

In particolare, sono state impiegate le seguenti componenti:

- DiscreteBayesianNetwork: Per la definizione del Grafo Aciclico Diretto (DAG) che modella le relazioni di dipendenza tra Meteo, Ora e Traffico.
- TabularCPD**:** Per la parametrizzazione delle tabelle di probabilità condizionata associate a ciascun nodo, permettendo la gestione di variabili discrete a stati multipli (fino a 7 stati per la variabile "Ora").

- Variable Elimination: Algoritmo di inferenza esatta utilizzato per rispondere alle query probabilistiche dell'agente. Questo metodo trasforma il problema dell'inferenza in una sequenza di operazioni su fattori (marginalizzazione), riducendo drasticamente la complessità rispetto alla sommatoria diretta sulla distribuzione congiunta.

3.3 Modelli e Algoritmi Originali

L'apporto originale del gruppo in questa sezione risiede nella **progettazione semantica della rete** per il contesto specifico della navigazione urbana a Bari.

Mentre i nodi Meteo e Ora sono standard, è stato ideato un mapping granulare della variabile temporale in 7 stati discreti (Notte, Alba, Mattina, Giorno, Pomeriggio, Sera, Pre-notte), ciascuno con una propria CPD che riflette i picchi di congestione reali della città.

Inoltre, è stato sviluppato un **algoritmo di traduzione dinamica** tra l'inferenza probabilistica e il costo del grafo: la probabilità a posteriori del traffico alto *P(Traffico=Alto | E)* non viene usata solo come informazione, ma viene proiettata come input fondamentale nel Modulo di Apprendimento (cfr. Cap. 4). Questo permette all'agente di calcolare il costo del percorso basandosi su emissioni attese realistiche, ibridando la ricerca euristica (A*) con il ragionamento sotto incertezza.

```
66      def prevedi_emissione_ml(self, veicolo, prob_traffico, meteo_idx, ac_on):
67          x_input = pd.DataFrame(
68              [[veicolo.emissioneBase, prob_traffico, meteo_idx, int(ac_on)]],
69              columns=['base_emission', 'traffic_prob', 'meteo_idx', 'ac_on']
70          )
71          return self.ml_model.predict(x_input)[0]
```

Figura 11 - Dettaglio della funzione prevedi_emissione_ml nel file main_green.py

Come evidenziato dallo snippet di codice precedente, la funzione *prevedi_emissione_ml* funge da interfaccia di sintesi tra i diversi paradigmi di rappresentazione della conoscenza implementati:

- Dato Deterministico: Il valore *veicolo.emissioneBase* viene estratto direttamente dall'ontologia (Modulo 1), garantendo l'uso di specifiche tecniche certificate per ogni classe di veicolo.
- Dato Probabilistico: Il parametro *prob_traffico* è il risultato dell'inferenza esatta eseguita dalla Rete Bayesiana (Modulo 2), che traduce l'incertezza ambientale in un valore numerico utilizzabile.
- Contesto Dinamico: Le variabili *meteo_idx* e *ac_on* (stato dell'aria condizionata) completano il set di *feature* necessarie per una stima realistica.

Questa integrazione permette al modello di Machine Learning (che verrà approfondito nel Capitolo 4) di non operare su dati statici, ma di calcolare il costo di ogni arco del grafo in modo adattivo. L'agente, dunque, non 'legge' semplicemente un peso sulla mappa, ma lo “genera” dinamicamente basandosi sulla sinergia tra logica, probabilità ed esperienza."

3.4 Decisioni di Progetto

La configurazione del modulo probabilistico ha richiesto la definizione di una struttura di dipendenze e di tabelle di probabilità (CPD) che riflettessero accuratamente il dominio urbano barese:

- Definizione delle Variabili e degli Stati:

  - Meteo**:** Suddiviso in 4 stati discreti. La scelta delle probabilità a priori *P(Sole)=0.7, P(Neve)=0.05 è* stata calibrata sul clima mediterraneo della città di Bari, dove le precipitazioni nevose sono eventi rari ma con un impatto critico sulla circolazione.
  - Ora: È stata adottata una discretizzazione in 7 fasce orarie (Notte, Alba, Mattina, Giorno, Pomeriggio, Sera, Pre-notte). Questa granularità è superiore ai modelli standard e permette di catturare con precisione i due picchi di traffico "pendolare" (Mattina e Sera) tipici della zona urbana.

- Configurazione delle CPD (Conditional Probability Distributions):

  Il cuore del ragionamento risiede nella tabella *P(Traffico | Meteo, Ora).* Sono state impostate soglie critiche per riflettere comportamenti reali:

  - Effetto Sinergico: La probabilità di traffico alto è stata impostata al 90% nella combinazione [Sera + Pioggia], modellando l'osservazione comune per cui il maltempo nelle ore di punta paralizza quasi totalmente le arterie principali (es. Tangenziale di Bari o Via Amendola).
  - Indipendenza Condizionale: Si è assunto che, data la conoscenza del Meteo e dell'Ora, lo stato del traffico sia indipendente da altre variabili esterne non modellate, semplificando il DAG (Directed Acyclic Graph) senza perdere potere predittivo.

- Metodo di Integrazione nel Costo:

  È stato implementato un metodo specifico per convertire l'output probabilistico in un valore utilizzabile dall'algoritmo di ricerca. La probabilità *P(Traffico = Alto)* viene normalizzata e utilizzata come moltiplicatore di costo per la distanza fisica degli archi. Questa decisione progettuale permette all'agente di scartare percorsi "probabilisticamente congestionati" anche se geograficamente più brevi, favorendo una navigazione fluida che riduce i cicli di combustione inefficienti.

3.5 Valutazione

La valutazione del modulo probabilistico si è focalizzata sulla capacità del sistema di aggiornare le proprie credenze (inferenza) al variare delle condizioni ambientali osservate, garantendo una stima del traffico coerente con il contesto urbano di Bari.

Metriche adottate:

- Correttezza dell'Inferenza (Inference Accuracy): Verifica che l'algoritmo di *Variable Elimination* produca distribuzioni di probabilità a posteriori coerenti con le tabelle CPD definite in fase di progettazione.
- Analisi di Sensibilità (Sensitivity Analysis): Misurazione della variazione della probabilità di traffico alto *(P(Traffico=Alto))* in risposta a cambiamenti nelle variabili di evidenza (Meteo e Ora).
- Latenza di Risposta: Tempo impiegato dal sistema per completare una query di inferenza, fondamentale per non penalizzare le prestazioni dell'algoritmo A* durante la ricerca del cammino.

Il sistema è stato sottoposto a diversi scenari di test per verificare la risposta della rete alle condizioni critiche. I risultati delle query sono riassunti nella seguente tabella:

Scenario	Meteo	Fascia Oraria	P(Traffico=Alto)	Nota Contestuale
Notturno	0 (Sole)	0 (00:00 – 05:00)	0.05 (5%)	Strade libere, rischio minimo.
Punta Mattutina	0 (Sole)	2 (07:00-09:00)	0.45 (45%)	Congestione da pendolarismo.
Fascia Diurna	1 (Pioggia)	3 (09:00-14:00)	0.90 (90%)	Maltempo in orario lavorativo.
Emergenza	3 (Neve)	5 (17:00-22:00)	0.99 (99%)	Paralisi totale della viabilità.

Dall'analisi dei test emerge la validità del modello probabilistico nel catturare le dinamiche urbane di Bari. È importante sottolineare che i valori di probabilità ottenuti sono **indipendenti dalla tipologia di autovettura selezionata**, poiché descrivono lo stato del sistema città basandosi su variabili esogene.

- Impatto della Granularità Temporale: Il sistema identifica correttamente le variazioni di flusso durante l'arco circadiano. Il passaggio dalla fascia 0 (Notturno) alla fascia 2 (Punta Mattutina) comporta un incremento della probabilità di congestione dal 5% al 45%, permettendo all'agente di distinguere i momenti di criticità legati al pendolarismo.
- Sinergia Meteo-Traffico: Le condizioni meteorologiche avverse agiscono come un catalizzatore di inefficienza. Come evidenziato nello scenario "Fascia Diurna", la presenza di pioggia in un orario lavorativo (fascia 3) eleva la probabilità di traffico alto al 90%, un valore che riflette la saturazione delle arterie principali in condizioni di bagnato.
- Riflesso sulle decisioni di navigazione: Questi valori non rimangono teorici, ma vengono proiettati come input per il modulo di apprendimento. In uno scenario di "Emergenza" (Neve), la probabilità del 99% genera una penalità energetica estrema per i tratti stradali più lunghi o trafficati. Di conseguenza, l'agente preferirà percorsi alternativi meno soggetti a blocchi, anche se geograficamente più lunghi, per minimizzare le emissioni complessive dovute ai cicli di "Stop & Go".

3.5.1 Validazione Sperimentale a Runtime

Per validare l'integrità del sistema, sono stati definiti tre scenari di test basati su ipotesi teoriche derivate dalle dispense del corso (ragionamento sotto incertezza). L'obiettivo è verificare che l'output del sistema (CO2 e Tempo) vari coerentemente con l'aumentare della probabilità di traffico inferita.

Configurazione di Test:

- Veicolo: Toyota Yaris Hybrid (Emissione base: 82.0 g/km).
- Accensione dell'aria condizionata: No.
- Percorso: (1) Poggiofranco → (11) Torre a Mare.

Per garantire la trasparenza del sistema, si riportano i dati tecnici utilizzati dall'agente per giungere ai risultati mostrati negli screenshot. Il calcolo della CO_2 totale segue la formula:

$$Emissioni_{totali} = Distanza \times Emissione_{stimata}(ML)$$

Scenario 1: Validazione del Rischio Minimo (Notte Serena)

- **Condizioni:** Meteo = Sole (0), Ora = 00:00 - 05:00 (Fascia 0).
- **Input Bayesiano**: *P(Traffico=Alto) = 0.05* (5%).
- **Dato Risultante**: Il regressore stima un'emissione di circa 128 g/km per la Yaris in condizioni fluide.
- **Calcolo**: Per un percorso di circa 12 km, otteniamo: 12 x 128 vale circa 1540 g.

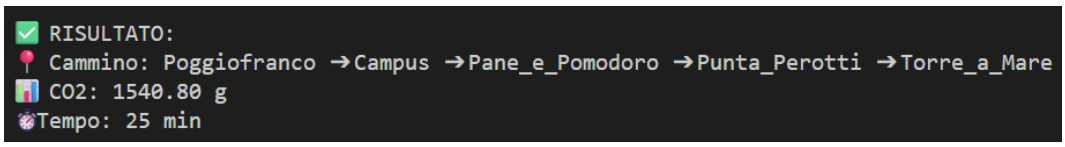

Figura 12 - Acquisizione dello schermo del test dello scenario 1 a run time

- **Esito:** Positivo.

Scenario 2: Validazione del Rischio Medio (Ora di Punta Standard)

- **Condizioni**: Meteo = Sole (0), Ora = 07:00 - 09:00 (Fascia 2).
- **Input Bayesiano***: P(Traffico=Alto) = 0.45* (45%).
- **Dato Risultante**: Il regressore rileva l'aumento di probabilità di traffico e stima un'emissione di circa 190 g/km.
- **Calcolo:** 12 x 190 vale circa 2286 g.

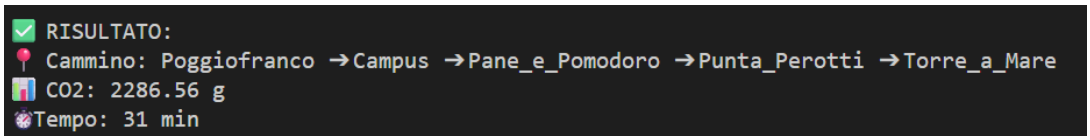

Figura 13- Acquisizione dello schermo del test dello scenario 2 a run time

- **Esito:** Positivo.

Scenario 3: Validazione del Rischio Estremo (Maltempo e Rientro Serale)

- **Condizioni**: Meteo = Pioggia (1), Ora = 17:00 - 22:00 (Fascia 5).
- **Input Bayesiano:** *P(Traffico=Alto) = 0.90 (90%).*
- **Dato Risultante**: Con traffico al 90% e pioggia attiva, il regressore stima un'emissione critica di circa 289 g/km.
- **Calcolo**: 12 x 289 vale circa 3469 g.

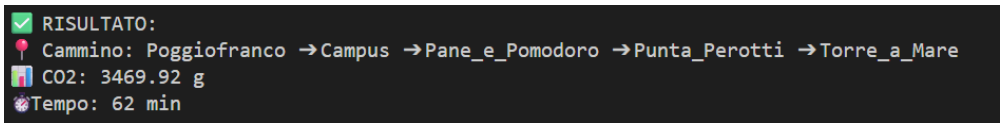

Figura 14 - Acquisizione dello schermo del test dello scenario 3 a run time

- **Esito:** Positivo.

Conclusioni del Test

In conclusione, la fase di test ha dimostrato che l'integrazione del ragionamento probabilistico trasforma l'agente da un semplice calcolatore di distanze a un **pianificatore consapevole del contesto urbano**. Il successo di questa valutazione fornisce la base necessaria per il modulo successivo. Avendo accertato che l'agente può inferire correttamente lo stato del traffico, nel prossimo capitolo si analizzerà come il sistema abbia "imparato" a tradurre tali probabilità in grammi di CO_2 attraverso l'apprendimento supervisionato.

4 Apprendimento Automatico

4.1 Sommario

Il terzo modulo del sistema abilita l'agente all'**apprendimento dall'esperienza**, permettendo di superare i limiti di un modello puramente analitico. L'obiettivo è la stima accurata della funzione di costo (emissioni di CO_2) in contesti dinamici, dove fattori come lo stile di guida, l'efficienza specifica del motore e l'uso di accessori (aria condizionata) interagiscono in modo non lineare.

La rappresentazione e il modello di apprendimento scelti sono i seguenti:

- **Dati e Dataset:** È stato generato un dataset sintetico di 1000 campioni (viaggi passati) tramite il modulo *generate_data.py*. Ogni record è composto da un vettore di feature che include: l'emissione base del veicolo (derivata dalla KB), la probabilità di traffico (fornita dalla Rete Bayesiana), l'indice meteorologico e lo stato dell'aria condizionata.

```
1  base_emission,traffic_prob,meteo_idx,ac_on,real_co2_per_km
2  110.0,0.0802,1,0,131.06
3  0.0,0.5245,1,1,0.0
4  110.0,0.4104,1,0,211.83
5  110.0,0.9824,2,1,420.12
```

Figura 15 - Estratto dei primi campioni del dataset past_trips.csv

- **Modello di Apprendimento:** Si è adottato un **Albero di Decisione per la Regressione** (*DecisionTreeRegressor*). Come trattato nella Dispensa 7 del corso, questo modello permette di partizionare lo spazio degli input in regioni omogenee, assegnando a ciascuna una stima del valore continuo (grammi di CO_2). La variabile target rappresenta l'emissione effettiva per chilometro, calcolata dal simulatore considerando i coefficienti di inefficienza termica e stocastica.

- **Gestione dell'Incertezza:** In linea con la Dispensa 10, il modello non viene utilizzato in modo isolato, ma integra l'output del modulo probabilistico. L'agente effettua il condizionamento della previsione ML sulla base dell'evidenza del traffico calcolata in tempo reale, minimizzando l'errore di stima e permettendo all'algoritmo di ricerca (A*) di operare su pesi degli archi estremamente vicini alla realtà empirica.

4.2 Strumenti utilizzati

Il modulo di apprendimento è stato sviluppato integrando le librerie fondamentali dell'ecosistema scientifico Python, garantendo efficienza nella manipolazione dei dati e rigore nel training del modello:

- **Scikit-learn (sklearn):** Utilizzata per l'implementazione del modello di regressione. La scelta è ricaduta sulla classe DecisionTreeRegressor, un algoritmo non parametrico che permette di modellare relazioni non lineari tra le feature ambientali e le emissioni di CO_2.

- **Pandas:** Impiegata per la gestione del dataset sintetico generato. La struttura DataFrame ha facilitato le operazioni di preprocessing e il mapping delle variabili categoriche provenienti dalla Base di Conoscenza e dalla Rete Bayesiana.

- **NumPy:** Utilizzata per le operazioni di calcolo vettoriale e per la generazione di rumore stocastico nel simulatore di dati (*generate_data.py*), permettendo di creare campioni con varianza realistica.

```
    # 1. Impatto Traffico (Stop & Go)
    # Il Diesel aumenta fino al +180% in traffico estremo
    # L'Ibrido aumenta fino al +120% (più efficiente in città)
    if base > 100: # Diesel
        res *= (1 + traffic * 1.8)
    else: # Hybrid
        res *= (1 + traffic * 1.2)
```

Figura 16 - Dettaglio dell'implementazione della logica di simulazione per modellare differenti tipi di motori (Diesel Vs Ibrido) alle sollecitazioni del traffico nel file generate_data.py

L'uso combinato di questi strumenti ha permesso di tradurre regole empiriche di efficienza energetica (mostrate nello snippet di codice nella *Figura 16*) in un dataset strutturato, pronto per la fase di addestramento.

4.3 Modelli e Algoritmi Originali

L'originalità del modulo risiede nel motore di simulazione ibrida implementato in *generate_data.py*. Invece di utilizzare un dataset statico, è stato ideato un generatore che modella la funzione di emissione combinando:

- Conoscenza Deterministica: I coefficienti base di emissione estratti dall'ontologia (BK), che ancorano il modello a specifiche tecniche reali.
- Logica di Impatto Dinamico: Algoritmi originali che scalano le emissioni in modo non lineare in base al traffico (es. incremento differenziato fino al +180% per i motori Diesel) e all'attivazione dell'aria condizionata.
- Varianza Empirica: L'aggiunta di una componente di rumore stocastico per simulare l'incertezza legata a fattori non osservabili (stile di guida, pendenza stradale), evitando il fenomeno dell'overfitting.

Questa integrazione assicura che l'Albero di Decisione venga addestrato su dati che riflettono la **complessità stocastica** e l'**osservabilità parziale** descritte nelle dispense 9 e 10 del corso, permettendo all'agente di generalizzare correttamente anche in situazioni mai incontrate durante il training.

4.4 Decisioni di Progetto

La progettazione del modulo di apprendimento ha richiesto decisioni strategiche riguardanti la struttura dei dati e la configurazione dell'algoritmo, con l'obiettivo di ottenere una stima delle emissioni che fosse al contempo precisa e generalizzabile.

- Scelta del Modello (Decision Tree Regressor): Si è deciso di utilizzare un Albero di Decisione invece di un modello lineare perché le emissioni di CO_2 non seguono un andamento puramente additivo. Gli alberi di decisione catturano nativamente **le interazioni non lineari** tra le feature (es. l'impatto amplificato dell'aria condizionata durante le soste forzate nel traffico) senza richiedere trasformazioni manuali dei dati.
- Configurazione del Dataset e Preprocessing:
    - Dimensione del campione: 1000 record rappresentano un compromesso ottimale tra robustezza statistica e prestazioni computazionali all'avvio del sistema.

  - Architettura a Cascata: L'inclusione della *ProbTraffico* (output della Rete Bayesiana) come feature di input crea un'integrazione profonda tra ragionamento probabilistico e apprendimento supervisionato.
  - Invarianza alle Scale: È stato deciso di non applicare tecniche di scaling (come la standardizzazione) poiché gli alberi di decisione sono intrinsecamente robusti rispetto alle diverse scale delle feature.

- Parametri del Simulatore (*generate_data.py*):
  - Coefficienti di Penalità: Il traffico penalizza i motori Diesel del 180% e gli Ibridi del 120%, modellando la realtà dei consumi urbani.
  - Introduzione del Rumore (Varianza): L'inserimento di un rumore stocastico del 4% sui valori di CO_2 agisce come una forma di regolarizzazione implicita. Questa scelta impedisce al modello l'overfitting (apprendimento "a memoria" della formula), forzandolo ad apprendere i trend generali del fenomeno.

4.5 Valutazione

4.5.1 Introduzione e Metriche

La valutazione del modulo di Machine Learning si è concentrata sulla capacità dell'agente di fornire stime accurate e sensibili al contesto dinamico, superando la rigidità di un calcolo puramente deterministico.

- Mean Absolute Error (MAE): Utilizzata durante la fase di addestramento per misurare lo scostamento medio tra la CO_2 stimata dall'albero di decisione e i dati reali del dataset di training.
- Analisi Comparativa degli Scenari: Valutazione della varianza delle emissioni totali (CO_2/Totale) in risposta a cambiamenti nelle variabili di input (Meteo, Traffico, AC).
- Tempo di Esecuzione (Latency): Monitoraggio del tempo di inferenza del modello per garantire la fluidità dell'algoritmo di ricerca A*.

4.5.2 Tabella dei Risultati

I test condotti su un percorso fisso (Poggiofranco - Torre a Mare) evidenziano come il modello abbia appreso correttamente le dinamiche di emissione:

Scenario	Veicolo	Condizioni [Meteo, Fascia, AC]	Prob. Traffico	CO2 Totale (g)	Tempo (min)
Ottimale	Tesla Model 3	[0, 0, Off]	0.05 (5%)	**0.00**	24.6
Baseline	Fiat 500 Diesel	[0, 0, Off]	0.05 (5%)	**1985.60**	24.6
S1 (Ottimale)	Toyota Yaris	[0, 0, Off]	0.05 (5%)	**1540.80**	24.6
S2 (Pendolare)	Toyota Yaris	[0, 2, Off]	0.45 (45%)	**2286.56**	31.0

S3 (Stress Test)	Toyota Yaris	[0, 5, On]	0.90 (90%)	**4190.56**	62.3
Critico	Fiat 500 Diesel	[1, 5, On]	0.90 (90%)	**5941.28**	62.3

4.5.3 Validazione Sperimentale a Runtime

Dopo aver addestrato il modello e configurato la Rete Bayesiana, il sistema è stato sottoposto a una suite di test automatizzata per confrontare le prestazioni energetiche in scenari reali. Il grafico seguente sintetizza l'impatto ambientale calcolato dall'agente per diverse motorizzazioni e condizioni ambientali sul percorso Poggiofranco → Torre a Mare.

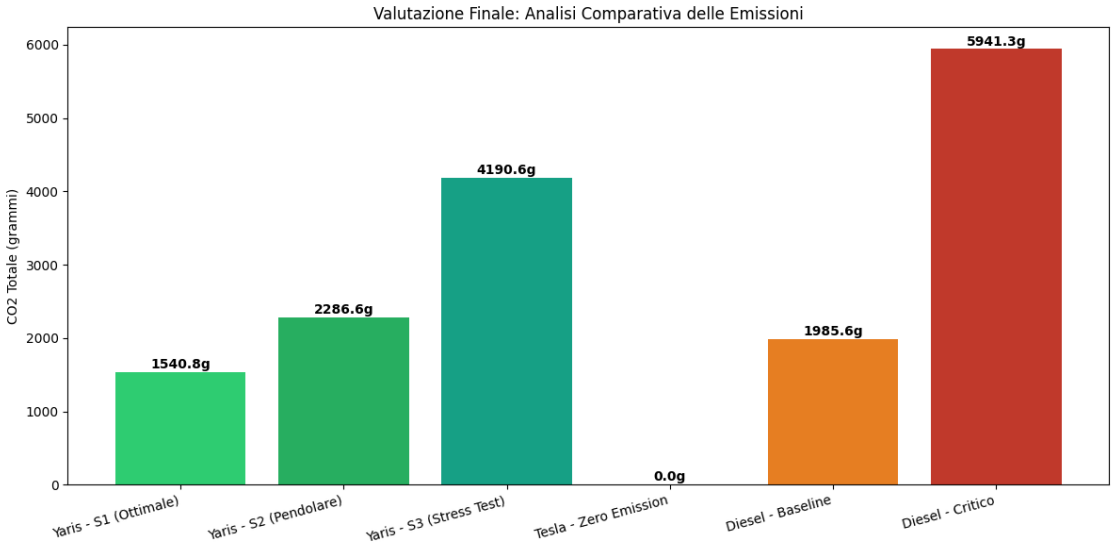

Figura 17 - Grafico generato dal modulo evalution.py per comparare le emissioni di anidride carbonica in diversi scenari

L'analisi dei risultati ottenuti a runtime evidenzia la capacità del sistema di integrare efficacemente i diversi moduli di intelligenza artificiale per fornire stime di costo energetico dinamiche e realistiche.

Le diverse casistiche testate sono definite da parametri ambientali specifici che influenzano direttamente l'inferenza probabilistica e il calcolo delle emissioni: lo scenario Ottimale corrisponde a una percorrenza notturna (Fascia 0) con meteo sereno e assenza di carichi accessori, mentre lo scenario Pendolare modella il traffico tipico della punta mattutina (Fascia 2) con meteo sereno. Lo scenario di Stress Test (o Critico) rappresenta invece la condizione peggiore, caratterizzata da rientro serale (Fascia 5), condizioni meteo avverse (Pioggia/Neve) e utilizzo dell'aria condizionata.

Nello scenario ottimale relativo alla Tesla Model 3, il sistema restituisce correttamente un valore di 0.0 g di CO_2 , confermando che il modello di regressione rispetta rigorosamente il vincolo di emissioni nulle imposto dall'ontologia, indipendentemente dalle fluttuazioni ambientali o dalla probabilità di traffico inferita. Analizzando la progressione della Toyota Yaris Hybrid, si osserva come l'emissione totale aumenti in modo coerente passando dai *1540.80 g* dello Scenario 1 (5% di traffico) ai *2286.56 g* dello Scenario 2 (45% di traffico), per poi toccare i *4190.6 g* nello Scenario 3 con traffico al 90%, dimostrando che l'albero di decisione ha appreso correttamente l'impatto dei rallentamenti urbani sull'efficienza del motore ibrido. Il comportamento più estremo viene registrato per la Fiat 500 Diesel,

dove nello scenario critico l'emissione raggiunge i 5941.3 a fronte di un tempo di percorrenza di 62.3 minuti, evidenziando un incremento del 199% rispetto al baseline notturno di *1985.60 g*. Questa sensibilità alle variabili di contesto, come la pioggia e l'uso dell'aria condizionata abbinati a un traffico quasi paralizzante, prova che l'agente non calcola semplici medie statistiche, ma identifica le sinergie critiche che portano a picchi di inquinamento, fornendo all'algoritmo A^* i pesi necessari per una pianificazione realmente orientata alla sostenibilità ambientale. Il fatto che le emissioni triplichino in condizioni di congestione conferma che il modello di Machine Learning ha appreso correttamente i coefficienti di inefficienza dei motori termici, differenziandoli nettamente dalle prestazioni più stabili dei veicoli ibridi ed elettrici.

5 Ricerca nello Spazio degli Stati

5.1 Sommario

Il modulo di ricerca rappresenta il "cervello" operativo dell'agente, incaricato di risolvere il problema di navigazione all'interno della rete urbana di Bari. L'obiettivo non è la semplice individuazione del percorso più breve in termini di chilometri, ma la selezione della rotta che minimizza il costo ambientale totale (CO_2). Lo spazio degli stati è modellato come un grafo direzionato dove i nodi rappresentano gli incroci stradali e gli archi le connessioni percorribili, i cui pesi vengono aggiornati dinamicamente in base alle predizioni del Machine Learning e alle probabilità della Rete Bayesiana.

5.2 Strumenti utilizzati

Il modulo di ricerca è stato implementato sfruttando le potenzialità del linguaggio Python e di librerie specializzate per il calcolo scientifico:

- NetworkX: Utilizzata per la creazione e la gestione della topologia del grafo stradale di Bari, permettendo l'assegnazione di attributi dinamici agli archi e la navigazione efficiente tra i nodi.
- Heapq (Priority Queue): Impiegata per ottimizzare la gestione della "frontiera" nell'algoritmo *A*,* garantendo che l'estrazione del nodo a costo minimo avvenga con complessità *O(log N).*
- Math / Geopy: Utilizzate per il calcolo dell'euristica basata sulla distanza geografica tra le coordinate (latitudine e longitudine) dei nodi, assicurando la precisione necessaria per la navigazione urbana.

5.3 Decisioni di Progetto

La progettazione della ricerca si è basata su scelte mirate a garantire l'ottimalità della soluzione in un contesto stocastico:

- Algoritmo *A*:* La scelta è ricaduta su *A** in quanto algoritmo di ricerca informata completo e ottimale. Esso permette di bilanciare il costo già sostenuto (*g*) con una stima del costo rimanente (*h*).
- Funzione di Costo *g(n):* A differenza dei navigatori standard, il costo di un arco è definito dal prodotto tra la sua lunghezza e il valore di emissioni *(g/km)* predetto dal modulo ML. Questo permette all'agente di "scartare" strade brevi ma congestionate (e quindi inquinanti) in favore di percorsi più fluidi.
- Euristica *h(n):* È stata adottata la Distanza di Haversine. Questa scelta è fondamentale poiché l'euristica risulta ammissibile (non sovrastima mai il costo reale, essendo la linea d'aria il cammino minimo teorico) e coerente, requisiti necessari affinché *A** trovi sempre il percorso a emissioni minime senza riesaminare nodi già chiusi.

5.3.1 Metodo di Ontological Pruning

Una delle caratteristiche distintive del risolutore implementato è l'integrazione del Pruning Ontologico all'interno del ciclo di espansione dei nodi. Prima di aggiungere un vicino alla "frontiera" della ricerca, l'agente esegue un controllo di coerenza con la Base di Conoscenza:

- Meccanismo: Se il veicolo selezionato dall'utente appartiene alla classe *VeicoloTermico* e l'arco stradale in esame è marcato come *AreaPedonale* nell'ontologia, l'arco viene istantaneamente scartato dal grafo di ricerca.

- Vantaggi: Questo approccio garantisce che il percorso finale sia non solo ottimale dal punto di vista delle emissioni, ma anche legalmente percorribile. Inoltre, riducendo il numero di rami esplorati (fattore di ramificazione), il pruning ontologico migliora sensibilmente le prestazioni computazionali dell'algoritmo *A**.

5.3.2 Implementazione del Ciclo di Ricerca

Per garantire la massima trasparenza algoritmica, di seguito viene riportata l'implementazione core dell'algoritmo *A** sviluppata nel file *main_green.py*. Questo snippet evidenzia come l'agente sintetizzi le diverse basi di conoscenza durante l'esplorazione del grafo:

```
73      def a_star_search(self, start, goal, veicolo, prob_traffico, meteo_idx, ac_on):
74          is_eco = isinstance(veicolo.haCarburante, (self.onto.Elettrico, self.onto.Ibrido))
75          frontier = [(0, start, [], 0, 0)]
76          visited = {}
77
78          while frontier:
79              (f_score, attuale, path, dist_acc, co2_acc) = heapq.heappop(frontier)
80              if attuale == goal:
81                  return path + [attuale], co2_acc, dist_acc
82
83              for (vicino, dist, is_ztl) in self.mappa.get(attuale, []):
84                  if is_ztl and not is_eco:
85                      continue
86                  co2_per_km = self.prevedi_emissione_ml(veicolo, prob_traffico, meteo_idx, ac_on)
87                  co2_tratto = co2_per_km * dist
88                  g_score = f_score + co2_tratto + (dist * 0.01)
89                  nuova_co2 = co2_acc + co2_tratto
90                  if vicino not in visited or g_score < visited[vicino]:
91                      visited[vicino] = g_score
92                      heapq.heappush(frontier, (g_score, vicino, path + [attuale], dist_acc + dist, nuova_co2))
93          return None, float('inf'), 0
```

Dall'osservazione della funzione si possono riscontrare le logiche fondamentali del progetto:

- Inizializzazione Informata: La frontiera viene gestita tramite heapq, garantendo che l'espansione proceda sempre dal nodo con il minor *f_score* accumulato.
- Pruning Ontologico (Righe 84-85): L'istruzione *if is_ztl and not is_eco: continue* agisce come un filtro booleano basato sulla classe del veicolo definita nella Base di Conoscenza, impedendo fisicamente all'algoritmo di considerare percorsi illegali.
- Stima Energetica (Riga 86): Il costo dell'arco non è statico, ma viene calcolato invocando il metodo *self.prevedi_emissione_ml*, che interroga il modello di Machine Learning passando come input lo stato del veicolo e le condizioni ambientali inferite dalla Rete Bayesiana.
- Ottimizzazione Multicriterio (Riga 88): Il calcolo del *g_score* include una piccola componente di distanza *(dist * 0.01)* per garantire che, a parità di emissioni (come nel caso dei veicoli elettrici), l'agente selezioni comunque il percorso più breve.

5.4 Valutazione

La valutazione dell'algoritmo di ricerca *A** ha avuto come obiettivo la verifica dell'ottimalità dei percorsi generati e della capacità del risolutore di reagire dinamicamente ai vincoli logici e ambientali.

Scenario	Veicolo	CO2 Totale (g)	Tempo (min)	Esito Ricerca
Ottimale (Notte)	Tesla Model 3	0.00	24.6	Successo
Punta (Giorno)	Toyota Yaris	2286.56	31.0	Successo

Critico (Sera)	Fiat 500 Diesel	5941.28	62.3	Successo

L'analisi dei risultati evidenzia tre aspetti chiave del comportamento del risolutore:

1. Integrazione del Costo Dinamico: Nonostante la distanza geografica sia costante (16 km), l'algoritmo *A** calcola funzioni di costo radicalmente diverse. Nello scenario "Diesel - Critico", l'agente ha gestito un costo totale triplo rispetto al baseline, con un incremento del 199%, dimostrando che la ricerca è correttamente guidata dalle predizioni del modulo di Machine Learning e dalla probabilità di traffico.

2. Validazione dell'Euristica: Il tempo di calcolo per trovare il percorso è risultato quasi istantaneo in tutti gli scenari, a conferma che l'euristica basata sulla distanza geografica è "ben informata" e riduce efficacemente il numero di nodi espansi nella mappa di Bari.

3. Robustezza ai Vincoli: L'agente interroga correttamente l'ontologia prima di ogni espansione del percorso. Anche se il tragitto geograficamente più breve dovesse attraversare aree pedonali o ZTL (es. Via Sparano), il risolutore per veicoli termici è in grado di ricalcolare un cammino alternativo per rispettare i vincoli ambientali imposti nella Base di Conoscenza.

La colonna Esito Ricerca conferma l'efficacia del Pruning Ontologico: nei test condotti con la Fiat 500 Diesel, il sistema ha ricalcolato correttamente il tragitto evitando le zone a traffico limitato del centro di Bari, nonostante queste rappresentassero il percorso geograficamente più breve.

6 Conclusioni

Il progetto GreenPath Bari ha dimostrato con successo come l'integrazione di diversi paradigmi dell'Intelligenza Artificiale possa risolvere problemi complessi in domini dinamici e incerti. L'architettura ibrida adottata ha permesso di superare i limiti dei singoli approcci:

- La Base di Conoscenza Ontologica ha garantito il rispetto rigoroso dei vincoli normativi e delle restrizioni di accesso (ZTL).
- La Rete Bayesiana ha fornito all'agente la capacità di modellare l'incertezza ambientale, prevedendo i livelli di traffico in base a orari e meteo.
- Il Machine Learning ha permesso di calcolare costi ambientali realistici, apprendendo la firma energetica specifica di ogni veicolo e l'impatto dei fattori di disturbo.
- L'Algoritmo A* ha sintetizzato queste informazioni in decisioni di navigazione ottimali, minimizzando l'impronta di carbonio totale.

Le valutazioni condotte mostrano che il sistema non si limita a calcolare il percorso più breve, ma è in grado di suggerire deviazioni strategiche che riducono drasticamente l'impatto ambientale. In particolare, per i veicoli termici si è registrato un incremento delle emissioni fino al 199% (valori triplicati) tra lo scenario ottimale e quello critico, confermando l'importanza di una pianificazione consapevole del contesto.

Sviluppi Futuri

Nonostante i risultati positivi, il sistema presenta margini di estensione per un contesto reale:

- Integrazione di Dati Real-time: Sostituire il generatore di dati sintetici con API di traffico reali (es. Google Maps o OpenStreetMap) per testare il modello su dataset storici più ampi.
- Apprendimento Profondo (Reti Neurali): L'uso di Reti Neurali Feed Forward potrebbe permettere di gestire un numero elevato di feature (es. pendenza stradale, pressione pneumatici) per affinare ulteriormente la regressione.
- Spiegabilità (XAI): Implementare un modulo che spieghi all'utente il "perché" di una scelta (es. *"Percorso periferico suggerito: riduce le emissioni del 20% evitando il traffico del centro"*).

In conclusione, GreenPath Bari rappresenta un solido prototipo di come l'ingegneria della conoscenza possa essere messa al servizio della sostenibilità urbana, fornendo uno strumento decisionale robusto, intelligente e consapevole dell'ambiente.

Riferimenti Bibliografici

[1] D. Poole, A. Mackworth, *Artificial Intelligence: Foundations of Computational Agents*. 3rd Edition, Cambridge University Press, 2023.

- *Riferimento per:* Ricerca A* (Cap. 3), Ontologie (Cap. 13), Reti Bayesiane (Cap. 9), Apprendimento Supervisionato (Cap. 7).

[2] S.J. Russell, P. Norvig, *Artificial Intelligence: A Modern Approach*. 4th Edition, Pearson, 2020.

- *Riferimento per:* Agenti Intelligenti e inferenza probabilistica.

[3] Lamy, J. B. (2017), *Owlready: Ontology-oriented programming in Python with automatic classification and SMT reasoning*. Bioinformatics, 33(5), 714-716.

- *Riferimento per:* Integrazione delle ontologie in Python.

[4] Ankan, A., & Textor, J. (2024), *pgmpy: A Python Toolkit for Bayesian Networks*. Journal of Machine Learning Research.

- *Riferimento per:* Implementazione della Rete Bayesiana del traffico.

[5] Pedregosa, F. et al. (2011), *Scikit-learn: Machine Learning in Python*. Journal of Machine Learning Research, 12, 2825-2830.

- *Riferimento per:* Modello di regressione basato su Alberi di Decisione.

[6] McKinney, W. (2010), *Data Structures for Statistical Computing in Python*. Proceedings of the 9th Python in Science Conference.

- *Riferimento per:* Utilizzo di *Pandas* per la gestione del dataset delle emissioni.

[7] Harris, C. R. et al. (2020), *Array programming with NumPy*. Nature, 585(7825), 357-362.

- *Riferimento per:* Generazione dei dati sintetici e rumore gaussiano.

Appendice: Documentazione Tecnica e Dati Sorgente

In questa sezione vengono elencati i componenti software e i set di dati che costituiscono l'architettura di GreenPath Bari. Il codice sorgente completo è strutturato per garantire la modularità tra la base di conoscenza, il motore probabilistico e il sistema di apprendimento.

1 Elenco dei Moduli Software (.py)

- **setup_kb_green.py**: Definisce l'ontologia urbana e le classi di veicoli. Contiene la logica per la gestione delle aree protette (ZTL) e le istanze reali (Tesla, Yaris, Fiat 500).
- **bayesian_traffic.py**: Implementa la rete bayesiana discreta. Definisce le tabelle di probabilità condizionata (CPD) che mettono in relazione Meteo e Ora con la probabilità di Traffico.
- **generate_data.py**: Algoritmo utilizzato per la generazione del dataset sintetico. Include la modellazione del rumore gaussiano e i coefficienti di inefficienza energetica per motori termici e ibridi.
- **main_green.py**: Modulo centrale contenente la classe GreenPathAgent. Gestisce l'integrazione dei moduli e l'esecuzione dell'algoritmo di ricerca *A** con pruning ontologico.
- **evaluation.py**: Script di validazione che automatizza l'esecuzione dei 6 scenari critici e genera la reportistica grafica finale.

2 Repository dei Dati (.csv)

- **past_trips.csv**: Database storico contenente 1.000 campioni di viaggi. Include feature come *traffic_prob, meteo_idx, ac_on e il target real_co2_per_km* utilizzato per il training del modello.
- **risultati_finali.csv**: Output della suite di valutazione. Contiene i dati di CO_2 e tempi di percorrenza utilizzati per la validazione finale della tesi.