Automatizzazione degli aiuti per sinistri stradali

Gruppo di lavoro

- Giulio evangelista, 758335, g.evangelista13@studenti.uniba.it

Link repo su GitHub

AA 2024-25

Sommario

1. Introduzione al Progetto 3
1.1. Contesto e Obiettivi 3
1.2. Architettura Generale del Sistema 3
2. Acquisizione dei Dati e Riconoscimento Visivo (Deep Learning) 4
2.1. Il Modello YOLOe e il Dataset di Immagini 4
2.2. Funzionamento di YOLOe e Output 4
2.3. Cenni Teorici: Object Detection 7
3. Rappresentazione della Conoscenza con Knowledge Graph e Ontologie 7
3.1. Progettazione dell'Ontologia per gli Incidenti Stradali 7
3.3. Popolamento del Knowledge Graph (script build_kg.py) 8
4. Ragionamento Simbolico e Inferenza (OWL & Prolog) 9
4.1. Inferenza con Reasoner HermiT 9
4.1.1. Regole di Inferenza Utilizzate 9
4.2. Ragionamento Relazionale con Prolog 11
4.2.1. Trasformazione del Knowledge Graph in Fatti Prolog (script extraction_from_kg.py) 11
4.2.2. Ruolo di Prolog nell'Estrazione di Conoscenza Aggiuntiva 12
5. Risoluzione di Problemi con Vincoli (CSP) per l'Assegnamento Risorse 12
5.1. Definizione del Problema di Assegnamento Ambulanze 12
5.2. Implementazione del CSP con Prolog 13
5.3. Risultati e Discussione dell'Assegnamento 14
6. Conclusioni e Lavoro Futuro 14
6.1. Valutazione dell'Approccio Ibrido 14
6.2. Limiti del Progetto Attuale 15
6.3. Possibili Sviluppi Futuri 15

1. Introduzione al Progetto

1.1. Contesto e Obiettivi

La gestione tempestiva ed efficiente degli incidenti stradali è di fondamentale importanza per minimizzare i danni a persone e veicoli, decongestionare il traffico e ottimizzare l'allocazione delle risorse di soccorso. I sistemi attuali spesso si basano su segnalazioni manuali o su analisi frammentate. Il presente progetto si propone di sviluppare un prototipo di sistema intelligente che, grazie a delle immagini registrate da un drone che arriva sulla scena dell'incidente, integri capacità di percezione visiva (tramite Deep Learning) con la rappresentazione e il ragionamento della conoscenza (tramite Knowledge Graph e Programmazione Logica) per fornire un supporto decisionale avanzato nella gestione degli incidenti.

L'obiettivo principale è dimostrare come l'integrazione di diverse tecniche di Intelligenza Artificiale (AI) possa portare a un sistema più robusto e "intelligente" rispetto a un approccio monolitico. Nello specifico, il progetto mira a:

- Automatizzare l'identificazione di entità rilevanti (veicoli incidentati/non incidentati, persone) da immagini.
- Costruire una base di conoscenza strutturata (Knowledge Graph) che rappresenti gli scenari di incidente.
- Eseguire inferenze semantiche per categorizzare gli incidenti e derivare nuove informazioni implicite.
- Utilizzare il ragionamento logico per risolvere problemi complessi di allocazione delle risorse, come l'assegnamento delle ambulanze.

1.2. Architettura Generale del Sistema

Il sistema è concepito come una pipeline modulare composta da diverse fasi integrate, come illustrato di seguito:

1. **Modulo di Riconoscimento Visivo (YOLOe):** Elabora immagini di incidenti stradali per rilevare e classificare oggetti chiave quali auto_incidentata, auto_normale e persona.
2. **Modulo di Costruzione del Knowledge Graph:** I risultati delle rilevazioni di YOLOe vengono utilizzati per popolare un Knowledge Graph (KG),

modellato tramite un'ontologia OWL, trasformando i dati visivi grezzi in conoscenza strutturata.
3. **Modulo di Inferenza OWL:** Un reasoner OWL viene applicato al KG per derivare nuova conoscenza implicita basata su regole semantiche predefinite, arricchendo la base di conoscenza.
4. **Modulo di Ragionamento Logico (Prolog):** La conoscenza inferita dal KG viene trasformata in fatti Prolog. Prolog viene quindi utilizzato per applicare ulteriori regole di ragionamento, culminando nella risoluzione di un Problema di Soddisfacimento di Vincoli (CSP) per l'assegnamento ottimale delle ambulanze.

Pertanto, avremo i seguenti collegamenti in base ai riferimenti del programma di studio:

1. **Reti neurali e Apprendimento Profondo**
2. **Knowledge Graph e Ontologie**
3. **Ragionamento con Vincoli**

2. Acquisizione dei Dati e Riconoscimento Visivo (Deep Learning)

2.1. Il Modello YOLOe e il Dataset di Immagini

La fase iniziale del progetto si basa sull'analisi delle immagini per estrarre informazioni grezze sugli oggetti presenti nella scena dell'incidente. A tal fine, è stato impiegato un modello di Deep Learning basato sull'architettura YOLO (You Only Look Once), in particolare nella sua variante YOLOe.

Il modello YOLOe è stato sottoposto a un processo di **fine-tuning** su un dataset personalizzato di immagini raffiguranti incidenti stradali. Questo dataset è stato creato con l'obiettivo di coprire una varietà di scenari e condizioni e opportunamente diviso in training, validation e test set. Per lo split ho seguito un 80-10-10 rispettivamente per training, validation e test set.

2.2. Funzionamento di YOLOe e Output

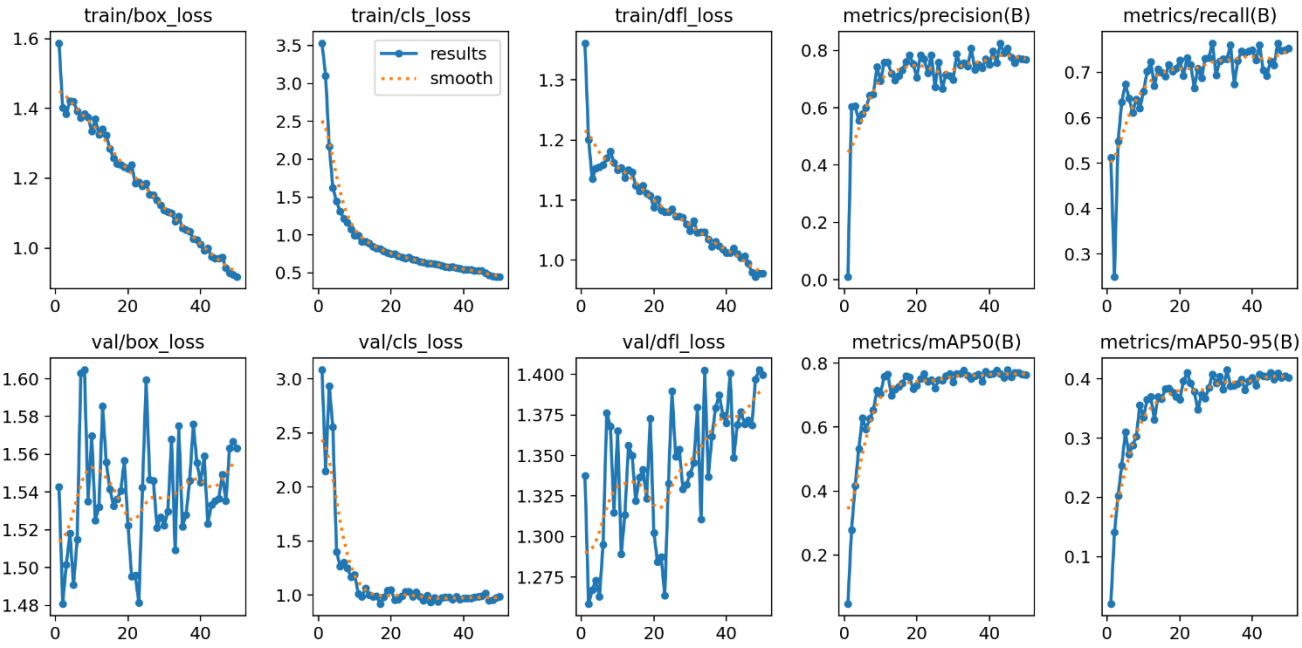
Una volta addestrato, il modello YOLOe è stato eseguito sul dataset di immagini di incidenti stradali. La sua funzione principale è stata quella di identificare e categorizzare gli oggetti rilevanti, generando per ciascuno:

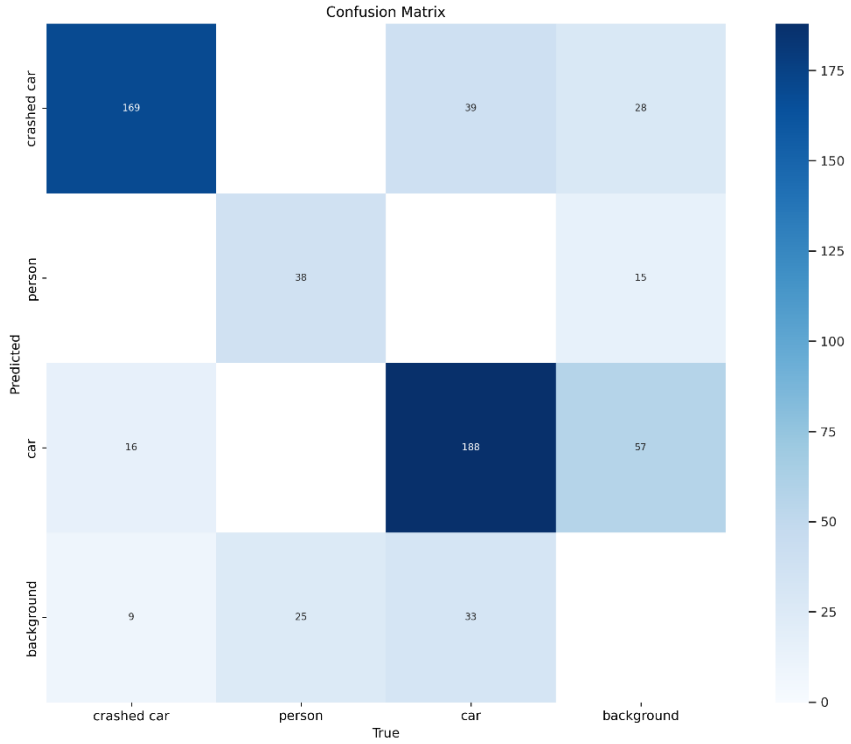
- **Classe dell'oggetto:** Il modello è stato specificamente addestrato per distinguere tra:
    - Auto incidentata (crashed_car)

    - Auto normale (car)
    - Persona (person)
- **Bounding Box:** Le coordinate del rettangolo di delimitazione che individua l'oggetto nell'immagine.

L'output di YOLOe per ciascuna immagine viene salvato in formato YOLO (file .txt), dove ogni riga rappresenta un oggetto rilevato con il suo class_id, x_center, y_center, width, height.

Dopo aver fatto fine tuning è stato possibile notare un netto miglioramento delle performance; sono riportati di seguito i principali grafici di valutazione, matrice di confusione e mosaici:




Confusion Matrix
crashed car
person
car
background
Predicted
True
169
39
28
38
15
16
188
57
9
25
33
175
150
125
100
75
50
25
0


car 0.4
crashed car 0.3
person 0.3
crashed car 0.3
person 0.5
person 0.3
crashed car 0.7
77.jpg
car 0.5
car 0.3
car 0.7
crashed car 0.8
crashed car 0.5
crashed car 0.9
car 0.8
74.jpg
car 0.4
car 0.3
crashed car 0.5
78.jpg
car 0.3
car 0.6
crashed car 0.9
81.jpg
crashed car 0.8
car 0.3
75.jpg
crashed car 0.9
79.jpg
crashed car 0.3
crashed car 0.8
82.jpg
crashed car 0.9
crashed car 0.9
crashed car 0.9
crashed car 0.9
crashed car 0.8
crashed car 0.8
car 0.4
car 0.4
car 0.9
crashed car 0.9
crashed car 0.9

2.3. Cenni Teorici: Object Detection

L'**Object Detection** è un task del Computer Vision che combina il riconoscimento di oggetti (classificazione) con la loro localizzazione spaziale all'interno di un'immagine. Algoritmi come YOLO sono noti per la loro velocità ed efficienza. YOLO divide l'immagine in una griglia e predice simultaneamente le bounding box e le probabilità di classe per ciascuna cella della griglia, rendendolo particolarmente adatto per applicazioni in tempo reale.

Questo modulo, pur essendo fondamentale per l'acquisizione dei dati, non è il focus principale del progetto, che si concentra sulle fasi di Ingegneria della Conoscenza successive.

3. Rappresentazione della Conoscenza con Knowledge Graph e Ontologie

3.1. Progettazione dell'Ontologia per gli Incidenti Stradali

L'ontologia del progetto, definita nel file kg.ttl, è stata progettata per modellare il dominio degli incidenti stradali, focalizzandosi sugli elementi rilevati da YOLOe e sulle relazioni necessarie per il ragionamento. È stato usato il sw Protege come GUI per interfacciarsi col knowledge graph una volta creato.

Classi Principali:

- ex:Auto: Classe generica per i veicoli a quattro ruote.
- ex:AutoIncidentata: Una sottoclasse di ex:Auto, specificamente rilevata come danneggiata dal modello YOLOe. Questo dimostra la capacità di YOLOe di fornire una categorizzazione già semantica a un certo livello.
- ex:Persona: Rappresenta gli individui coinvolti o presenti nella scena dell'incidente.
- ex:PersonaFerita: Una sottoclasse di ex:Persona. Questa classe è inferita, non direttamente rilevata da YOLOe, ma è un risultato del ragionamento sulle condizioni.
- ex:Incidente: Rappresenta un evento incidente stradale. Ogni set di rilevazioni da una singola immagine o scenario viene modellato come un'istanza di Incidente.

- ex:IncidenteGrave: sottoclasse di ex:Incidente anch’essa inferita col reasoner tramite le regole.

Proprietà:

- ex:coinvolge: Proprietà generica che collega un ex:Incidente a ex:Persona
- ex:vicino: Proprietà che indica una relazione spaziale di prossimità tra una ex:Persona e un ex:Auto. Questa proprietà viene inferita tramite un calcolo euristico basato sulle bounding box.

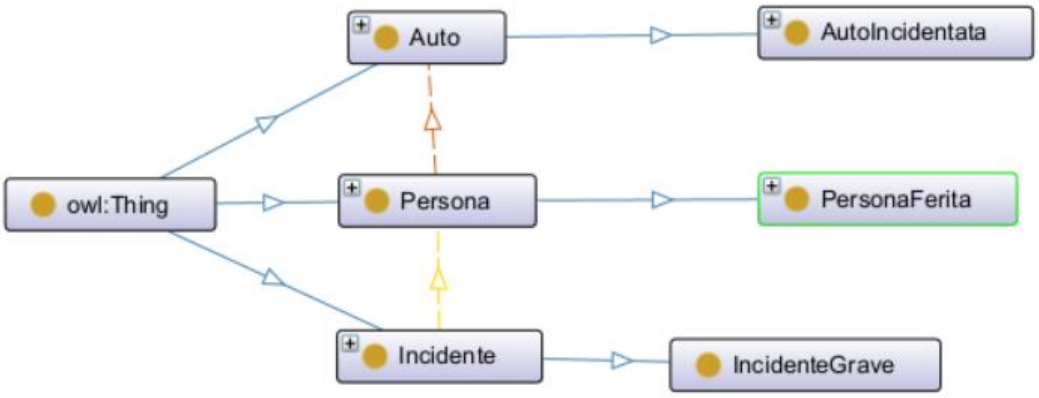
3.3. Popolamento del Knowledge Graph (script build_kg.py)

Lo script Python build_kg.py è responsabile della costruzione del Knowledge Graph. Esso prende in input i file di output di YOLOe (in formato .txt, contenenti le rilevazioni per ogni immagine) e li traduce in fatti OWL/RDF che popolano l'ontologia.

Il processo di popolamento include:

1. **Parsing dell'Output YOLOe:** Per ogni file .txt di YOLOe (corrispondente a un'immagine/scenario), lo script legge le rilevazioni.
2. **Creazione delle Istanze:**
   - Viene creata un'istanza di ex:Incidente per ogni immagine processata. Per ogni rilevazione di AutoIncidentata, Auto o Persona, viene creata un'istanza corrispondente (es. ex:car_inc_img_1_0, ex:person_img_1_1).
3. **Aggiunta delle Proprietà:**
   - Le istanze di Incidente e Persona vengono collegate tramite la proprietà ex:coinvolge.
   - Viene calcolata la vicinanza tra le persone e le auto basandosi sulle coordinate delle bounding box (se la distanza tra i centri delle bounding box è inferiore a una soglia definita, come THRESHOLD = 0.2). Se vicini, viene aggiunta la relazione ex:vicino tra la Persona e l'Auto.
4. **Serializzazione del KG:** Il grafo popolato viene salvato nel file kg.ttl in formato Turtle, pronto per l'inferenza.

Di seguito è riportata la rappresentazione delle classi e proprietà del knowledge graph:



4. Ragionamento Simbolico e Inferenza (OWL & Prolog)

Questa fase rappresenta il cuore del processo di estrazione della conoscenza, dove fatti espliciti vengono arricchiti con conoscenza implicita tramite motori di ragionamento.

4.1. Inferenza con Reasoner HermiT

Una volta popolato il Knowledge Graph, il reasoner HermiT viene utilizzato per dedurre nuove conoscenze basate sugli assiomi definiti nell'ontologia.

4.1.1. Regole di Inferenza Utilizzate

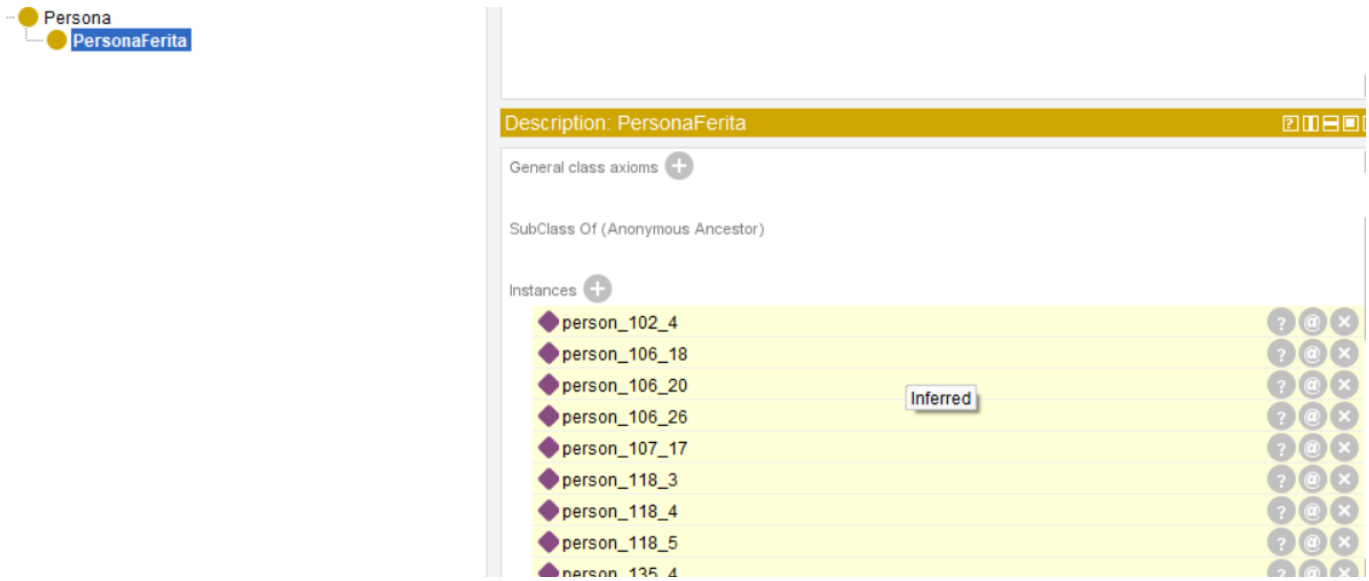
Nel kg.ttl sono presenti due regole SWRL (Semantic Web Rule Language) principali, che permettono al reasoner di inferire importanti classificazioni:

1. **Regola per IncidenteGrave:**
    - **Linguaggio Naturale:** "Se ci sono più di 2 auto diverse nello stesso incidente, allora è un incidente grave."
    - **Formato SWRL:**

ex:AutoIncidentata(?a1) ^ ex:AutoIncidentata(?a2) ^
ex:AutoIncidentata(?a3) ^ differentFrom(?a1, ?a2) ^
differentFrom(?a1, ?a3) ^ differentFrom(?a2, ?a3) ^
ex:coinvolge(?i, ?a1) ^ ex:coinvolge(?i, ?a2) ^ ex:coinvolge(?i, ?a3)
^ ex:Incidente(?i) -> ex:IncidenteGrave(?i)

2. **Regola per PersonaFerita:**
   - **Linguaggio Naturale:** “Se una persona è vicina ad un’auto incidentata, allora sarà una persona ferita.”
   - **Formato SWRL:**
     ex:Persona(?p) ^ ex:vicino(?p, ?a) ^ ex:AutoIncidentata(?a) ->
     ex:PersonaFerita(?p)

Di seguito è riportato un esempio di ciò che il sistema ha inferito:



4.2. Ragionamento Relazionale con Prolog

Il modulo successivo sfrutta la conoscenza inferita dal Knowledge Graph per eseguire ragionamenti più procedurali e specifici del problema, in particolare la risoluzione di un CSP.

4.2.1. Trasformazione del Knowledge Graph in Fatti Prolog (script extraction_from_kg.py)

Lo script Python extraction_from_kg.py funge da ponte tra il Knowledge Graph OWL e Prolog. Esso carica il kg.ttl (che si assume contenga già le inferenze del reasoner OWL) ed estrae i dati rilevanti, traducendoli in fatti Prolog.

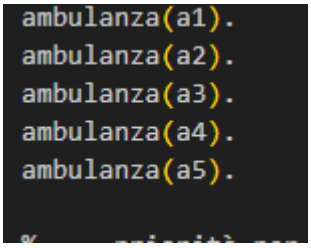
Lo script estrae:

- Tutti gli Incidente rilevati.
- Per ogni incidente, la sua gravità (inferita da OWL: grave se è IncidenteGrave, altrimenti normale).
- Il numero di PersonaFerita coinvolte in ciascun incidente.

Questi dati vengono poi scritti nel file csp_ambulanze.pl sotto forma di fatti

incidente('URI_incidente', gravita, numero_feriti).

```
incidente('http://example.org/incidente#img_96', normale, 0).
incidente('http://example.org/incidente#img_924', normale, 0).
incidente('http://example.org/incidente#img_25', normale, 3).
incidente('http://example.org/incidente#img_121', normale, 0).
incidente('http://example.org/incidente#img_134', normale, 0).
incidente('http://example.org/incidente#img_132', normale, 0).
incidente('http://example.org/incidente#img_123', normale, 0).
incidente('http://example.org/incidente#img_135', normale, 0).
incidente('http://example.org/incidente#img_133', normale, 0).
incidente('http://example.org/incidente#img_156', normale, 0).
incidente('http://example.org/incidente#img_925', normale, 0).
incidente('http://example.org/incidente#img_100', normale, 0).
incidente('http://example.org/incidente#img_28', normale, 0).
incidente('http://example.org/incidente#img_928', normale, 1).
incidente('http://example.org/incidente#img_105', normale, 0).
incidente('http://example.org/incidente#img_112', normale, 0).
incidente('http://example.org/incidente#img_106', normale, 0).
incidente('http://example.org/incidente#img_23', normale, 4).
```



4.2.2. Ruolo di Prolog nell'Estrazione di Conoscenza Aggiuntiva

In csp_ambulanze.pl, Prolog non solo riceve i fatti estratti dal KG, ma definisce anche regole aggiuntive per:

- Prioritizzare gli Incidenti:
  - La regola priorita(Gravita, Valore) assegna un valore numerico di priorità (1 per grave, 2 per normale).
  - La regola ordina_incidenti_priorita utilizza predsort per ordinare gli incidenti in base a questa priorità, garantendo che quelli più gravi vengano considerati per primi nell'assegnamento delle risorse.
- Determinare il Numero di Ambulanze Richieste:
  - La regola NumAmb = 2 ; NumAmb = 1 all'interno di assegna_ambulanze indica che se il numero di feriti è maggiore di 2, l'incidente richiede 2 ambulanze, altrimenti 1.

Queste regole sono un esempio di come Prolog può esprimere logica decisionale che va oltre la semplice classificazione ontologica, preparando il terreno per la risoluzione del CSP.

5. Risoluzione di Problemi con Vincoli (CSP) per l'Assegnamento Risorse

L'ultima fase del progetto utilizza il ragionamento logico di Prolog per risolvere un Problema di Soddisfacimento di Vincoli (CSP), specificamente per l'assegnamento di ambulanze agli incidenti.

5.1. Definizione del Problema di Assegnamento Ambulanze

Nel contesto di questo progetto, il CSP mira ad assegnare in modo efficiente un numero limitato di ambulanze disponibili agli incidenti, tenendo conto della loro gravità e del numero di feriti.

- Variabili: Gli incidenti rilevati e le ambulanze disponibili. In questo specifico CSP, le "variabili" sono le assegnazioni stesse: a quale incidente va quale ambulanza.
- Domini: Le 5 ambulanze disponibili (amb_1, amb_2, amb_3, amb_4, amb_5).
- Vincoli (implementati in csp_ambulanze.pl):

Priorità degli Incidenti: Gli incidenti vengono ordinati per priorità (più gravi prima) usando la regola **ordina_incidenti_priorita**. Questo garantisce che le risorse vengano assegnate prima dove più necessarie.

- Numero di Ambulanze per Incidente:
- Se un incidente ha più di 2 feriti, richiede 2 ambulanze.
- Altrimenti, richiede 1 ambulanza.

Disponibilità delle Ambulanze: Le ambulanze assegnate a un incidente non sono più disponibili per altri incidenti (seleziona_ambulanze e il passaggio di NuoveAmbDisp). Questo assicura che un'ambulanza non sia assegnata a più incidenti contemporaneamente.

Capacità Limitata: Il sistema considera un pool fisso di 5 ambulanze. La soluzione deve rispettare questa limitazione.

5.2. Implementazione del CSP con Prolog

Il file csp_ambulanze.pl contiene la logica per risolvere questo CSP:

- lista_ambulanze(L): Un fatto che definisce il pool iniziale di 5 ambulanze.
- priorita(Gravita, Valore): Definisce i valori di priorità per la gravità.
- ordina_incidenti_priorita(Incidenti, Ordinati): Ordina gli incidenti in base alla gravità.
- seleziona_ambulanze(Numero, ListaDisponibili, Selezionate, Resto): Una regola ricorsiva che seleziona un certo Numero di ambulanze da ListaDisponibili, restituendo le Selezionate e il Resto di quelle disponibili.
- assegna_ambulanze(IncidentiOrdinati, AmbulanzeDisponibili, Assegnamenti): La regola ricorsiva principale che gestisce l'assegnamento. Per ogni incidente ordinato, determina quante

ambulanze richiede e le seleziona dal pool disponibile, continuando il processo con le ambulanze rimanenti.

- assegnamento(Incidente, Ambulanze): Il predicato principale interrogabile, che lega un incidente alle ambulanze assegnate, una volta trovata una soluzione.

Lo script Python csp_solve.py utilizza la libreria pyswip per interagire con l'interprete Prolog. Carica il file csp_ambulanze.pl e interroga il predicato assegnamento(Incidente, Ambulanze) per ottenere le soluzioni all'assegnamento.

5.3. Risultati e Discussione dell'Assegnamento

Il modulo CSP, attraverso Prolog, fornisce una soluzione all'assegnamento delle ambulanze. Data la natura non esaustiva dei dati di esempio in csp_ambulanze.pl (molti incidenti "normali" con 0 feriti), l'output mostrerà come le ambulanze vengono assegnate agli incidenti che realmente richiedono attenzione (cioè, quelli con feriti). Ciò deriva dal dataset inziale povero di immagini in cui ci sono feriti per terra accanto le auto, o comunque scene significative che si possono presentare nella realtà.

Di seguito è riportato un esempio di assegnamento:

```
Assegnamenti ambulanze:
Incidente http://example.org/incidente#img_157 -> Ambulanze: ['a1']
```

Questo dimostra come la conoscenza derivata (gravità incidente, numero feriti) e le regole logiche implementate in Prolog permettano al sistema di prendere decisioni operative cruciali, ottimizzando l'uso delle risorse limitate in base a criteri di urgenza e necessità.

6. Conclusioni e Lavoro Futuro

6.1. Valutazione dell'Approccio Ibrido

Questo progetto dimostra efficacemente il potere dell'approccio ibrido all'Intelligenza Artificiale, combinando:

- **Percezione (Deep Learning):** Con YOLOe per l'analisi visiva delle scene di incidente.
- **Rappresentazione e Ragionamento della Conoscenza (Ontologie/KG):** Per strutturare le informazioni e inferire conoscenza semantica di alto livello.
- **Programmazione Logica e Ragionamento con Vincoli (Prolog):** Per risolvere problemi decisionali complessi e ottimizzare l'allocazione delle risorse.

L'integrazione di questi moduli permette di superare i limiti di ciascun approccio preso singolarmente, fornendo un sistema più robusto, interpretabile e capace di ragionamento di alto livello. La conoscenza derivata semanticamente da OWL e raffinata con Prolog diventa la base per decisioni operative.

6.2. Limiti del Progetto Attuale

- **Dipendenza da Etichette Dettagliate:** Le inferenze sulla "PersonaFerita" e la gravità dell'incidente dipendono dalle capacità di YOLOe di distinguere tra AutoIncidentata e Auto normale, e da una metrica di vicinanza. Un sistema più avanzato richiederebbe un modello di Deep Learning per il riconoscimento dei danni specifici e della gravità, o l'integrazione di sensori medici per lo stato delle persone.
- **Regole di Ragionamento Semplificate:** Le regole OWL e Prolog, pur essendo dimostrative, sono semplificate. Un sistema reale richiederebbe un'ontologia più ricca e un set di regole molto più complesso e finemente granulare, magari basato su normative specifiche.
- **Assenza di Dati in Tempo Reale:** Il sistema opera su un dataset statico. L'integrazione con flussi di dati in tempo reale (telecamere live, GPS ambulanze) sarebbe una sfida successiva.
- **CSP Basic:** Il CSP sull'assegnamento delle ambulanze è basilare e non considera vincoli come la distanza geografica, il tempo di risposta, la disponibilità reale delle ambulanze in un determinato momento, o la tipologia di ferite che richiedono ambulanze specializzate.

6.3. Possibili Sviluppi Futuri

- **Modulo di Riconoscimento Danni:** Sviluppare o integrare un modulo di Deep Learning specifico per la classificazione del tipo e della gravità dei

danni sui veicoli, superando la necessità di annotazione manuale o euristiche semplici.

- **Ontologia Estesa:** Arricchire l'ontologia con maggiori dettagli, includendo classi per tipi di strade, condizioni meteorologiche avanzate, tipi di veicoli specifici, geolocalizzazione, strutture ospedaliere, ecc.
- **Ragionamento Temporale e Spaziale:** Integrare capacità di ragionamento temporale (es. "quanto tempo fa è avvenuto l'incidente", "tempo stimato di arrivo delle risorse") e spaziale più sofisticato (calcolo distanze reali, percorsi).
- **Sistema di Interrogazione Utente:** Sviluppare un'interfaccia utente che permetta agli operatori di interrogare il Knowledge Graph in linguaggio naturale o tramite un'interfaccia grafica per ottenere insight e supporto decisionale.
- **CSP Avanzato:** Estendere il CSP in Prolog per includere vincoli più realistici:
    - Posizione geografica delle ambulanze e degli incidenti.
    - Tempi di risposta stimati.
    - Specializzazioni delle ambulanze (es. ALS vs BLS).
    - Costi associati alle assegnazioni.