UNIVERSITÀ DEGLI STUDI DI BARI ALDO MORO

DIPARTIMENTO DI INFORMATICA

Ontologia di Biblioteca

De Nora Antonella [MAT. 797866] – a.denora13@studenti.uniba.it

Link GitHub: https://github.com/denora-antonella/OntologiaBiblioteca

Documentazione Caso di Studio
Ingegneria della Conoscenza
A.A. 2025-2026

Sommario

Capitolo 0 – Introduzione

0.1 Obiettivi del progetto

L'obiettivo del progetto "Ontologia di Biblioteca" è di realizzare un sistema di ricerca intelligente basato su un'ontologia OWL del dominio "Biblioteca" ed integrando tecniche di rappresentazione della conoscenza con algoritmi di ricerca informata.

Il sistema consente di trasformare una base di conoscenza ontologica in uno spazio di ricerca navigabile, sul quale viene applicato l'algoritmo A* per l'individuazione dei percorsi semantici tra entità eterogenee, ossia persone, libri, categorie tematiche e prestiti.

Il progetto si concentra sull'analisi del comportamento della ricerca in presenza di diverse strategie euristiche. Vengono implementate tre modalità:

- ricerca non informata (A con euristica nulla, Dijkstra),
- euristica base,
- euristica informata basata sulla gerarchia tassonomica delle categorie.

L'obiettivo non è solo di trovare un percorso tra due nodi, ma studiare come la struttura semantica dell'ontologia possa influenzare l'efficienza della ricerca, in termini di numero di nodi espansi e tempo di esecuzione.

Il sistema è progettato in modo modulare per permettere l'aggiunta di nuove entità o relazioni all'interno dell'ontologia senza richiedere modifiche all'algoritmo di ricerca. L'ontologia diventa quindi uno spazio degli stati dinamico e riutilizzabile.

0.2 Ambiente di sviluppo e linguaggio

Per questo progetto è stato utilizzato Visual Studio Code come ambiente di programmazione. Precisamente è stato sviluppato in Python 3 per la chiarezza sintattica, la facilità di modularizzazione e la disponibilità di librerie dedicate alla gestione di ontologie e all'analisi sperimentale.

La struttura del progetto è organizzata in moduli distinti (integrazione della knowledge base, modulo di ricerca, valutazione sperimentale) per garantire separazione delle responsabilità, leggibilità del codice e facilità di manutenzione.

0.3 Librerie e Moduli utilizzati

Il progetto è stato sviluppato utilizzando librerie Python sia standard sia di terze parti, oltre a moduli implementati per l'integrazione tra ontologia e algoritmo di ricerca.

Tra queste abbiamo:

owlready2
utilizzata per il caricamento dell'ontologia OWL (biblioteca.owl) e per l'accesso programmatico a individui, object property, e relazioni tra entità. Questa libreria consente di trasformare la rappresentazione ontologica in una struttura navigabile dal sistema di ricerca.

pandas
impiegata nella fase di valutazione sperimentale per aggregare i risultati delle esecuzioni multiple, calcolare media e deviazione standard e generare tabelle riassuntive salvate in formato CSV.

matplotlib
utilizzata per la generazione dei grafici comparativi relativi a tempo medio di esecuzione, numero medio di nodi espansi, confronto tra euristiche.

heapq
utilizzata per implementare la coda di priorità dell'algoritmo A*, necessaria per l'ordinamento dei nodi.

time
utilizzata per misurare con precisione i tempi di esecuzione delle ricerche durante gli esperimenti.

pathlib
impiegata per la gestione dei percorsi dei file in modo portabile tra sistemi operativi.

collections.deque
utilizzata per implementare strutture FIFO nella BFS di supporto utilizzata dall'euristica informata e nelle funzioni di analisi dei nodi raggiungibili.

csv
modulo standard utilizzato per salvare i risultati sperimentali in formato CSV, per poterli analizzare e riutilizzare in modo strutturato.

json
modulo standard impiegato per salvare i risultati anche in formato JSON, utile per eventuali elaborazioni successive o integrazioni future.

0.4 Moduli e Classi Implementate

Il progetto è strutturato in moduli separati, ciascuno con una responsabilità specifica.

Modulo: integrazione_kb

Contiene tutto ciò che serve per l'interazione con la knowledge base.

- **costruisci_grafo.py**

Funzione principale:

- costruisci_grafo(ontologia)

per estrarre individui e relazioni dall'ontologia, per costruire un grafo orientato pesato e per definire i costi associati alle diverse tipologie di relazioni.

- **euristiche_biblioteca.py**

Contiene:

- euristica_informata_tassonomia(...)

per stimare la distanza tra uno stato corrente e una categoria obiettivo sfruttando la gerarchia sottoCategoriaDi per migliorare l'efficienza della ricerca.

Modulo: ricerca_percorsi

Contiene l'implementazione dell'algoritmo di ricerca.

- **problema_biblioteca.py**

Classe:

- ProblemaBiblioteca

per definire lo stato iniziale, per definire l'insieme degli stati obiettivo e per fornire le funzioni di espansione dei successori.

- **algoritmo_a_stella.py**

Funzione principale:

- a_stella(problema, euristica)

per implementare l'algoritmo A*, per gestire la frontiera tramite coda di priorità e per restituire il percorso trovato, il costo totale e il numero di nodi espansi.

Modulo: valutazione_sperimentale

Responsabile dell'analisi quantitativa delle prestazioni. Include script per l'esecuzione multipla automatizzata dei test, raccolta dati in CSV e JSON, calcolo di statistiche aggregate e generazione (media e deviazione standard) e generazione grafici finali.

Capitolo 1 – Acquisizione e Preparazione della KB

1.1 Acquisizione dell'ontologia e selezione delle relazioni rilevanti

La base informativa di questo progetto è costituita dall'ontologia biblioteca.owl, che rappresenta il dominio di una biblioteca. L'ontologia contiene un insieme eterogeneo di individui e relazioni, tra cui persone, libri, categorie tematiche e prestiti, collegati tramite proprietà oggettuali per le varie interazioni. Essa è stata costruita in modo coerente con gli obiettivi del sistema, ovvero consentire l'analisi e l'esplorazione di percorsi semantici tra entità diverse del dominio. La modellazione segue i principi classici della rappresentazione della conoscenza in ambito ontologico (3).

La fase di acquisizione è gestita nel file main.py, tramite la funzione carica_ontologia(), che utilizza la libreria Owlready2 (4) per leggere il file OWL e caricarlo in memoria come oggetto navigabile.

Essendo molte le informazioni presenti nell'ontologia, è stata necessaria una fase di selezione delle relazioni rilevanti ai fini della ricerca. Le object property considerate fondamentali nel progetto sono:

- haPrestito: collega una persona a un prestito effettuato;
- riguardaLibro: collega un prestito al libro corrispondente;
- appartieneCategoria: collega un libro a una categoria tematica;
- sottoCategoriaDi: definisce la gerarchia tassonomica tra categorie.

In questa fase (2) non viene modificata la conoscenza presente nell'ontologia, ma si isolano esclusivamente le relazioni utili alle successive fasi di elaborazione e ricerca così da non modificare la conoscenza ma individuare gli elementi strutturali che costituiranno la base del grafo su cui verrà applicato l'algoritmo A*.

1.2 Preparazione dello spazio di ricerca e costruzione del grafo

Dopo il caricamento dell'ontologia, è stato necessario trasformare la rappresentazione semantica in una struttura dati adatta agli algoritmi di ricerca su grafi nel file integrazione_kb/costruisci_grafo.py utilizzando la funzione costruisci_grafo(). Il processo consiste nello scorrere tutti gli individui presenti nell'ontologia e nel trasformare ogni relazione selezionata in un arco di un grafo orientato pesato. In particolare:

- ogni individuo dell'ontologia viene convertito in un nodo;
- ogni object property rilevante viene trasformata in un arco tra nodi;
- ad ogni arco viene associato un costo numerico (nella maggior parte dei casi uguale a 1), con un peso ridotto per la relazione sottoCategoriaDi, al fine di favorire l'esplorazione della gerarchia tassonomica.

Questa trasformazione costituisce l'equivalente del preprocessing nei sistemi basati su dataset (2), ma in questo caso non si tratta di normalizzare valori numerici ma di convertire una rappresentazione logico-semantica in una struttura computazionale navigabile.

Il grafo risultante mantiene:

- le connessioni tra persone e prestiti,
- i collegamenti tra prestiti e libri,
- l'appartenenza dei libri alle categorie,
- la gerarchia multilivello tra categorie.

Importante è la presenza della relazione sottoCategoriaDi perché introduce una struttura tassonomica profonda, che consente di esplorare percorsi di diversa lunghezza e complessità. Questa impostazione è coerente con il formalismo classico dei problemi di ricerca nello spazio degli stati (1), in cui:

- un nodo rappresenta uno stato,
- un arco rappresenta una transizione,
- un percorso rappresenta una sequenza di azioni o relazioni.

Al termine di questa fase, la Knowledge Base non è soltanto una rappresentazione descrittiva del dominio, ma diventa uno spazio di ricerca formalmente definito, pronto per l'applicazione dell'algoritmo A* e per l'analisi sperimentale delle sue prestazioni.

Capitolo 2 – Costruzione del modello di ricerca

In questo progetto la ricerca euristica viene applicata a uno spazio degli stati derivato da una base di conoscenza ontologica che modella il dominio di una biblioteca. Lo spazio di ricerca non è definito manualmente ma viene estratto automaticamente da una ontologia OWL, che rappresenta entità e relazioni del dominio in forma strutturata.

2.1 Modellazione del dominio tramite ontologia

Il dominio della biblioteca è rappresentato mediante un'ontologia OWL, sviluppata per descrivere in modo formale le entità principali e le relazioni tra esse.

L'ontologia include individui appartenenti alle classi principali Persona, Libro, Categoria e Prestito. Le relazioni tra tali entità sono modellate attraverso object property, tra cui haPrestito, riguardaLibro, appartieneCategoria e sottoCategoriaDi.

La relazione sottoCategoriaDi riveste un ruolo centrale nel progetto, in quanto definisce una gerarchia tassonomica tra categorie. Tale struttura gerarchica costituisce una fonte di informazione semantica che può essere sfruttata nella definizione di euristiche informate (2). Il caricamento dell'ontologia avviene tramite la libreria owlready2 (4), utilizzando la funzione carica_ontologia(), che impiega get_ontology().load() per rendere disponibili classi, individui e proprietà all'interno dell'ambiente Python. Attraverso il metodo individuals() è possibile iterare su tutti gli individui presenti nella knowledge base, consentendo al sistema di analizzare dinamicamente la struttura del dominio senza dover codificare manualmente i nodi.

La base di conoscenza così strutturata rappresenta un modello semantico del dominio, ma non è direttamente utilizzabile come spazio di ricerca. È quindi necessario trasformarla in una rappresentazione a grafo.

2.2 Estrazione del grafo ontologico

La trasformazione dell'ontologia in uno spazio degli stati navigabile viene realizzata nel modulo costruisci_grafo.py, attraverso la funzione costruisci_grafo(). Questa funzione analizza gli individui dell'ontologia e costruisce una struttura dati di tipo dizionario annidato, in cui ogni nodo è associato ai nodi adiacenti. Ogni individuo dell'ontologia diventa un nodo del grafo, mentre ogni object property genera uno o più archi tra entità collegate.

Ad esempio, se una Persona è collegata a un Prestito tramite la proprietà haPrestito, nel grafo verrà creato un arco tra i due nodi corrispondenti. Un altro esempio è la relazione appartieneCategoria che genera collegamenti tra libri e categorie, mentre sottoCategoriaDi costruisce la gerarchia tra categorie.

Il grafo risultante rappresenta dunque una proiezione strutturale della knowledge base, in cui la semantica delle relazioni ontologiche viene tradotta in connessioni navigabili. Il grafo è rappresentato come un dizionario {nodo: {vicino: costo}}, così da poter essere utilizzato direttamente dagli algoritmi di ricerca su grafi.

La scelta progettuale è stata quella di considerare un grafo non pesato, attribuendo implicitamente costo unitario a ciascun arco (ad eccezione di eventuali relazioni tassonomiche che possono essere trattate in modo un po' diverso a livello implementativo), perché consente di concentrare l'analisi sull'impatto delle euristiche, evitando che pesi arbitrari possano influenzare i risultati sperimentali. Questa impostazione è coerente con la formulazione classica dei problemi di ricerca nello spazio degli stati (1). La struttura del grafo costituisce lo spazio degli stati su cui opera l'algoritmo di ricerca A*.

2.3 Implementazione dell'algoritmo A*

L'algoritmo di ricerca è implementato nel modulo algoritmo_a_stella.py tramite la funzione a_stella(). L'algoritmo utilizza la formulazione classica della ricerca informata (1), in cui ogni nodo viene valutato attraverso la funzione: f(n) = g(n) + h(n) (g(n) rappresenta il costo accumulato dal nodo iniziale fino al nodo corrente e h(n) è una stima euristica del costo rimanente verso l'obiettivo). La gestione dell'insieme dei nodi da esplorare avviene tramite una coda di priorità implementata con il modulo heapq (5), che consente di estrarre il nodo con valore f(n) minimo.

La funzione a_stella() riceve in ingresso un oggetto di tipo ProblemaBiblioteca, definito nel modulo problema_biblioteca.py, che incapsula lo stato iniziale, l'insieme degli obiettivi e la funzione di espansione dei nodi.

Durante l'esecuzione, l'algoritmo mantiene traccia del numero di nodi espansi. Questa informazione viene utilizzata in seguito nella fase sperimentale per confrontare l'efficienza delle diverse euristiche. La funzione restituisce percorso, costo totale e numero di nodi espansi. Il percorso viene ricostruito risalendo i predecessori a partire dal nodo obiettivo, mentre il costo corrisponde alla lunghezza del cammino nel grafo. In questo modo, la knowledge base ontologica viene trasformata da una semplice rappresentazione descrittiva del dominio in uno spazio degli stati definito, su cui è possibile applicare e valutare un algoritmo di ricerca informata.

Capitolo 3 – Progettazione delle Euristiche

La qualità della funzione euristica rappresenta l'elemento centrale che determina l'efficienza dell'algoritmo A*. Una funzione euristica efficace consente di guidare l'esplorazione verso regioni promettenti dello spazio degli stati, riducendo il numero di nodi espansi e, di conseguenza, il costo computazionale complessivo (1). In questo progetto sono state implementate tre diverse configurazioni euristiche, con l'obiettivo di analizzare come livelli differenti di informazione influenzino il comportamento della ricerca su un grafo derivato da una ontologia.

3.1 Euristica nulla

L'euristica nulla è definita dalla funzione euristica_nulla() all'interno del modulo principale. Tale funzione restituisce costantemente il valore 0 per qualsiasi stato corrente. Quindi la funzione di valutazione dell'algoritmo A* si riduce a: f(n) = g(n).

L'algoritmo non utilizza alcuna informazione sul nodo obiettivo e si comporta esattamente come una ricerca uniforme, ossia equivalente all'algoritmo di Dijkstra (1).

L'euristica nulla non fornisce una guida aggiuntiva, ma garantisce comunque l'ottimalità del percorso trovato, in quanto A* con h(n) = 0 soddisfa le condizioni di ammissibilità. Consente di misurare il comportamento della ricerca in assenza totale di informazione euristica, costituendo una baseline per il confronto con le altre strategie.

3.2 Euristica base

L'euristica base è implementata tramite la funzione euristica_base(). Questa funzione introduce una distinzione minimale tra stato corrente e obiettivo. Restituisce 0 se il nodo corrente coincide con il nodo obiettivo e 1 in tutti gli altri casi. h(n) = 0 se n = goal, altrimenti h(n) = 1.

Si tratta di una funzione che non utilizza informazioni strutturali del grafo, ma introduce una leggera differenziazione rispetto all'euristica nulla. Anche questa euristica è ammissibile, in quanto non sovrastima mai il costo reale del percorso verso l'obiettivo (1).

La sua capacità di guida però rimane molto limitata, poiché non tiene conto né della distanza effettiva nel grafo né della struttura tassonomica dell'ontologia. Valuta se anche una minima informazione possa produrre un miglioramento rispetto alla ricerca puramente uniforme.

3.3 Euristica informata basata sulla tassonomia

L'euristica informata è implementata nel modulo euristiche_biblioteca.py tramite la funzione euristica_informata_tassonomia(), che utilizza una procedura di visita in ampiezza (BFS) sul grafo per stimare la distanza tra lo stato corrente e una categoria obiettivo.

Quando il nodo obiettivo appartiene alla classe Categoria, l'euristica calcola una stima della distanza semantica tra il nodo corrente e l'obiettivo utilizzando la relazione sottoCategoriaDi. Nel caso in cui lo stato corrente non sia direttamente una categoria (come ad esempio persona, prestito o

libro), l'euristica considera i collegamenti intermedi (persona → prestito → libro → categoria) per ottenere una stima coerente. La funzione analizza il grafo costruito e risale la gerarchia tassonomica per stimare il numero di livelli che separano una categoria dal nodo obiettivo. Questa idea si fonda sull'utilizzo di informazione strutturale del dominio per guidare la ricerca, in linea con i principi della ricerca euristica informata (1).

Questa stima viene utilizzata come valore di h(n), fornendo una misura della profondità semantica che separa il nodo corrente dalla destinazione. Nel caso in cui l'obiettivo non sia una categoria, il sistema effettua un fallback verso l'euristica base, evitando di applicare una stima non significativa. L'euristica è progettata per rimanere conservativa rispetto alla struttura del grafo e, nei casi considerati nel progetto, non sovrastima il costo reale del percorso verso l'obiettivo.

Questa scelta progettuale consente di integrare la conoscenza ontologica direttamente nella funzione di valutazione dell'algoritmo A*, realizzando un collegamento concreto tra rappresentazione della conoscenza e ricerca euristica. A differenza delle precedenti euristiche, l'euristica informata utilizza la semantica del dominio per guidare l'esplorazione, riducendo il branching effettivo nei casi caratterizzati da percorsi profondi all'interno della gerarchia delle categorie.

Capitolo 4 – Analisi sperimentale e valutazione delle prestazioni

L'analisi sperimentale rappresenta la fase in cui il comportamento dell'algoritmo A* viene valutato in modo sistematico per comprendere l'impatto delle diverse configurazioni euristiche sulle prestazioni complessive del sistema.

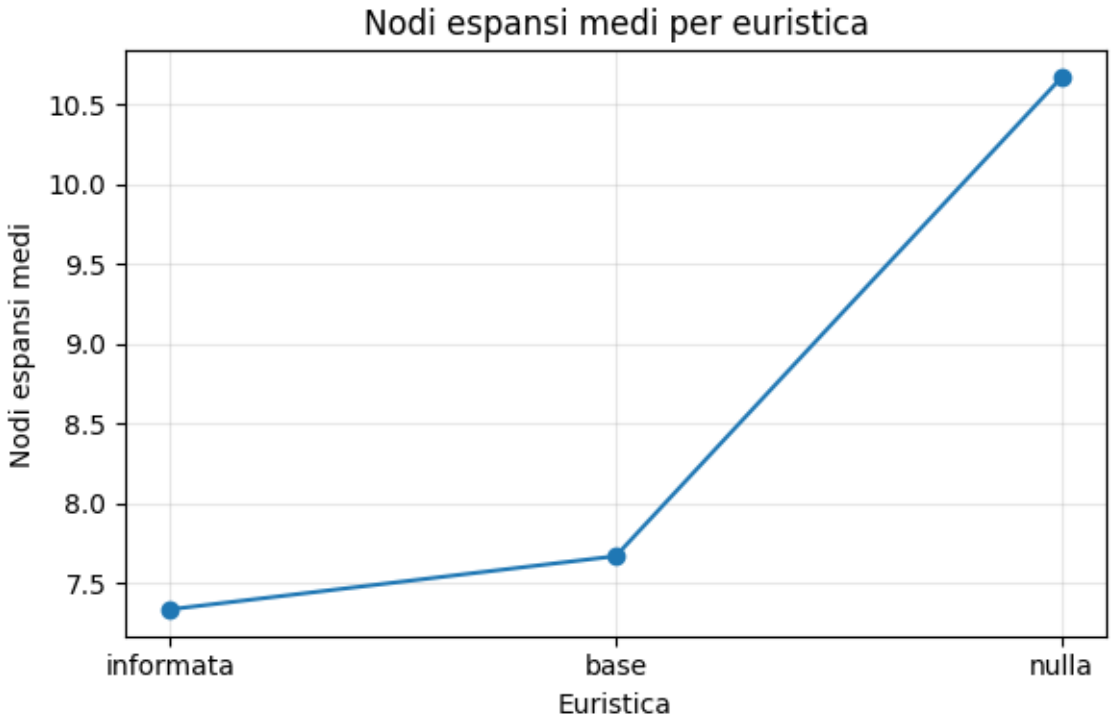
La sperimentazione è stata automatizzata attraverso il modulo runner_esperimenti.py, che esegue una serie di casi di studio predefiniti in cui variano nodo iniziale, nodo obiettivo ed euristica utilizzata. Per ciascun caso vengono effettuate più esecuzioni indipendenti, in modo da ottenere misure stabili e non dipendenti da una singola run.

I risultati grezzi vengono salvati nel file risultati.csv e nel file risultati.json. Successivamente, il modulo genera le statistiche aggregate all'interno del processo di valutazione utilizzando genera_report.py, producendo il file tabella_riassuntiva.csv con media e deviazione standard delle principali metriche (tempo di esecuzione, nodi espansi e costo)..

La visualizzazione grafica è gestita dal file visualizza_grafici.py, che utilizza la libreria pandas per leggere i dati aggregati e matplotlib per produrre automaticamente i grafici salvati nella cartella valutazione_sperimentale/risultati/.

4.1 Confronto tra le euristiche

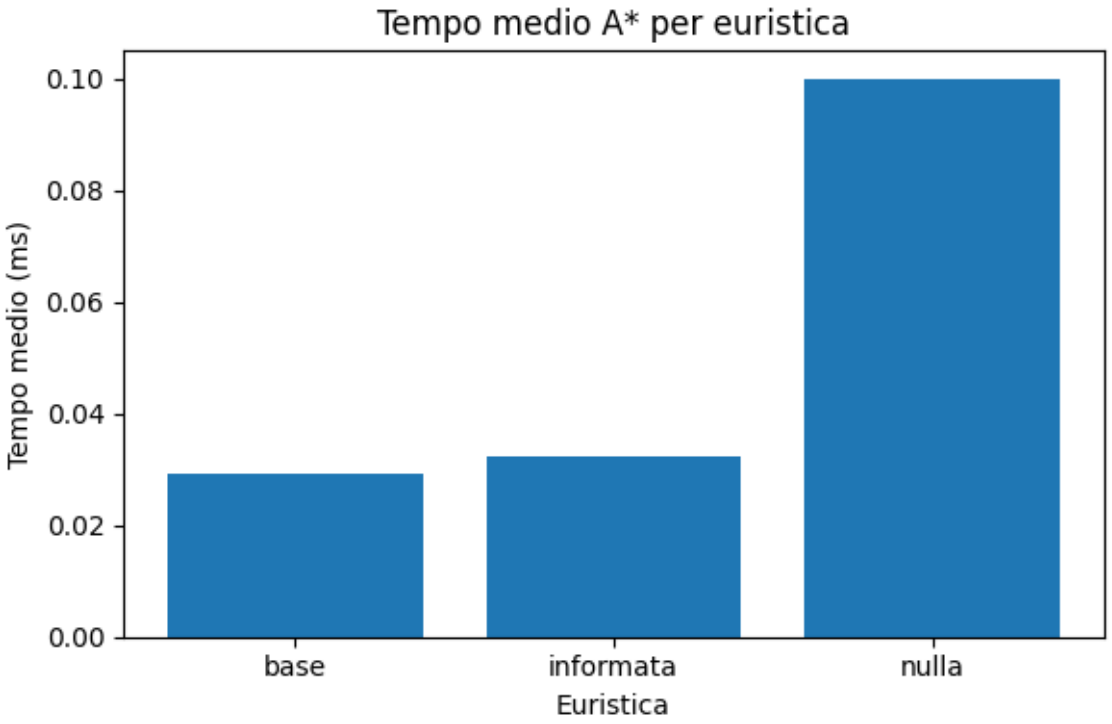
Il primo livello di analisi consiste nel confronto globale tra le tre euristiche implementate: nulla, base e informata. Nel file visualizza_grafici.py, i valori della colonna nodi_espansi vengono raggruppati per euristica e viene calcolata la media direttamente tramite pandas. Il grafico viene generato tramite matplotlib e salvato automaticamente con plt.savefig() nel file:



Dal grafico si nota che l'euristica nulla comporta mediamente il numero più elevato di nodi espansi. Questo comportamento è coerente con la teoria della ricerca informata (1): in assenza di una funzione guida, l'algoritmo esplora una porzione più ampia dello spazio degli stati.

L'euristica base introduce un miglioramento moderato, mentre l'euristica informata risulta sistematicamente la più efficiente nei casi più complessi, riducendo il numero medio di espansioni grazie all'utilizzo della struttura tassonomica dell'ontologia.

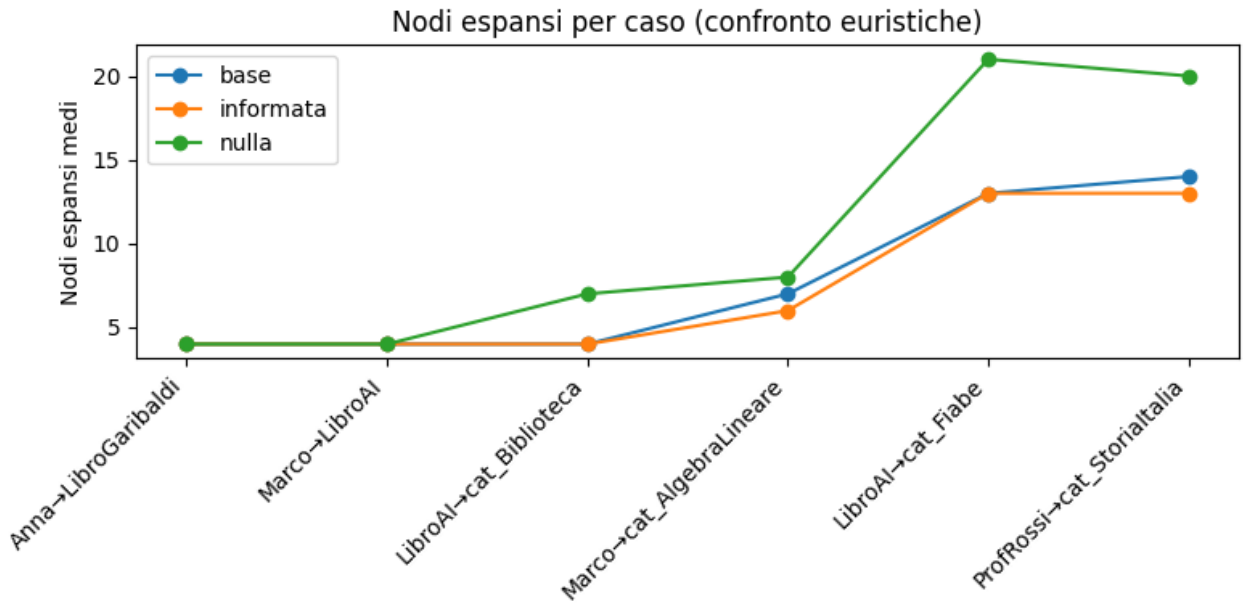
Un'analisi analoga viene effettuata sui tempi medi di esecuzione.



Anche in questo caso l'euristica nulla presenta valori medi superiori. Le differenze temporali risultano meno marcate rispetto ai nodi espansi, in quanto il grafo ha dimensioni contenute; tuttavia, la tendenza generale conferma che una migliore informazione euristica contribuisce a ridurre il lavoro computazionale complessivo.

4.2 Analisi per singolo caso di studio

Per comprendere più a fondo il comportamento dell'algoritmo, è stata condotta un'analisi disaggregata per ciascun caso di studio. Nel file visualizza_grafici.py, i dati vengono raggruppati per combinazione di caso ed euristica. Per ogni coppia vengono estratti i valori medi dei nodi espansi, che vengono rappresentati in un grafico comparativo salvato nel file:

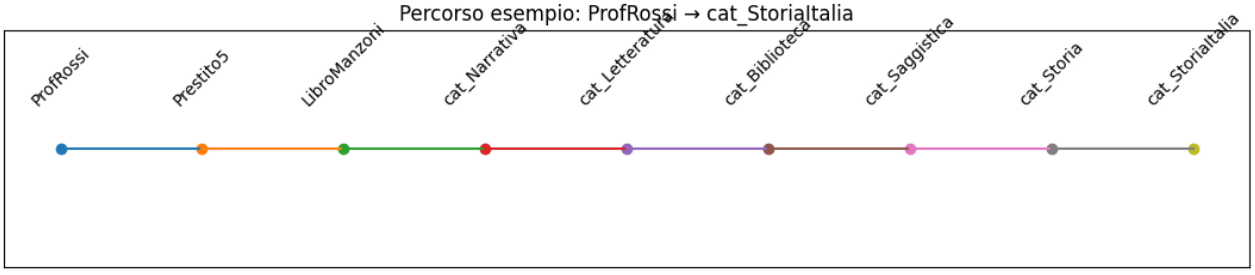


L'analisi evidenzia che nei casi semplici con percorsi brevi tra individuo e libro, le differenze tra le euristiche risultano limitate. Questo avviene perché lo spazio degli stati esplorato è ridotto e l'algoritmo raggiunge rapidamente l'obiettivo anche in assenza di una guida informativa.

Nei casi più complessi invece, in particolare quelli che attraversano più livelli della gerarchia delle categorie, l'euristica informata mostra invece un vantaggio significativo. In tali situazioni, la funzione euristica_informata_tassonomia() riesce a stimare in modo più accurato la distanza semantica dal nodo obiettivo, indirizzando l'esplorazione verso rami più promettenti del grafo. Questo comportamento è coerente con l'idea che una buona euristica riduca il branching effettivo dello spazio degli stati, limitando l'espansione di nodi non rilevanti (1).

4.3 Esempio di percorso trovato

Il sistema interattivo implementato nel file principale consente di selezionare nodo iniziale, nodo obiettivo ed euristica. Una volta eseguita la ricerca, il percorso viene stampato passo per passo e in forma compatta.



Mostra il percorso che attraversa relazioni di tipo prestito, libro e categoria, mostrando come il sistema sfrutti le proprietà ontologiche per costruire una sequenza coerente di passaggi semantici.

Il percorso, dovuto all'integrazione tra rappresentazione della conoscenza e ricerca, non è soltanto una sequenza di nodi ma riflette relazioni significative definite nell'ontologia.

Capitolo 5 – Conclusioni e Sviluppi futuri

Questo progetto mostra come sia possibile integrare in modo coerente tecniche di rappresentazione della conoscenza e algoritmi di ricerca euristica all'interno di un unico sistema intelligente.

A partire da un'ontologia OWL che modella il dominio di una biblioteca, è stato costruito un grafo navigabile sul quale è stato applicato l'algoritmo A*, opportunamente adattato al contesto. L'obiettivo non è soltanto individuare percorsi tra entità eterogenee, ma analizzare in modo quantitativo come la scelta dell'euristica influenzi il comportamento della ricerca.

L'architettura modulare del sistema favorisce la separazione delle fasi di caricamento dell'ontologia, costruzione del grafo, definizione del problema di ricerca, implementazione delle euristiche e valutazione sperimentale. Questa suddivisione ha reso il progetto più leggibile, estendibile e coerente dal punto di vista progettuale.

Dal punto di vista sperimentale, i risultati confermano quanto previsto dalla teoria della ricerca informata (1) ossia che una funzione euristica più informativa consente di ridurre il numero di nodi espansi e quindi anche il lavoro computazionale complessivo. Soprattutto l'euristica informata basata sulla struttura tassonomica dell'ontologia si è dimostrata efficace nei casi caratterizzati da maggiore profondità gerarchica, guidando l'algoritmo verso rami più promettenti del grafo.

Nonostante le dimensioni contenute del dominio, il comportamento osservato evidenzia la differenza tra ricerca non informata e ricerca guidata dalla conoscenza strutturata. Questo risultato rafforza l'idea che la qualità della rappresentazione della conoscenza ha un impatto diretto sull'efficienza dei meccanismi di inferenza e ricerca.

Un altro aspetto rilevante del progetto è l'automatizzazione dell'analisi sperimentale. L'utilizzo di moduli dedicati alla raccolta, aggregazione e visualizzazione dei risultati ha permesso di trasformare dati grezzi in informazioni interpretabili, così da rendere il confronto tra le diverse configurazioni euristiche oggettivo e riproducibile.

Dal punto di vista degli sviluppi futuri, il sistema potrebbe essere esteso in diverse direzioni. Una possibile evoluzione riguarda l'introduzione di pesi differenziati sulle relazioni del grafo, in modo da modellare meglio l'importanza semantica dei collegamenti tra entità. Un'altra estensione potrebbe consistere nell'integrazione del reasoner di Owlready2 per sfruttare inferenze automatiche direttamente sull'ontologia, combinando la ricerca su grafo con meccanismi di ragionamento logico più avanzati.

Infine, naturalmente il dominio potrebbe essere adattato a dimensioni meno contenute per analizzare l'impatto dell'euristica in contesti con spazio degli stati più ampio e maggiore branching factor.

Per concludere, questo progetto dimostra come la combinazione tra ontologie e algoritmi di ricerca informata costituisca un approccio efficace per la costruzione di sistemi intelligenti capaci di navigare strutture semantiche complesse. L'integrazione tra rappresentazione della conoscenza e ricerca euristica non rappresenta solo un esercizio teorico, ma uno strumento pratico per migliorare l'efficienza e l'interpretabilità dei sistemi basati su grafi semantici.

Bibliografia

(1) Russell, S., Norvig, P. (2021). *Artificial Intelligence: A Modern Approach* (4th ed.). Pearson.

(2) Poole, D., Mackworth, A. (2023). *Artificial Intelligence: Foundations of Computational Agents*. Cambridge University Press.

(3) Baader, F., Horrocks, I., Sattler, U. (2008). *The Description Logic Handbook*. Cambridge University Press.

(4) Owlready2 Documentation https://owlready2.readthedocs.io.

(5) Python Documentation – heapq https://docs.python.org/3/library/heapq.html

(6) Pandas Documentation https://pandas.pydata.org/docs/

(7) Matplotlib Documentation https://matplotlib.org/stable/contents.html

Ulteriori fonte usate per la realizzazione del codice:

(8) A* Pathfinding Algorithm – Computerphile https://www.youtube.com/watch?v=ySN5Wnu88nE

(9) A* Algorithm in Python – Tech With Tim https://www.youtube.com/watch?v=JtiK0DOeI4A

(10) Graph Theory Basics – William Fiset https://www.youtube.com/watch?v=09_LlHjoEiY

(11) Matplotlib Tutorial – Corey Schafer https://www.youtube.com/watch?v=UO98lJQ3QGI

(12) GeeksforGeeks – A* Search Algorithm https://www.geeksforgeeks.org/a-search-algorithm/