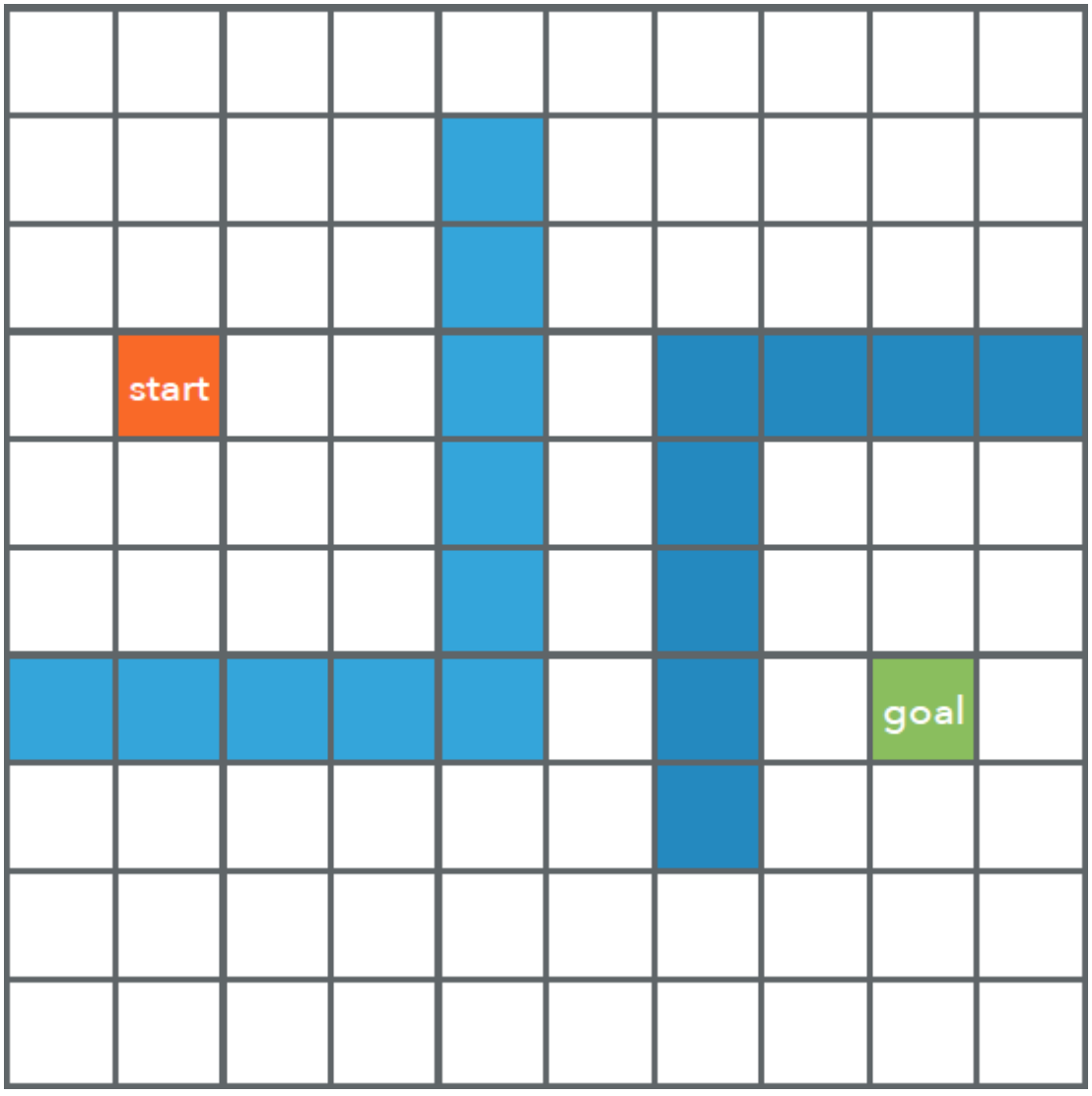
RISOLUZIONE LABIRINTO

Ricerca e Ragionamento in Prolog



Studente:

Antonio Di Trani;
Matricola: 776806;
a.ditrani9@studenti.uniba.it

Anno Accademico
2024-25

Repository GitHub

https://github.com/ditraniantonio/Risolutore-Labirinto-Prolog

1. Dominio di interesse

Questo progetto nasce con l'obiettivo di realizzare **un sistema basato su conoscenza** per la risoluzione automatica di labirinti di varie dimensioni (le dimensioni dei due labirinti sono: **5x5 e 10x10**) tramite l'implementazione e il confronto di diversi algoritmi di ricerca nello spazio degli stati.
L'ambiente di riferimento è rappresentato da una griglia (labirinto) descritta in Prolog, in cui ogni cella corrisponde a una posizione potenzialmente raggiungibile dall'agente. I movimenti consentiti sono modellati come transizioni tra nodi del grafo, mentre ostacoli e percorsi vengono gestiti tramite fatti e regole nella base di conoscenza.

Il sistema integra varie **strategie di ricerca**—tra cui **Depth-First Search** (DFS), **Breadth-First Search** (BFS), **Iterative Deepening** e **A***—per esplorare le possibili sequenze di azioni che consentono di raggiungere l'uscita del labirinto. Ogni strategia permette di analizzare in che modo differenti approcci influiscono sull'efficienza, in particolare sul numero di stati visitati prima di accedere alla soluzione del labirinto giungendo allo stato finale.

Dal punto di vista della rappresentazione della conoscenza, lo stato dell'ambiente e le regole di movimento sono espressi tramite clausole e predicati Prolog, facilitando così sia il ragionamento automatico sia l'estensione a nuovi scenari. Il progetto fornisce inoltre strumenti per definire facilmente nuovi labirinti e personalizzare le regole di validità delle mosse o le funzioni euristiche da utilizzare con gli algoritmi informati. Il labirinto viene visualizzato tramite terminale indicando con il simbolo (**#**) le celle bloccate, con il simbolo (*****) il percorso che porta alla soluzione, con il simbolo (**S**) lo stato iniziale e con il simbolo (**E**) quello finale.

Questo lavoro affronta alcuni dei principali temi trattati nel corso di Ingegneria della Conoscenza:

- **Rappresentazione proposizionale e relazionale**: il mondo è descritto mediante fatti e regole logiche, sfruttando il paradigma dichiarativo per modellare dinamiche e vincoli dell'ambiente.
- **Ricerca di soluzioni in spazi degli stati**: il problema viene formalizzato come un grafo, le cui soluzioni vengono esplorate sia tramite algoritmi informati sia non informati.
- **Ragionamento per risoluzione di problemi**: il motore Prolog viene utilizzato per dedurre la validità delle azioni, valutare percorsi possibili e identificare la soluzione migliore in base ai criteri specificati.

2. Rappresentazione proposizionale e relazionale:

In questo progetto, la rappresentazione della conoscenza è fondamentale per modellare in modo dichiarativo gli ambienti di labirinto e le possibili azioni dell'agente risolutore. L'intero sistema è stato progettato per esprimere formalmente la struttura dei labirinti e le regole di navigazione, sfruttando i costrutti logici di Prolog per garantire una manipolazione flessibile e interrogabile degli stati.

La mappa viene concepita come una griglia rettangolare di dimensione variabile (ad esempio 5x5 o 10x10), in cui ogni cella rappresenta uno stato del mondo. I collegamenti tra celle adiacenti sono modellati come archi di un grafo, che determinano le transizioni consentite dall'agente. Gli ostacoli e i vincoli di movimento vengono rappresentati attraverso predicati logici, permettendo così al sistema di distinguere tra celle accessibili e celle bloccate. Le regole logiche specificano quali mosse sono permesse e consentono di verificare, data una posizione, quali azioni siano valide e con quale costo (funzione euristica).

Le informazioni rilevanti per la risoluzione del labirinto vengono organizzate come fatti e regole, rendendo agevole sia la definizione di nuovi ambienti sia l'integrazione con differenti strategie di ricerca.

Fatti e regole

Nel nostro sistema, la base di conoscenza descrive in modo esplicito la struttura del labirinto, le posizioni di partenza e di arrivo, e le eventuali celle ostacolate. I principali fatti utilizzati sono:

- **row_num(X):** definisce il numero delle righe della griglia.
- **col_num(X):** definisce il numero delle colonne della griglia.
- **initialPosition(pos(X,Y)):** posizione di partenza dell'agente.
- **finalPosition(pos(X, Y)):** posizione dell'uscita.
- **Occupied(pos(X,Y))** identifica le celle che rappresentano ostacoli (muri) e che non possono essere attraversate.
- **allowed/2:** relazioni tra celle adiacenti per specificare i movimenti consentiti.

Questi fatti sono definiti nei file dedicati a ciascun labirinto (5x5.pl, 10x10.pl) e nel resto dei file della cartella labyrinth, e possono essere modificati o estesi con semplicità per creare nuovi scenari di prova.

Esempio di fatti della base di conoscenza (labirinto 5x5):

row_num(5).
col_num(5).
initialPosition(pos(2,2)).
finalPosition(pos(5,4)).
occupied(pos(1,1)).
occupied(pos(1,2)).
occupied(pos(2,4)).
occupied(pos(3,2)).
occupied(pos(3,3)).

Questa modellazione garantisce una chiara separazione tra la descrizione statica dell'ambiente e la logica dinamica delle strategie di ricerca, rendendo la base di conoscenza facilmente interrogabile e manipolabile.

Le **regole logiche** sono utilizzate per dedurre nuove informazioni sulla validità delle mosse e per determinare le possibili transizioni tra stati. In particolare, le regole principali gestiscono:

- **Adiacenza tra celle:** definiscono i movimenti legittimi tra celle ortogonalmente adiacenti, nel rispetto dei limiti della griglia.
- **Validità delle mosse:** verifica che la cella di destinazione non sia bloccata.

Esempio di regole:
*(**Adiacenza tra celle**)*
allowed(nord, pos(R,C)) :-

R>1,

R1 is R-1,

\+occupied(pos(R1,C)).

(Validità delle mosse)
move(est, pos(R,C), pos(R, CAdjacent)) :-

CAdjacent is C+1.

Le strategie di ricerca (BFS, DFS, A*, ecc.) sfruttano queste regole per generare e valutare i percorsi possibili in modo automatico, applicando ciascuna la propria logica di esplorazione.

3. Ricerca di soluzioni in spazi degli stati

La fase di ricerca è il punto principale del sistema, poiché consente di determinare un percorso valido dall'ingresso all'uscita di labirinti di diversa dimensione e complessità. L'obiettivo della ricerca è individuare una sequenza di mosse che consenta all'agente di giungere nella posizione finale. Le celle bloccate vengono escluse a priori dalle azioni ammissibili.
l problema è modellato tramite i seguenti elementi:

- **Stato iniziale:** la posizione di partenza dell'agente (es. (2, 2)).
- **Stato finale:** la cella di arrivo (es. (5, 4) in un labirinto 5x5).
- **Spostamenti:** spostamento in una delle celle adiacenti che non siano bloccate.
- **Funzione move/3:** restituisce la nuova posizione dopo il movimento.
- **Funzione euristica:** per gli algoritmi informati come A*, si utilizza la distanza di Manhattan tra la posizione corrente e l'obiettivo.

Il sistema implementa diverse strategie di ricerca nello spazio degli stati, ciascuna con specifiche caratteristiche:

- **Depth-First Search (DFS):** Algoritmo non informato che esplora in profondità un percorso fino a raggiungere un vicolo cieco prima di tornare indietro. Può trovare soluzioni non ottimali e tende a visitare molti stati superflui.
- **Breadth-First Search (BFS):** Algoritmo non informato che esplora il grafo per livelli, garantendo la soluzione più breve possibile in termini di numero di mosse, ma con elevato utilizzo di memoria.
- **A*:** Algoritmo informato che combina il costo del percorso già effettuato e una funzione euristica (distanza di Manhattan). Permette di trovare soluzioni ottimali riducendo il numero di stati esplorati rispetto agli algoritmi non informati.
- **Iterative Deepening:** Variante che unisce i vantaggi di DFS e BFS, consentendo una ricerca in profondità con limiti crescenti, per garantire il ritrovamento della soluzione ottima senza esplorare eccessivamente lo spazio degli stati.

Ogni algoritmo è definito in un modulo Prolog separato (ad esempio, dfs.pl, bfs.pl, astar.pl, iterative_deepening.pl). Tutte le strategie si basano sui predicati di rappresentazione degli stati (allowed/2, move/3, ecc.) e utilizzano regole condivise nei file presenti nella cartella labyrinth (ad esempio: actions.pl, heuristic.pl) per determinare la validità delle mosse e per esplorare lo spazio degli stati. Le chiamate agli algoritmi richiedono la specifica del labirinto da risolvere attraverso il **modulo loader.pl** in cui viene definita la configurazione del labirinto da utilizzare tra **5x5.pl e 10x10.pl**, successivamente gli algoritmi caricano il seguente modulo per inizializzare il labirinto; i percorsi trovati sono rappresentati tramite liste di posizioni(pos(X,Y) ecc) e liste di azioni(nord, sud, ovest, est ..) oltre che dalla grafica del labirinto e del percorso risolutivo tramite il simbolo (*). Per ciascun algoritmo viene tenuta traccia del numero totale di stati visitati durante la ricerca.

Valutazione

La valutazione delle prestazioni degli algoritmi implementati è stata effettuata considerando **il numero di nodi visitati** durante la ricerca della soluzione in due diverse configurazioni di labirinto: una griglia 5x5 e una griglia 10x10. Questa metrica consente di evidenziare le differenze di efficienza tra algoritmi informati e non informati, specialmente al crescere della complessità del

problema.
I risultati ottenuti sono riportati nella seguente tabella:

Algoritmo	**Nodi visitati (5x5)**	**Nodi visitati (10x10)**
BFS	39	2053
DFS	19	109
Iterative Deepening	120	39,153,240
A*	19	73

Nota: Nel caso del labirinto 10x10, l'algoritmo di Iterative Deepening risulta particolarmente inefficiente, con oltre 39 milioni di nodi visitati e un tempo di esecuzione di circa un minuto, a dimostrazione dei suoi limiti pratici in spazi di grandi dimensioni.
L'analisi dei risultati evidenzia come le prestazioni dei vari algoritmi divergano significativamente in funzione della dimensione del labirinto:

- **A*** si conferma l'algoritmo più efficiente, visitando il minor numero di nodi sia nel labirinto 5x5 (19) che in quello 10x10 (73). Questo risultato è dovuto all'uso della funzione euristica, che guida la ricerca in modo mirato verso la soluzione.
- **DFS** ottiene anch'esso risultati ottimi nel caso 5x5 (19 nodi visitati), ma tende a essere meno prevedibile al crescere delle dimensioni, pur mantenendo buone prestazioni nel test 10x10 (109 nodi).
- **BFS** garantisce la soluzione più breve possibile, ma il numero di nodi visitati cresce rapidamente: da 39 nel 5x5 fino a oltre 2000 nel 10x10, mostrando un'elevata sensibilità all'aumento della dimensione della griglia.
- **Iterative Deepening**, pur offrendo un compromesso teorico tra BFS e DFS, si dimostra inefficiente per labirinti di grandi dimensioni, con

un’esplosione combinatoria del numero di nodi visitati (oltre 39 milioni nel 10x10), risultando di fatto impraticabile.

Questi risultati confermano come la scelta dell’algoritmo debba essere guidata dalle caratteristiche specifiche del problema da risolvere. Gli algoritmi informati come A* sono fortemente raccomandati per scenari di maggiore complessità, mentre algoritmi non informati possono risultare adatti solo in casi semplici o fortemente strutturati.

4. Ragionamento per risoluzione di problemi

Il cuore del sistema è costituito dal motore di inferenza logica fornito da Prolog, che abilita tutte le funzionalità di ragionamento automatico necessarie per affrontare e risolvere i labirinti proposti. Grazie al paradigma dichiarativo e all’uso di predicati dinamici, il progetto riesce a adattarsi con flessibilità a diversi ambienti, valutare la validità delle azioni consentite, e individuare percorsi di soluzione attraverso una catena di deduzioni logiche.

La risoluzione del labirinto è affidata a predicati logici che implementano le diverse strategie di ricerca. Ciascun algoritmo (DFS, BFS, Iterative Deepening, A*) sfrutta la base di conoscenza per esplorare lo spazio degli stati e produrre una sequenza di azioni che collega la posizione iniziale all’uscita.

L’output di ciascun predicato risolutivo consiste (come già detto) in una lista ordinata di posizioni e azioni insieme al labirinto, che rappresenta il percorso calcolato in modo completamente automatico sulla base delle regole dichiarate nel sistema.

Conclusioni

Attraverso la modellazione del problema dei labirinti, il progetto ha permesso di esplorare non solo le tecniche classiche di ricerca di percorsi, ma anche le potenzialità del ragionamento logico automatico in un ambiente strutturato. L’agente che naviga nel labirinto non si limita a muoversi alla cieca, ma si avvale di regole dichiarative per valutare la validità delle proprie azioni, confrontare strategie alternative e adattarsi alla configurazione dell’ambiente.

Il lavoro svolto rappresenta una base solida per futuri approfondimenti nell’ambito dei sistemi intelligenti applicati a spazi degli stati complessi.

L’architettura modulare, unita all’uso di predicati dinamici, rende il sistema facilmente estendibile e personalizzabile. Possibili linee di sviluppo includono:

- L’integrazione di nuovi algoritmi di ricerca avanzata (esempio ricerca bidirezionale ecc.);
- L’introduzione di elementi dinamici, come ostacoli mobili, eventi a tempo o modifiche impreviste della mappa;
- L’ampliamento della rappresentazione dell’ambiente, con nuove entità, regole di interazione o bonus che modificano i costi;
- La sperimentazione su labirinti di dimensioni maggiori, per valutare la reale scalabilità delle soluzioni adottate;
- L’individuazione di altri parametri che permetto di valutare l’efficienza degli algoritmi implementati (come, ad esempio, il tempo di esecuzione per ogni algoritmo);
- L’introduzione di altre funzioni di costo personalizzate;
- L’inclusione di un’interfaccia grafica per la visualizzazione dei percorsi e l’interazione con l’utente.

Alcuni aspetti non sono stati approfonditi per ragioni di tempo o di complessità. Il sistema dimostra come sia possibile costruire, a partire da una base dichiarativa in Prolog, uno strumento flessibile e potente per l’analisi e la soluzione automatica di problemi complessi. l lavoro svolto dimostra l’efficacia delle tecniche di Intelligenza Artificiale classica e la potenza espressiva del paradigma logico-dichiarativo, fornendo strumenti concreti e flessibili sia per la didattica che per la ricerca futura.