RELAZIONE PROGETTO DI INGEGNERIA DELLA CONOSCENZA

Anno Accademico 2021 - 2022

AUTOBIBLO PROJECT : Gestione di una biblioteca



Studenti:

Mario Franco, matricola 717768, e-mail: m.franco34@studenti.uniba.it

Antonio Curione, matricola 716131 , e-mail: a.curione7@studenti.uniba.it

Rosanna Acquafredda, matricola 724972, e-mail: r.acquafredda8@studenti.uniba.it

Indice

1. [Introduzione](#introduzione)
2. [Ontologia](#ontologia)
3. [CSP](#csp)
4. [Ricerca sul grafo](#ricerca)
5. [Recommender system con KNN](#knn)
6. [Valutazione dei risultati](#risultati)
7. [Note finali](#note)

INTRODUZIONE

La seguente documentazione rappresenta la descrizione tecnica del progetto “**AUTOBIBLO project**”, realizzato dagli studenti Mario Franco, Antonio Curione e Rosanna Acquafredda, per il corso di Ingegneria della Conoscenza (A.A. 2021-2022).

*Panoramica del progetto*

Il progetto realizzato rappresenta un’automazione della gestione di una biblioteca scolastica, con l’aggiunta di un sistema di raccomandazione dei libri, con lo scopo di incentivare la lettura , offrendo un servizio semplice ed efficace.

*Obiettivi*

* Progettare un sistema sotto forma di grafo, che specifichi l’organizzazione della biblioteca, così da poterlo utilizzare nelle fasi successive
* Creare un’ontologia adeguata, che possa rappresentare al meglio il sistema gestionale della biblioteca.
  + Di seguito, popolare tale ontologia, inserendo le istanze entità-individui, così da poter effettuare delle simulazioni
  + dimostrare le funzionalità attraverso delle query, che ci permettono di visualizzare l’ontologia
* Modellare, utilizzando CSP (Constraint Satisfaction Problem), un sistema per gestire le prenotazioni dei libri presenti nella biblioteca e le relative restituzioni
* Definire ed implementare un algoritmo di ricerca sul grafo pesato, realizzato al primo punto, così da poter esplicitare il percorso da seguire per trovare, più agilmente, un determinato libro nell’ambiente dalla biblioteca
* Appropriarsi di un dataset di libri adeguato e strutturare un recommender system, utilizzando il KNN, per poter consigliare agli studenti dei libri, basandoci sulle similarità tra il libro di interesse e quelli presenti nel dataset
* Fare un resoconto del sistema realizzato, effettuando vari test e confrontando i risultati ottenuti con quelle che erano le aspettative

*Specifiche di realizzazione*

* Per realizzare il progetto abbiamo utilizzato come IDE PyCharm, con il linguaggio Python (versione 3.10 )
* L’ontologia è stata realizzare su Protégé (versione 5.5.0)
* Altri dettagli sulle librerie utilizzate sono presenti nel file requirement.txt

Come prima cosa, abbiamo individuato una planimetria di una biblioteca, che meglio si adattasse alla nostra idea di “libreria scolastica”. Per costruire il nostro spazio di ricerca abbiamo fatto riferimento alla biblioteca della città di Roncade, prendendone la piantina dal sito comunale.

A partire da tale piantina, abbiamo individuato i nodi principali e i percorsi che li collegano.

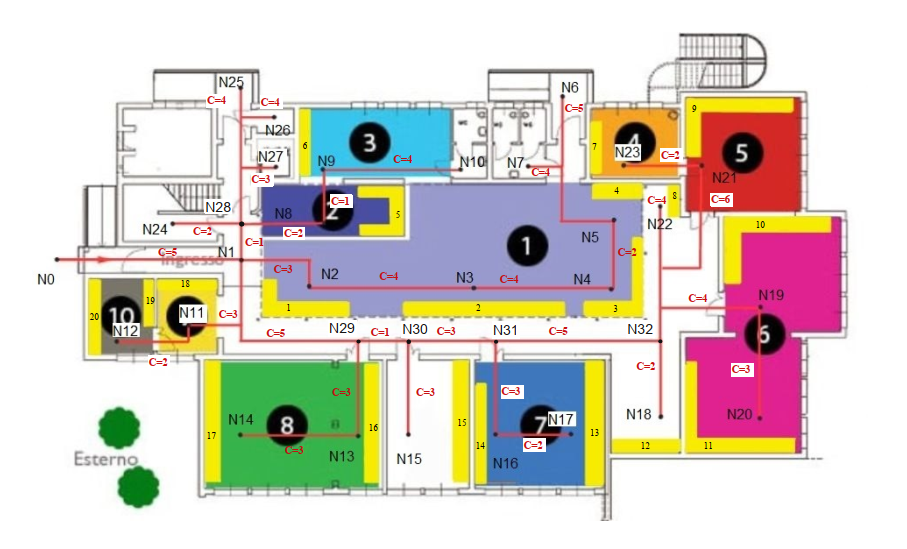


Figura - Planimetria della biblioteca

I vari nodi (numerati da N0 a N32) sono collegati tra di loro da degli archi pesati, dove il peso è rappresentato dalla funzione costo, che in questo caso stima la distanza in cm, basandoci sulla piantina, tra i nodi :

***<ni,nj> --> costo(<ni,nj>) ,***

dove :

* **ni,nj** sono due generici nodi
* con **<ni,nj>** indichiamo l'arco che li collega, con il peso equivalente alla distanza.

Nella planimetria, l'esito della funzione costo è indicato in forma compatta con C=numero;

per ragioni di spazio non abbiamo inserito i costi di alcuni percorsi intermedi:

|  |  |
| --- | --- |
| costo(<N25,N26>) = 2 | costo(<N22,N19>) = 6 |
| costo(<N25,N27>) = 3 | costo(<N21,N22>) = 6 |
| costo(<N26,N27>) = 3 | costo(<N21,N19>) = 8 |
| costo(<N29,N11>) = 5 | costo(< N6 , N7 >) = 3 |

All’interno della biblioteca, i libri sono partizionati per genere e organizzati in scaffali, rappresentati in giallo ed enumerati. Oltre al numero, ad ognuno di essi abbiamo associato un nodo, il più vicino.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| BOOK GENRE | LIBRARY SHELF | NODE | BOOK GENRE | LIBRARY SHELF | NODE |
| Novel | 1 | N2 | Romance | 11 | N20 |
| Narrative | 2 | N3 | Finance | 12 | N18 |
| Poem | 3 | N4 | Humor | 13 | N17 |
| Drama | 4 | N5 | Horror | 14 | N16 |
| Mythology | 5 | N8 | Fable | 15 | N15 |
| Fiction | 6 | N9 | Folklore | 16 | N13 |
| Scientific | 7 | N23 | Short Story | 17 | N14 |
| Math | 8 | N22 | Poetry | 18 | N11 |
| Crime | 9 | N21 | Fairy Tale | 19 | N12 |
| Computer Science | 10 | N19 | Historical | 20 | N12 |

A partire da tale spazio, ossia dalla planimetria in figura 1, abbiamo creato un grafo equivalente, dove il nodo radice corrisponde a N0 (che nella figura rappresenta l'ingresso all'edificio) , tutti gli archi, che collegano gli altri nodi, hanno associato il peso corrispondente e rappresentano tutti i vari percorsi individuati.

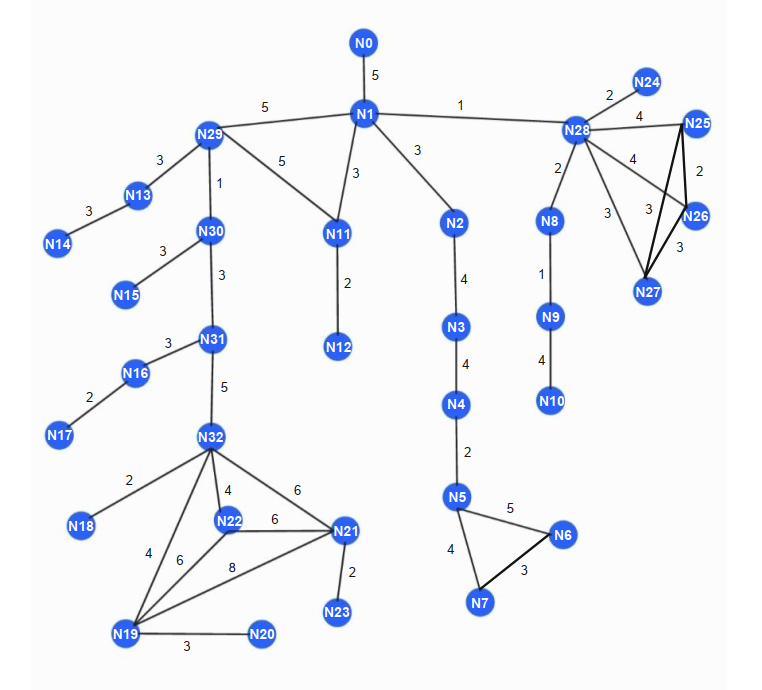


Figura – Grafo pesato corrispondente piantina in figura 1

Per adattarci meglio ad un’ipotetica situazione reale, abbiamo stabilito degli orari di apertura e chiusura della biblioteca. Gli orari rappresentano un vincolo, che lo studente (utente medio che potrebbe utilizzare il nostro sistema), deve tener presente per potervi accedere e prendere in prestito e/o restituire un qualsiasi libro.

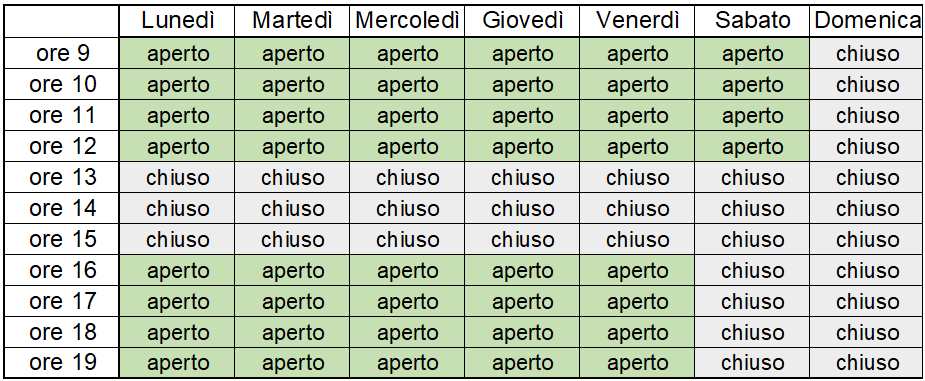


Figura – Tabella riassuntiva degli orari di apertura della biblioteca

In più, per effettuare delle simulazioni del sistema, in particolare per la prenotazione di un libro e la ricerca, si assume che la biblioteca contenga un catalogo di 28 libri.

Dopo queste fasi preliminari, utili per iniziare a modellare la realtà presa in considerazione, siamo passati a progettare un’ontologia adatta.

ONTOLOGIA

Una ontologia specifica il significato dei simboli utilizzati dal sistema. Essa si modella decomponendo la realtà, da rappresentare, individuandone individui e relazioni che intercorrono tra di essi. Le ontologie vengono impiegate, nei sistemi intelligenti, per ovviare alle problematiche che si incontrano durante l’acquisizione di conoscenza (per rappresentare una KB), in quanto, documentando le associazioni simboli -> concetto, riducono le ambiguità di interpretazione (dato che lo stesso simbolo, in base al dominio del problema, può assumere significati diversi) e favoriscono l’interoperabilità semantica e sintattica della conoscenza, ossia la capacità di diverse KB di collaborare tra loro, senza conflitti, rispettando il significato dei simboli.

Le ontologie distribuite sulla rete vengono descritte attraverso il linguaggio OWL (che sta per Web Ontology Language); esso permette di descrivere il mondo in termini di :

* Individui: entità del mondo
* Classi: insieme di individui, accomunati da delle caratteristiche comuni
* Proprietà: relazioni che associano agli elementi del proprio dominio (individui), un elemento del codominio (valori che può assumere). In particolare, si dividono in:
  + Datatype property : se il codominio contiene solo tipi primitivi (interi, stringhe,…)
  + Object property: quando il codominio contiene altre classi

Descrizione dell’ontologia

Per strutturare e definire la nostra ontologia abbiamo utilizzato il software l’applicativo **Protégé**, il quale ne permette la realizzazione in maniera intuitiva e completa, con l’aggiunta di plug-in molto utili al fine progettuale.

Innanzitutto, abbiamo creato il file dell’ontologia (.owl) per poi localizzarla, assieme a tutti i suoi elementi, in modo univoco nel Semantic Web (tecnologia che garantisce l’interoperabilità delle ontologie distribuite sulla rete), utilizzando l’Ontology IRI (Internationalized Resource Identifier).

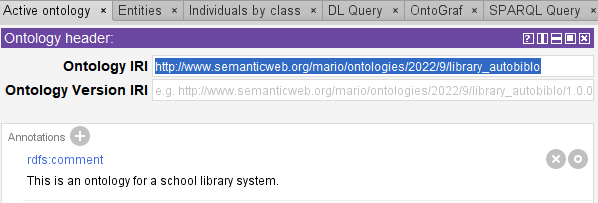


Figura - IRI Ontology

Come primo step per la realizzazione dell’ontologia su Protégé, abbiamo individuato le classi, che vanno a schematizzare il nostro sistema.

In base all’importanza dei concetti che esprimono, in relazione al nostro problema, abbiamo scelto di decomporre la nostra realtà nelle classi:

“Book”, “Author”, “Genre”, “Publisher”, “Room”, “Shelf” e “Student”.

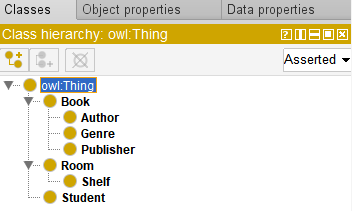


Figura - Classi

Dopo di che, pensando a ciò che il nostro sistema può manipolare e visualizzare, tramite delle query, abbiamo formulato le proprietà, suddivise in datatype e object properties:

|  |  |
| --- | --- |
| Immagine che contiene tavolo  Descrizione generata automaticamente | Immagine che contiene testo  Descrizione generata automaticamente |
| Figura 6 - Datatype properties | Figura - Object properties |

Dopo aver modellato le proprietà e, di conseguenza, le relazioni che intercorrono tra le classi/entità, prima di poter simulare il funzionamento di un sistema basato su questa ontologia, l’abbiamo popolata con le entità (individui) adatte ad ogni classe.

|  |  |
| --- | --- |
|  | Immagine che contiene testo  Descrizione generata automaticamente  Figura – Lista degli individui presenti nell’ontologia |

Ogni individuo possiede delle proprietà, che eredita dalla classe di appartenenza; esempio:

l’individuo *Franz Kafka*, appartenente alla classe Author, ha come datatype properties il proprio nome e cognome, ID e come object properties la relazione hasWritten, che lo connette ad un libro che ha scritto.

Nella figura, invece, è rappresentato un ulteriore esempio, per l’individuo *If this is a man*, che appartiene alla classe Book, con le proprietà descritte nell’ultima sezione.

Immagine che contiene testo

Descrizione generata automaticamente

Figura 9 - Esempio "if this is a man"

L’ontologia realizzata rappresenta e decompone, correttamente, tutti i concetti utili per modellare la realtà del nostro problema.

Interrogazioni

Il nostro sistema si basa sull’ontologia appena descritta; per poter interrogarla e visualizzarne il comportamento (così da poter verificare anche il funzionamento del sistema) abbiamo utilizzato il **Reasoner** di default del software (HermiT 1.4.3.456) sfruttando i due plug-in già integrati: “**DL query**” e “**SPARQL query**” .

Immagine che contiene testo

Descrizione generata automaticamente

Figura – Primo tool : DL query

Inizialmente, abbiamo effettuato le query tramite DL query:

|  |  |
| --- | --- |
| Immagine che contiene testo  Descrizione generata automaticamente  Figura - query 1 | Immagine che contiene testo  Descrizione generata automaticamente  Figura - query 2 |
| Immagine che contiene testo  Descrizione generata automaticamente  Figura - query 3 | Immagine che contiene testo  Descrizione generata automaticamente  Figura - query 4 |

Poi siamo passati a SPARQL

Immagine che contiene testo

Descrizione generata automaticamente

Figura – Secondo tool: SPARQL

|  |  |
| --- | --- |
| Immagine che contiene tavolo  Descrizione generata automaticamente  Figura 16 - SPARQL query n.1 | Figura 17 - SPARQL query n.2 |
| Figura 18 - SPARQL query n.3 | Figura - SPARQL query n.4 |

Una volta verificato il giusto funzionamento di tutte le componenti dell’ontologia, tramite le interrogazioni varie, abbiamo creato il grafo corrispondente, utilizzando il plugin “**Ontograf**”, che permette di **visualizzare graficamente** tutte le componenti e le relazioni. Ogni nodo del grafo indica una classe, mentre gli archi indicano le proprietà (archi orientati, grazie ai quali si individuano facilmente i rispettivi domini e codomini).

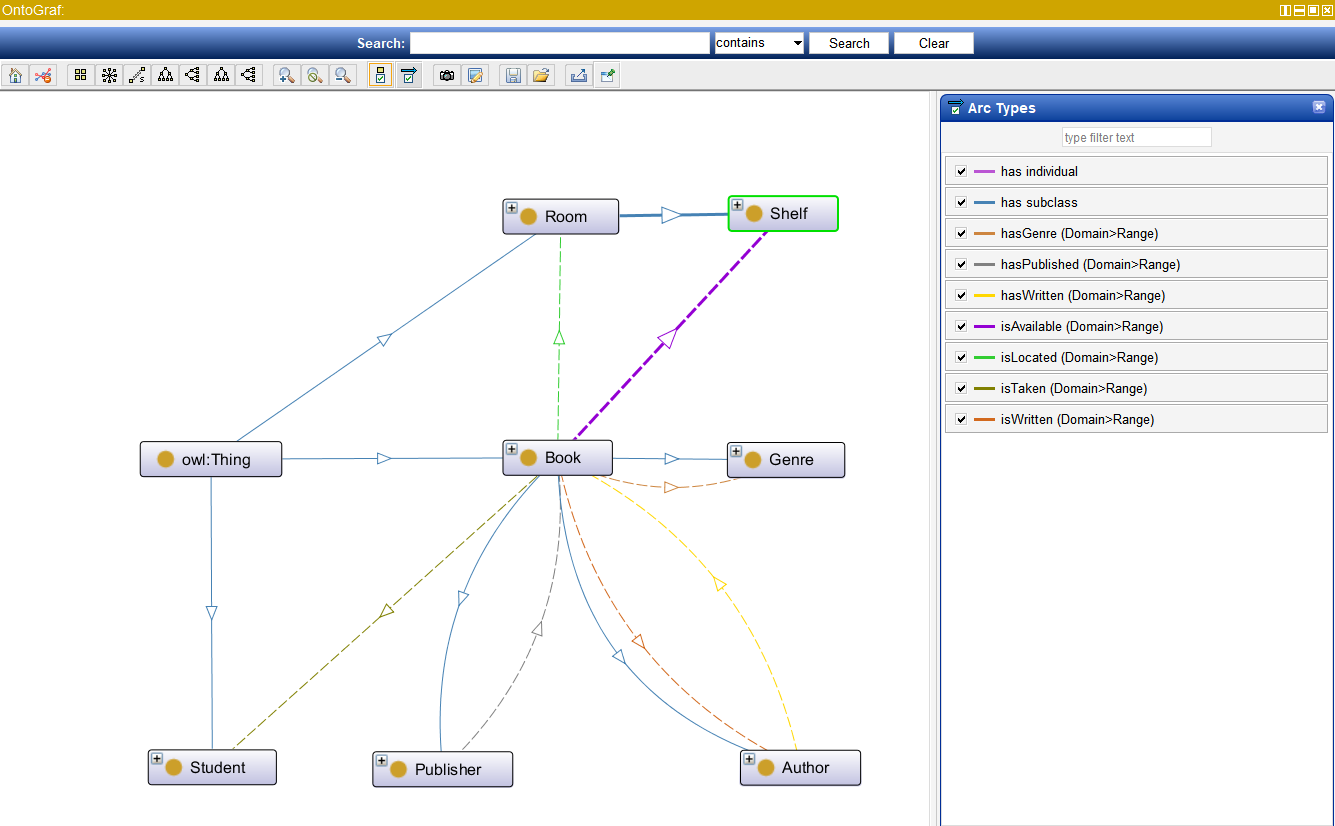


Figura – Rappresentazione dell’ontologia mediante un grafo

Di seguito, l’ontologia è stata interamente codificata in Python e inserita nel progetto, nel modulo “**Ontology.py**”, attraverso il quale è possibile visualizzarla.

CONSTRAINT SATISFACTION PROBLEM

Come abbiamo già detto, il nostro progetto sviluppa un’automazione della gestione di una biblioteca scolastica. Una delle realtà che abbiamo voluto rappresentare riguarda il prestito dei libri, perché è l’azione che più frequentemente viene effettuata in un contesto come il nostro.

Prima di prendere in prestito un libro, in una comune biblioteca, bisogna considerare alcuni aspetti:

1. Innanzitutto, dato che questa operazione viene effettuata in sede, devo rispettare gli orari di apertura della biblioteca, poiché non posso prendere un libro se tale servizio non è accessibile;
   * Nel nostro caso, l’orario è riassunto nella tabella della Figura 3
2. Di seguito, ne va verificata la disponibilità negli archivi, al momento della richiesta, in quanto se un libro è stato già preso da un’altra persona, oppure, la biblioteca non ne dispone nemmeno una copia, l’operazione non è fattibile.

Un discorso analogo vale per la restituzione dei libri (precedentemente presi in prestito).

Riportando questa situazione nell’ambito dell’intelligenza artificiale, ci è sembrato che il modo più consono per risolverla fosse trattarla come un **CSP.**

Per CSP si intende un problema di soddisfacimento dei vincoli, ed è composto da una tripla

**< X , D, C >**

Dove :

* **X** = insieme delle variabili {X1, X2, …, Xn} ;
* **D** = insieme dei domini {d1, d2, …, dn}, dove il generico dominio *di* contiene i valori ammessi dalla variabile *Xi* ;
* **C** = insieme dei vincoli (rigidi) che specificano le combinazioni di assegnazioni ammesse.

La soluzione di un problema così strutturato è un modello, ossia un’assegnazione totale (fatta su tutte le variabili, rispetto ai propri domini) consistente (che rispetti tutti i vincoli specificati in C).

Per adattarci ad un problema simile, abbiamo modellato la nostra realtà, in modo da ricavare un insieme di variabili, individuare i relativi domini e tradurre in vincoli le riflessioni fatte in precedenza.

***Individuare le variabili***

L’oggetto principale del nostro sistema è il libro (da prendere in prestito o da restituire), perciò è appropriato prendere *Libro* come variabile.

L’osservazione 1 verte sull’importanza di rispettare l’orario di apertura, per tanto, rifacendoci alla figura 3, che lo riassume in giorni e fasce orarie, le altre due variabili sono rappresentate da *Giorno* e *Ora*.

Ricapitolando, il nostro insieme delle variabili è composto da {Libro, Giorno, Ora}.

***Individuare i domini***

Dopo aver individuato le variabili, siamo passati a definire i domini (insieme di possibili valori assumibili) per ognuna, facendo delle semplici riflessioni:

* Dato che la biblioteca è aperta 6 giorni su 7 (dal lunedì al sabato), il dominio della variabile *Giorno* è rappresentato dall’insieme : {1,2,3,4,5,6} (dove ogni numero corrisponde ad un giorno della settimana)
* La biblioteca, generalmente, è aperta dalle 9 di mattina alle 19 di sera; perciò, tale intervallo rappresenta il dominio entro il quale la variabile *Ora* può variare
* Infine, il dominio della variabile *Libro*, così come ci suggerisce l’osservazione 2, è costituito dalla lista di libri disponibili per il prestito (o per il reso, a seconda del caso)

Nota: dato che sia l’insieme delle variabili, sia i singoli domini, sono di cardinalità finita, in questo caso siamo in presenza di un CSP finito.

***Individuare i vincoli***

Per codificare i vincoli sulle assegnazioni *Variabile – Valore nel dominio* , ci siamo basati principalmente sull’orario di apertura della biblioteca e sull’osservazione 2, che impone la disponibilità del libro per effettuare un’azione.

In generale, abbiamo imposto che:

* La variabile *Libro* corrisponda necessariamente ad un libro della lista
* La variabile *Giorno* possa variare da 1 a 6
* La variabile *Ora* rispetti i turni della biblioteca, dove per la mattina 9 <= Ora < 12, mentre per il pomeriggio 16 <= Ora < 19

In più, abbiamo incluso un vincolo “speciale” che interessa solo il sabato, poiché in questo giorno la nostra biblioteca risulta aperta solo di mattina; perciò, le uniche assegnazioni valide saranno quelle che rispetteranno:

Libro Lista di libri & Giorno = 6 & 9<= Ora < 12

Dopo aver modellato la nostra realtà, così da adattarla ad un problema di soddisfacimento dei vincoli, siamo passati alla codifica del csp in Python, nell’omonimo modulo “**Csp.py**”.

Immagine che contiene testo

Descrizione generata automaticamenteNello specifico, abbiamo incluso nel modulo la libreria “**Constraint**”, la quale contiene metodi adatti a trattare problemi simili.

Figura - Porzione del codice utilizzata per codificare il csp in python

La prima operazione fatta, per codificare il csp individuato per il nostro caso, è stata istanziare la classe Problem, la quale ci ha permesso di definire il nostro problema e ottenere delle soluzioni.

Dopodiché, abbiamo aggiunto all’oggetto problem :

* le variabili e i domini, attraverso il metodo addVariable(), specificando le singole variabili individuate e i rispettivi domini;
  + nota: dato che stiamo automatizzando il prestito e la restituzione di un articolo, il dominio della variabile *Libro* è rappresentato da una lista dinamica e non da un insieme statico, come per gli altri casi. La lista varia in base all’operazione da svolgere e si aggiorna al termine dell’azione.
* la codifica dei vincoli, sfruttando il metodo addConstraint(), con il quale abbiamo specificato le assegnazioni (variabile-elemento del dominio) ammesse dal nostro problema, trasformando i turni della biblioteca in restrizioni per Giorno e Ora e imponendo alla variabile Libro di assumere un valore che rispetti il proprio dominio

Dopo aver correttamente creato il nostro csp in Python, utilizzando il metodo getSolution() abbiamo ricavato tutti i modelli che possono rappresentare una soluzione per il nostro problema. Nello specifico, getSolution() restituisce una lista di dizionari che mappano le assegnazioni possibili tra variabili e valori, e lo fa sfruttando un *solver*.

Il solver di default, stabilito dalla libreria Constraint, è il Backtracking (seleziona un’opzione e, prima di passare alla successiva, completa l’intero percorso), ma in alternativa è possibile utilizzare il Recursive Backtracking o il Minimum Conflicts.

Data la natura del nostro problema, non troppo complicato, con uno spazio di ricerca ridotto e senza cicli infiniti, ci è sembrato consono utilizzare il solver di default.

Per poter stabilire se l’utente possa prendere in prestito, o rendere, un libro, effettuiamo un confronto tra i dati inseriti (giorno, ora e nome del libro) e la lista prodotta da getSolution, con lo scopo di trovare un modello per cui le assegnazioni rispecchiano i dati di input:

* Se tale modello esiste, vuol dire che l’utente ha rispettato tutti i vincoli, perciò è possibile effettuare l’operazione.
* Qualora i dati inseriti dall’utente non rispettassero i vincoli, dal confronto non si evidenzierà alcuna corrispondenza con i modelli consistenti e verrà comunicato allo studente l’impossibilità di procedere.

RICERCA SUL GRAFO

Per arricchire il nostro progetto, abbiamo pensato di collegare la parte del csp alla ricerca sul grafo: nel momento in cui viene preso in prestito un libro, il sistema indicherà allo studente il percorso da fare per recarsi allo scaffale dove si trova il libro in questione, così da poterlo prendere autonomamente.

Come già detto nell’introduzione, abbiamo ricavato un grafo pesato a partire dalla planimetria della piantina, evidenziando così tutti i percorsi possibili. Tale grafo rappresenta il nostro spazio di ricerca; viste le sue dimensioni ridotte, abbiamo deciso di utilizzare il metodo di ricerca il Best-First-Search, un algoritmo di ricerca informata che, per risolvere il problema, sfrutta la conoscenza a disposizione (nel nostro caso, le informazioni rispetto al peso degli archi) per restituire il percorso più corto e che rappresenti la soluzione ottimale (ossia, cammino con il minimo numero di archi e il minimo costo).

Tale metodo, utilizzato nel campo dell’Intelligenza Artificiale, si occupa di espandere i nodi del grafo, così da aumentare la distanza dalla radice(strarting node) fino al raggiungimento del goal node (ossia il nodo in corrispondenza dell’obiettivo da raggiungere)

La completezza e l’ottimalità di questo algoritmo sono altri due motivi che ci hanno portato a sceglierlo, poiché se esiste una soluzione, riesce sempre a ritrovarla e a restituire quella con il costo minimo

*Implementazione dell’algoritmo di ricerca:*

Immagine che contiene testo

Descrizione generata automaticamente

Figura - Algoritmo

Questo codice prende in input il nostro grafo e produce il percorso a costo minimo, ossia la sequenza di nodi **<N0, …, Nk>** , dove N0 indica l’ingresso alla biblioteca( dove si trova la reception) ed Nk lo scaffale di interesse, dov’è contenuto il libro.

Per la realizzazione e la stampa del grafo abbiamo incluso la libreria “**Igraph**”.

Il grafo, passato in input all’algoritmo di ricerca, viene creato “virtualmente” dalla funzione setup\_graph(), rispettando i nodi, gli archi e i relativi pesi.

Immagine che contiene testo, elettronico, tastiera

Descrizione generata automaticamente

Figura – Porzione della funzione setup\_grap(), nella quale vengono inseriti manualmente tutti gli archi e i pesi relativi

Immagine che contiene testo

Descrizione generata automaticamente

Figura - Porzione della funzione setup\_graph, dove si utilizzano i metodi della libreria igraph

Le relazioni tra scaffali, contenenti i libri, e i nodi che gli identificano (tale associazione scaffale – nodo è riassunta nella tabella a pagina 4) sono codificate dalle funzioni get\_nodes (che restituisce a partire dallo scaffale, il nodo corrispondente) e get\_shelf (che restituisce lo scaffale dov’è stipato il libro).

|  |  |
| --- | --- |
| Figura 25 - Pozione della funzione get\_nodes | Figura - Porzione della funzione get\_shelf |

La stampa, effettuata dopo aver trovato il cammino minimo, viene effettuata dalla funzione print\_solution, che crea un’immagine del grafo completo nel file “grafico.png” (situato all’interno della cartella src), dov’è possibile visualizzare tutti i nodi etichettati, i vari archi pesati ed il percorso restituito (i cui nodi sono evidenziati in viola, rispetto a tutti gli altri che sono gialli).

Immagine che contiene testo

Descrizione generata automaticamente

Figura - Funzione print\_solution

*Esempio di una ricerca*:

Immagine che contiene testo

Descrizione generata automaticamente

Figura - Esempio dell'utilizzo della ricerca nel grafo

Come si può vedere, dopo aver correttamente effettuato l’operazione di “borrow a book”, e aver richiesto di visualizzare il cammino, l’algoritmo

* dapprima fornisce delle informazioni sul libro preso, ricapitolandone il nome e specificando lo scaffale in cui è stipato (con associato il nodo che lo identifica);
* di seguito, prosegue stampando il percorso (Path) sotto forma di sequenza di nodi, che partono dalla reception, fino ad arrivare allo scaffale di interesse, ossia:

< 0, 1, 29, 30, 31, 16>

E come si può vedere, il nodo 16 (identifica lo scaffale) rappresenta il nodo “goal”

* In contemporanea alla stampa della sequenza di nodi, il programma plotta, sul grafo presente nel file ‘grafico.png’, il percorso corrispondente (in viola)

Immagine che contiene accessorio, collana

Descrizione generata automaticamente

Figura - output plottato sul grafo

Per avere un’idea reale del percorso restituito dall’algoritmo, lo abbiamo riportato sulla planimetria della biblioteca, cosicché lo studente possa avere delle indicazioni più precise sulla strada da intraprendere.

Immagine che contiene testo, microscopio

Descrizione generata automaticamente

Figura – percorso riportato sulla planimetria

RECOMMENDER SYSTEM – KNN

Con l’obiettivo di incentivare gli studenti alla lettura, nel nostro programma abbiamo inserito una forma di apprendimento supervisionato, ossia un recommender system che, partendo da un dataset di libri, potesse consigliare allo studente altre letture simili a quella di interesse.

I sistemi di raccomandazione sono largamente utilizzati, poiché permettono di creare raccomandazioni personalizzate, rispetto alle preferenze dell’utente. Ne esistono di diverse tipologie e quello che abbiamo implementato noi è il **KNN (k-nearest neighbors),** che rappresenta uno degli algoritmi di apprendimento supervisionato più popolari per i problemi di classificazione.

Abbiamo scelto il KNN, poiché ci è sembrato il più adatto per il nostro obiettivo, ossia scandire una lista dei libri più simili a quello specificato dall’utente, basando le predizioni su un dataset.

Esso si compone di una fase di apprendimento che, a differenza di altri algoritmi dove si costruisce un modello per fare le predizioni, si limita a memorizzare gli esempi del training set (nel nostro caso, il dataset con i libri e le relative valutazioni) e di una fase di classificazione, che consiste nel restituire i K elementi più simili al nuovo esempio da classificare, restituendone la lista.

Tutto ciò viene fatto servendosi di una misura di similarità, ossia di una metrica che stimi quanto siano simili il libro, fornito in input dall’utente, e ogni altro libro presente nel dataset. Grazie alla misura di similarità, infatti, siamo in grado di individuare i K libri più simili, da consigliare all’utente.

La misura di similarità utilizzata per il nostro problema è nota come *similarità del coseno*, la quale misura la somiglianza tra due vettori. Nel nostro caso i 2 vettori su cui, volta per volta, si calcola la metrica, sono :

* Il vettore delle valutazioni fatte sul libro dato in input
* Il vettore delle valutazioni dell’i-esimo libro, tra quelli presenti nel dataset, che stiamo attualmente prendendo in considerazione

*Fasi preliminari dell’implementazione*

Il primo passo per l’implementazione del sistema di raccomandazione è stato cercare un dataset adatto al nostro scopo.

Nella cartella “data” del nostro progetto sono presenti 3 file .csv:

* BX-Books.csv, con tutte le informazioni sui libri presenti nel dataset (ISBN, titolo, autore, anno di pubblicazione,)
* BX-Users.csv, con le informazioni relative agli utenti che hanno fornito le valutazioni (id, posizione e età)
* BX-Book-Ratings.csv, che collega e riassume i libri alle valutazioni fornite dagli utenti

Che abbiamo deciso di unire, con un merge, per fornire un dataset più completo al nostro algoritmo, con tutte le informazioni dei libri e le valutazioni.

Prima di utilizzare il dataset, abbiamo filtrato gli elementi, ponendo due condizioni:

* Ogni libro deve aver ricevuto almeno 10 voti,
* Ciascun utente deve aver espresso almeno 10 voti.

Per gestire al meglio un dataset di tali dimensioni abbiamo utilizzato la libreria “**Pandas**”.

Di seguito abbiamo impostato il parametro K a 10, perciò il sistema restituirà una lista con i 10 libri più simili a quello fornito in input.

*Funzionamento*

Immagine che contiene testo

Descrizione generata automaticamente

Innanzitutto, lo studente inserisce il nome del libro di suo interesse.

Di seguito, viene istanziata la classe KnnRecommender (da noi creata, contenente tutti i metodi utili) e richiamato il metodo make\_recommendations(), che restituisce l’output, ossia la lista di libri raccomandati.

Una delle prime cose che fa la funzione make\_racommendations è restituire, utilizzando la funzione *fuzz.ratio* della libreria “**Fuzzywuzzy**”, una lista di titoli e indici dal dataframe , il cui titolo ha una certa corrispondenza , espressa in percentuale, con quello del libro dell'utente (secondo la distanza di Levenshtein, che misura la differenza tra due stringhe).

Questa lista viene processata dalla funzione NearestNeighbors() della libreria “**Sklearn.neighbors**” con la quale viene stilata la lista di raccomandazioni, utilizzando la similarità del coseno.

Dato che abbiamo trovato un’incoerenza tra alcuni libri presenti nei file .csv delle valutazioni utenti e dei libri, abbiamo filtrato la lista delle raccomandazioni ottenute in precedenza, secondo i dati appartenenti esclusivamente al dataframe dei libri.

VALUTAZIONE DEI RISULTATI

In questa sezione abbiamo riassunto un po’ quelle che sono le nostre considerazioni rispetto al sistema appena creato, basandoci sui risultati ottenuti da una serie di test.

L’obiettivo comune a tutti i test effettuati è vedere come il sistema risponde ai diversi input, che rispecchiano non solo le normali situazioni di normale utilizzo, ma anche i casi limite, come ad esempio l’uso di input errati.

*Test relativi al sistema di prenotazione/reso di un libro (Csp):*

|  |  |  |
| --- | --- | --- |
| #Test | Obiettivo test | Risultato ottenuto |
| 1 | Effettuare la prenotazione di un libro disponibile, come It, di lunedì, alle 10.10 (questo test rappresenta un caso in cui tutti i vincoli sono rispettati) | L’operazione è andata a buon fine. Il sistema lavora correttamente con gli input corretti |
|  | | |
| 2 | Verificare che il sistema di prenotazione, inserendo input corretti, non sia case sensitive | Scrivere il nome del libro tutto in minuscole (mindhunter) o tutto in maiuscole (THE TRIAL) non impatta sul corretto funzionamento |
|  | | |
| 3 | Provare a prenotare un libro disponibile, inserendo input scorretti per il Giorno o l’ora. | Il sistema comunica correttamente all’utente il verificarsi di un errore e termina il programma |
|  | | |
| 4 | Provare a prenotare un libro non disponibile al momento(es: the idiot)  o che proprio non è presente nella biblioteca  (es: ciao) | In entrambi i casi il sistema non permette di prendere il libro, comunicandolo all’utente. Nel caso 2 viene visualizzato un messaggio in più poiché la stringa inserita rappresenta un input scorretto. |
|  | | |

Nota: i test sopra effettuati sono stati replicati per il caso “return”, ottenendo gli stessi esiti

*Test relativi alla ricerca del libro nella biblioteca (Ricerca nel grafo):*

il modulo della ricerca nel grafo, utilizzato per ricavare il percorso dall’ingresso della biblioteca, allo scaffale dov’è contenuto il libro, viene richiamato solo dopo che lo studente ha preso in prestito un libro e non richiede l’inserimento di altri dati; perciò, l’interazione con l’utente è minima in questa fase.

Per tali ragioni, in questo caso, non è possibile verificare la correttezza del sistema in situazioni particolari. Ci limitiamo quindi a fornire un semplice esempio:

|  |  |  |
| --- | --- | --- |
| #Test | Obiettivo test | Risultato ottenuto |
| 1 | Dopo aver preso in prestito un libro, visualizzare il percorso che porti lo studente allo scaffale in cui è contenuto | Il sistema elenca la sequenza di nodi che interessano il percorso e lo plotta correttamente sul grafo |
|  | | |

*Test relativi alla raccomandazione dei libri (Recommender system con KNN):*

|  |  |  |
| --- | --- | --- |
| #Test | Obiettivo test | Risultato ottenuto |
| 1 | Stilare una lista di 10 libri simili a quello dato in input | Il sistema restituisce la lista dei libri con titolo simile a quello scelto, di seguito esprime le 10 raccomandazioni per il titolo dato in input |
| Immagine che contiene testo  Descrizione generata automaticamente | | |
| 2 | Da quest’ultimo test, come possiamo notare, nella lista delle raccomandazioni ci sono delle posizioni contrassegnate da “not found”. Questo non indica un errore del recommender system, ma solo il verificarsi di un’incoerenza nel dataset (di cui abbiamo parlato nella sezione dedicata al KNN) | |
| Immagine che contiene testo  Descrizione generata automaticamente | | |

*Test dell’Ontologia:*

Il nostro programma, più che manipolare l’ontologia, permette di visualizzarla, stampando la lista degli individui che popolano le singole classi e delle query di esempio (permettono di verificare che l’ontologia sia stata correttamente codificata).

Il modo migliore per testare l’ontologia è effettuare delle query in SPARQL, come abbiamo già fatto nel capitolo dedicato, utilizzando il file **library.owl** (dov’è presente per intero l’ontologia descritta, in formato OWL).È possibile prendere spunto dal file **ontology\_queries.txt**, nella cartella docs, dove sono presenti delle query già fatte.

Considerazioni finali sui test

I test effettuati hanno evidenziato un corretto funzionamento del sistema, anche nei casi limite, in cui sono stati inseriti input errati, o in situazioni atipiche.

Considerando che:

* i risultati ottenuti sono sempre coerenti con le aspettative,
* le situazioni più insolite, che sono state gestite opportunamente con la raccolta delle eccezioni, non portano ad arresti anomali dell’applicazione,
* e soprattutto, il sistema intrattiene una buona comunicazione con l’utente, in quanto, dopo ogni azione corretta o meno che sia, stampa sempre dei messaggi riguardanti l’esito dell’operazione compiuta,

ci sentiamo soddisfatti del lavoro svolto. Il sistema realizzato, rispecchia la nostra idea iniziale e si adatta bene realtà che abbiamo preso in considerazione e modellato.

NOTE FINALI

Per la realizzazione di questo progetto abbiamo lavorato in pair-programming, sfruttando il repository GitHub appositamente creato.

Durante la durata del progetto, come canali di comunicazione, abbiamo utilizzato un gruppo Telegram ed un canale su Teams, che abbiamo sfruttato soprattutto per fare delle riunioni, per discutere del lavoro fatto o su come suddividerci i compiti restanti.