Università degli Studi di Bari
Dipartimento di Informatica

Spreading Activation Intelligente

Autore
Federico Maiorano

Intelligenza Artificiale
a.a. 2017-18

Indice

Introduzione

Nello studio qui presentato si è cercato di sviluppare un algoritmo intelligente per l'analisi di reti semantiche e per la scoperta di nuova conoscenza. Nello specifico, è stata realizzata una prima versione di quello che viene qui definito come algoritmo di "spreading activation intelligente". L'algoritmo è infatti una versione modificata di un classico algoritmo di spreading activation, a cui è stata aggiunta una componente intelligente, che sfrutti una qualche conoscenza che si ha sulla rete.

La *spreading activation*, in italiano chiamata *diffusione dell'attivazione*, è un metodo di ricerca per l'analisi di reti associative, neurali o semantiche, utile anche nel campo dell'information retrieval. Partendo da un insieme di nodi sorgente segnati con un valore di attivazione, questi valori vengono propagati verso i nodi adiacenti a quelli sorgente. Ogni volta che un nuovo arco viene percorso, il valore di attivazione decade progressivamente.
L'algoritmo intelligente sviluppato mira a calcolare dinamicamente il valore di decadenza da attribuire ad ogni arco percorso nel ciclo di diffusione dell'attivazione. Questo valore dinamico è stabilito a partire da una forma di conoscenza che si ha della rete. Nello specifico, dall'interesse che certe entità mostrano verso altre entità presenti nella stessa rete.
Per lo sviluppo dell'algoritmo si è utilizzato il linguaggio di programmazione Prolog, impiegato in molti programmi di intelligenza artificiale. Si è partiti dallo sviluppo in Prolog di un algoritmo di spreading activation tradizionale come quello di cui si parlerà nel capitolo 1, sezione 1.1. Questa implementazione è stata modificata inserendo la componente intelligente, che, consultando la base di conoscenza Prolog, elabora il peso dinamico degli archi.

L'algoritmo creato è stato testato sulla rete semantica "Gr@phBRAIN". Questa rete è suddivisa al suo interno in differenti domini. Al momento i domini presenti solo quelli del retrocomputing e del turismo. Per le sperimentazioni eseguite in questo studio, si è aggiornato il dominio attinente al turismo, andando ad inserire nel database Gr@phBRAIN informazioni riguardo alcu-

ni luoghi di interesse turistico della città di Bari e su un insieme di dipinti esposti nella pinacoteca cittadina.
Negli esperimenti si sono messi a confronto l'algoritmo intelligente con quello tradizionale, e l'implementazione in Prolog con quella realizzata in un altro sistema.

La relazione è così strutturata: nel primo capitolo sono presentati formalmente gli algoritmi di spreading activation tradizionale e l'algoritmo di spreading activation intelligente, di cui viene mostrata anche l'implementazione in Prolog; nel secondo capitolo sono esposte le sperimentazioni effettuate con l'algoritmo sviluppato; infine, nel terzo ed ultimo capitolo sono raccolte le conclusioni del lavoro.

Spreading activation

1.1 Spreading activation tradizionale

L'algoritmo di **Spreading activation** è un metodo di ricerca per reti associative, neurali e semantiche. A partire da un insieme di nodi sorgente etichettati con dei pesi (detti anche attivazioni), l'algoritmo propaga questi pesi verso i nodi collegati e, ogni volta che un nuovo arco viene percorso, il valore dei pesi decade progressivamente.

Dato un grafo orientato composto da n nodi, ognuno con un valore di attivazione associato a_i (numero reale compreso tra 0 e 1), e da un insieme di archi del tipo $e_{i,j}$ che connettono il nodo i con il nodo j, ognuno con un peso associato $w_{i,j}$ (di solito un numero reale compreso tra 0 e 1), un algoritmo di spreading activation generico funziona nel seguente modo:

1. Si stabiliscono due parametri, la soglia di attivazione F e il fattore di decadimento D, entrambi numeri reali compresi tra 0 e 1.

2. Si inizializza il grafo settando, per ogni nodo i, il valore di attivazione a_i a 0. Per il nodo o i nodi sorgente si imposta un valore di attivazione maggiore di F (generalmente pari a 1).

3. Per ogni nodo i del grafo non ancora esploso e avente un valore di attivazione a_i maggiore della soglia F:

4. Per ogni arco $e_{i,j}$ che connette il nodo sorgente i al nodo destinazione j,si imposta $a_j = a_j + (a_i * w_{i,j} * D)$, dove D è il fattore di decadimento.

5. Se il valore risultante a_j è maggiore di 1, si imposta a_j a 1. Invece, se il valore a_j è minore di 0, si imposta a_j a 0.

6. Il nodo i è segnato come esploso.

7. I nodi che hanno ricevuto una nuovo valore di attivazione che supera la soglia F sono i nodi da esplodere nel successivo ciclo di attivazione.

8. L'algoritmo termina quando non ci sono più nodi da esplodere oppure dopo un certo numero di iterazioni, stabilite all'avvio dell'algoritmo.

La direzione degli archi è trascurabile. L'algoritmo funziona nella stessa maniera sia per un grafo orientato che per uno non orientato.

Esempio Si consideri il grafo in figura 1.1, composto da 9 nodi e i cui archi hanno tutti egual peso pari a 0.9.
Viene eseguito l'algoritmo di spreading activation a partire dal nodo n_1, la cui attivazione viene impostata a 1. Il parametro F è pari a 0.3 mentre il fattore D è 0.8. Dopo 5 iterazioni dell'algoritmo, i valori di attivazione sono quelli indicati all'interno dei nodi.
Ad esempio, per il nodo n_2 l'attivazione è 0.72, ottenuta dalla moltiplicazione $1 * 0.9 * 0.8$, dove 1 è l'attivazione di n_1, 0.9 è il peso dell'arco che connette n_1 a n_2, e 0.8 è il fattore di decadimento D.
Per il nodo n_7, invece, il valore di attivazione è 0.54, somma di 0,27 ($0.373 * 0.9 * 0.8$), valore ottenuto esplorando l'arco che va n_5 a n_7, e 0.27, valore uguale ottenuto partendo dal nodo n_6.
Infine, il nodo n_9 ha un valore di attivazione pari a 0, dato che si è scelto di effettuare solo 5 iterazioni dell'algoritmo.

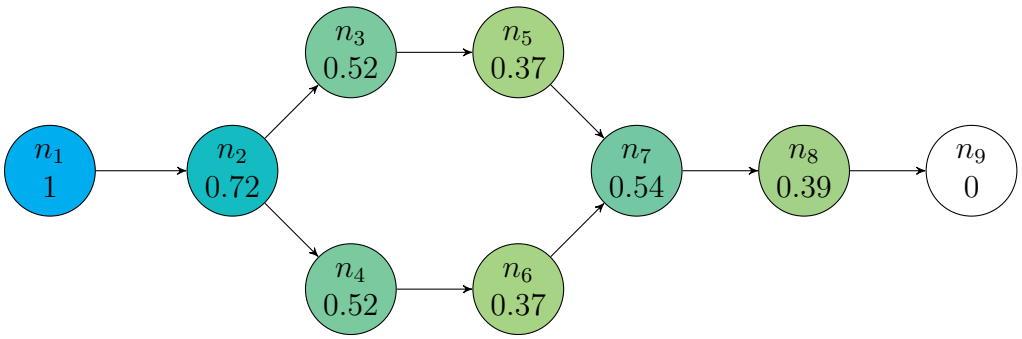

Figura 1.1: Esempio di spreading activation

1.2 Spreading activation intelligente

Immaginiamo di avere una rete semantica che rappresenti l'itinerario di una gita di un turista in una città. Sappiamo che il turista vuol visitare un determinato museo che espone un'opera d'arte di suo interesse. Siamo interessati

a conoscere il grado di interesse che il turista può mostrare rispetto ad altre opere presenti nel museo.
Una possibile soluzione a questo problema, potrebbe essere quella di assegnare dei pesi dinamici agli archi del grafo, partendo dal nodo di partenza (il turista) e sfruttando la conoscenza che si ha sui suoi interessi. Il peso di un generico arco $e_{i,j}$, che connette due nodi n_i e n_j, rappresenta l'interesse del turista rispetto al nodo n_j.
Per rispondere a questioni simili, anche in contesti differenti, si è scelto di modificare un generico algoritmo di spreading activation, inserendone all'interno un meccanismo intelligente, che faccia uso della conoscenza che si ha sui dati. Nello specifico, a differenza dell'algoritmo presentato nella precedente sezione, non ci sarà più un fattore di decadimento D e i pesi degli archi non dovranno essere necessariamente stabiliti a priori. Sarà compito dell'algoritmo calcolare dei pesi dinamici da assegnare agli archi, sfruttando la base di conoscenza.

Algoritmo Il peso dinamico calcolato per un generico arco $e_{i,j}$ del grafo è indicato con il termine I, e sostituisce nella formula del precedente algoritmo di spreading activation il prodotto $w_{i,j} * D$. Non è preclusa, tuttavia, la possibilità di definire a priori i pesi di alcuni archi: per questi archi il fattore I non verrà calcolato e si utilizzerà come peso quello definito nella base di conoscenza. Inoltre sarà possibile ignorare alcuni archi, senza andare a determinare per essi alcun peso.
L'algoritmo sarà suddiviso nei seguenti passi:

1. Si stabilisce il parametro F la soglia di attivazione, numero reale compreso tra 0 e 1.

2. Si inizializza il grafo settando, per ogni nodo i, il valore di attivazione a_i a 0. Per il nodo o i nodi sorgente si imposta un valore di attivazione pari a 1.

3. Per ogni nodo i del grafo non ancora esploso e avente un valore di attivazione a_i maggiore della soglia F:

4. Per ogni arco $e_{i,j}$ che connette il nodo sorgente i al nodo destinazione j:

   - se l'arco $e_{i,j}$ è un arco da ignorare, non si aggiorna il valore di a_j.
   - se il peso dell'arco $e_{i,j}$ è un valore $w_{i,j}$ definito, si imposta $a_j = w_{i,j}$.
   - altrimenti, si imposta $a_j = a_j + (a_i * I)$, dove I è il fattore intelligente.

5. Se il valore risultante a_j è maggiore di 1, si imposta a_j a 1. Invece, se il valore a_j è minore di 0, si imposta a_j a 0.

6. Il nodo i è segnato come esploso.

7. I nodi che hanno ricevuto una nuovo valore di attivazione che supera la soglia F sono i nodi da esplodere nel successivo ciclo di attivazione.

8. L'algoritmo termina dopo un certo numero di iterazioni stabilite all'avvio dell'algoritmo.

Calcolo del fattore I Il fattore I viene calcolato considerando gli interessi del nodo o dei nodi di origine dell'algoritmo. Si confrontano i nodi con quelli di interesse della stessa tipologia, valutando gli attributi e i vicini dei nodi. Sia S l'insieme dei nodi sorgente attivati nel secondo passo dell'algoritmo, e sia $e_{i,j}$ l'arco che va dal nodo n_i al nodo n_j, arco di cui si vuol stabilire il fattore I. I è così calcolato:

1. Per ogni nodo n_s dell'insieme S dei nodi sorgente:

2. Seleziona i nodi interessanti per il nodo n_s che siano della stessa tipologia del nodo n_j. Sia A l'insieme di questi nodi. Per ogni nodo n_a contenuto in A:

3. Se n_j è proprio il nodo n_a, allora I è uguale ad 1.

4. Altrimenti, conta il numero di attributi che il nodo n_j condivide con il nodo n_a. Sia I_{attr} uguale a questo numero moltiplicato per 0.1.

5. Conta il numero di coppie arco-nodo che il nodo n_j condivide con il nodo n_a, escludendo l'arco $e_{i,j}$. Sia I_{adj} uguale a questo numero moltiplicato per 0.25.

6. Il fattore I relativo al nodo n_j sarà uguale a:

$$\sum_{n_s \in S} \sum_{n_a \in A} I_{attr} + I_{adj}$$

1.3 Implementazione

L'algoritmo è stato scritto in Prolog. Viene di seguito trascritto il contenuto dello script Prolog (*spreadingINT.pl*) necessario per l'esecuzione dell'algoritmo di spreading activation intelligente:

```
%Read from two different file, the list of nodes and the list of edges.
%Skips headers.
prepare_db(FileNodes,FileEdges):-
        csv_read_file(FileNodes, [_|N], []),
        rows_to_lists(N, Nodes),
        assertz(nodes(Nodes)),
        csv_read_file(FileEdges, [_|E], []),
        rows_to_lists(E, Edges),
        assertz(edges(Edges)).

rows_to_lists(Rows, Lists):-
        maplist(row_to_list, Rows, Lists).

row_to_list(Row, List):-
        Row =.. [row|List].

%Starting from SourceNodes, execute 1 iterations of the algorithm.
%F is firing treshold parameter, Result the resulting activation table.
spreading_activation(SourceNodes,F,1,Result):-
        nodes(Nodes),
        activation_tab(Nodes,ActTab),
        spreading_activation_iter(F,SourceNodes,1,ActTab,Result),
        !.

%Starting from SourceNodes, execute NIteration iterations of the algorithm.
%F is firing treshold parameter, Result the resulting activation table.
spreading_activation(SourceNodes,F,NIteration,Result):-
        N is NIteration-1,
        spreading_activation(SourceNodes,F,N,R),
        spreading_activation_iter(F,SourceNodes,0,R,Result).

%The same as above, but with an input activation table.
spreading_activation(SourceNodes,F,1,Input,Result):-
        spreading_activation_iter(F,SourceNodes,1,Input,Result),
        !.

spreading_activation(SourceNodes,F,NIteration,Input,Result):-
        N is NIteration-1,
        spreading_activation(SourceNodes,F,N,Input,R),
```

```
        spreading_activation_iter(F,SourceNodes,1,R,Result).

%The iteration of the algorithm.
spreading_activation_iter(F,SNodes,0,Tab,Res1):-
        get_firing_nodes(Tab,F,FNodes),
        check_spreading_edges(SNodes,FNodes,Tab,NewTab),
        set_fired(FNodes,NewTab,Res1),
        !.

spreading_activation_iter(F,SNodes,1,ActTab,Res1):-
        set_activation(SNodes,1,ActTab,Tab),
        get_firing_nodes(Tab,F,FNodes),
        check_spreading_edges(SNodes,FNodes,Tab,NewTab),
        set_fired(FNodes,NewTab,Res1).

%Get the list of firing nodes (not already fired
%and with activation greater than F, the firing treshold).
get_firing_nodes([],_,[]):-!.
get_firing_nodes([H|T],F,Res):-
        check_node(H,F,N),
        get_firing_nodes(T,F,R1),
        put(R1,N,Res).

check_node((_,_,1),_,[]):-!.
check_node((Node,Activation,0),F,Res):-
        Activation >= F
        -> Res=Node
        ; Res=[].

%Check edges starting from firing nodes.
%Change activation value of target unfired nodes.
%For each edges (X,Y) or (Y,X), if Y is not fired,
%Y's activation = Y's activation + X's activation * I smart factor.
check_spreading_edges(_,[],In,In):-!.
check_spreading_edges(SN,[H|T],In,Out):-
        nodes(NL),
        edges(EL),
        adjacents(H,EL,Adj),
        activate_adjacents(SN,H,Adj,NL,EL,In,Out1),
        check_spreading_edges(SN,T,Out1,Out).
```

```
activate_adjacents(_,_,[],_,_,In,In):-!.
activate_adjacents(SN,N,[H|T],NL,EL,In,Out):-
        get_fired(H,In,F),
        F == 0
        ->
        get_activation(H,In,ActH),
        get_activation(N,In,ActN),
        %Computing the I factor
        compute_I_factor(SN,N,H,NL,EL,(I,L)),
        (L == 'weight'
        ->
        Value is I,
        round(Value,3,V)
        ;
        Value is ActH + (ActN * I),
        round(Value,3,V)),
        ((V > 1 ; V < 0)
        ->
        normalize(V,V1)
        ;
        V1 is V),
        set_activation([H],V1,In,Out1),
        activate_adjacents(SN,N,T,NL,EL,Out1,Out)
        ;
        activate_adjacents(SN,N,T,NL,EL,In,Out).

%Create the activation list. Elements are in the form (Node,Value,Fired),
%where Node is a node of the graph, Value is its activation value,
%and Fired is 1 if the node is activated,
%or 0 if the node isn't already activated.
activation_tab([],[]).
activation_tab([[H|_]|T],[(H,0,0)|T1]):-
        activation_tab(T,T1).

%Update Input list, setting activation of nodes to Value,
%returns updated list Res.
set_activation([],_,Input,Input).
set_activation([H|T],Value,Input,Res):-
        updateAct(H,Value,Input,R1),
        set_activation(T,Value,R1,Res).
```

```
updateAct(_,_,[],[]):-!.
updateAct(Node,Value,[(Node,_,F)|T],[(Node,Value,F)|T]):-
        !.
updateAct(Node,Value,[H|T],[H|T1]):-
        updateAct(Node,Value,T,T1).

%Update Input list, setting Fired value to 1, returns updated list Res.
set_fired([],Input,Input).
set_fired([H|T],Input,Res):-
        updateFir(H,Input,R1),
        set_fired(T,R1,Res).

updateFir(_,[],[]):-!.
updateFir(Node,[(Node,A,_)|T],[(Node,A,1)|T]):-
        !.
updateFir(Node,[H|T],[H|T1]):-
        updateFir(Node,T,T1).

%Set value > 1 to 1, and value < 0 to 0.
normalize(V,1):-
        V > 1,
        !.
normalize(V,0):-
        V < 0.

%Get the activation tab sorted by activation value.
ranking(ActTab,Sorted):-
        sort(2,@>=,ActTab,Sorted).

%Get the activation only of nodes of type Type.
activation_type([],_,[]).
activation_type([(N,P,_)|T],Type,[(N,P)|T1]):-
        nodes(NL),
        get_type(N,NL,Type),
        !,
        activation_type(T,Type,T1).
activation_type([_|T],Type,T1):-
        activation_type(T,Type,T1).

%Get the activation value of a node.
get_activation(N,[_|T],A):-
```

```
        get_activation(N,T,A),
        !.
get_activation(N,[(N,A,_)|_],A).

%Get the fired status of a node.
get_fired(N,[_|T],F):-
        get_fired(N,T,F),
        !.
get_fired(N,[(N,_,F)|_],F).

%Get the type of a node.
get_type(Node,[[Node,Type|_]|_],Type):-!.
get_type(Node,[_|T],Type):-
        get_type(Node,T,Type),
        !.

%Get the label of an edge.
get_label(N1,N2,[[N1,N2,L|_]|_],L):-!.
get_label(N1,N2,[[N2,N1,L|_]|_],L):-!.
get_label(N1,N2,[_|T],L):-
        get_label(N1,N2,T,L).

%Get adjacents of a node.
adjacents(_,[],[]):-!.
adjacents(N,[H|T],[Y|T1]):-
        contains(N,H,Y),
        !,
        adjacents(N,T,T1).
adjacents(N,[_|T],T1):-
        adjacents(N,T,T1).

%Check if edge contains node X, return the adjacent node.
contains(X,[X,Y|_],Y).
contains(X,[Y,X|_],Y):-!.

%Put an element into a list, only if element is not [].
put(L,[],L):-!.
put([],E,[E]):-!.
put([H|T],E,[L1,H|L]):-
        put(T,E,[L1|L]).
```

```
%Round X float number to D decimal digit.
round(X,D,Res):-
        Y is X * 10^D,
        round(Y, Z),
        Res is Z/10^D.

%Get the Nth element of a list. First element of the list is at 1.
get([H|_],1,H):-!.
get([_|T],N,R):-
        N1 is N-1,
        get(T,N1,R).

%Compute the I factor.
compute_I_factor([],_,_,_,_,(0,_)):-!.
compute_I_factor([H|T],N1,N2,NL,EL,(I,Res)):-
        get_label(N1,N2,EL,Label),
        ((ignore(N1,Label,N2) ; ignore(N2,Label,N1))
        ->
        I is 0,
        Res = 'ignore'
        ;
        (weight(N1,Label,N2,W) ; weight(N2,Label,N1,W))
        ->
        I is W,
        Res = 'weight'
        ;
        findall(Int,interest(H,Int),L),
        get_type(N2,NL,Type),
        get_interest_type(L,Type,NL,L1),
        compute_I_interest(H,L1,N2,N1,NL,EL,I1),
        compute_I_factor(T,N1,N2,NL,EL,(I2,Res)),
        I is I1 + I2).

%Get interest list by Type.
get_interest_type([],_,_,[]).
get_interest_type([H|T],Type,NL,[H|T1]):-
        get_type(H,NL,Type),
        !,
get_interest_type(T,Type,NL,T1).
get_interest_type([_|T],Type,NL,T1):-
        get_interest_type(T,Type,NL,T1).
```

```
%Compute I by interest of SN.
compute_I_interest(_,[],_,_,_,_,0):-!.
compute_I_interest(SN,[H|T],N,N1,NL,EL,I):-
        compute_I_interest(SN,T,N,N1,NL,EL,I3),
        (interest(SN,N)
        ->
        I is 1
        ;
        n_columns(NCol),
        K is NCol-2,
        compute_I_attr(N,H,K,NL,I1),
        compute_I_adjs(N,H,EL,N1,I2),
        !,
        I is I1 + I2 + I3).

%Compute I comparing attributes of nodes.
compute_I_attr(_,_,0,_,0):-!.
compute_I_attr(N,Int,K,NL,I):-
        K1 is K-1,
        compute_I_attr(N,Int,K1,NL,I2),
        get_attr_node(N,K,NL,AttrN),
        get_attr_node(Int,K,NL,AttrInt),
        ((AttrN == AttrInt,AttrN \= '-')
        ->
        Count is 1
        ;
        Count is 0),
        I is Count*(0.1) + I2.

%Get the K'th attribute of Node.
get_attr_node(Node,K,[[Node,_|Attr]|_],El):-
        get(Attr,K,El),
        !.
get_attr_node(Node,K,[_|T],Attr):-
        get_attr_node(Node,K,T,Attr).

%Compute I comparing adjacents of nodes.
compute_I_adjs(N,H,EL,N1,I):-
        adjacents(N,EL,Adj1),
        adjacents(H,EL,Adj2),
```

```
        delete(Adj2,N1,AdjN),
        delete(Adj1,N1,AdjH),
        get_adjs_label(H,AdjN,EL,AdjLabelN),
        get_adjs_label(N,AdjH,EL,AdjLabelH),
        intersection(AdjLabelN,AdjLabelH,Ins),
        length(Ins,Count),
        I is Count*(0.25).

%Get the label of each adjacent edge.
get_adjs_label(_,[],_,[]):-!.
get_adjs_label(N,[H|T],EL,[Res|Tl]):-
        get_label(N,H,EL,L),
        atom_concat(H,L,Res),
        get_adjs_label(N,T,EL,Tl).
```

Il predicato `prepare_db\2` serve per la lettura dei due dataset contenenti i dati sui nodi e sugli archi.
Il predicato `spreading_activation\4` serve per eseguire l'algoritmo di spreading activation intelligente. Come parametri bisogna specificare:

- la lista di nodi da cui originare la diffusione dell'attivazione,
- la soglia di attivazione F,
- il numero di iterazioni per la diffusione dell'attivazione,
- la lista di output con le attivazioni per ogni nodo.

Seguono poi una serie di regole richiamate durante l'esecuzione dell'algoritmo.
Il predicato `ranking\2` ritorna la lista ordinata per attivazione crescente, mentre `activation_type\3` serve per filtrare i risultati sui nodi di uno specifico tipo.
`compute_I_factor\6`, infine, calcola il fattore intelligente I: l'utilizzo di questo fattore differenzia l'algoritmo sviluppato dal classico algoritmo di spreading activation.

Base di conoscenza All'interno della base di conoscenza Prolog dovranno essere presenti 3 tipologie di fatti. Questi fatti servono per definire l'interesse di determinate entità (nodi) nei confronti di altre entità, per stabilire il peso di alcuni archi all'interno della rete e per definire quali archi ignorare nell'esecuzione dell'algoritmo di spreading activation.
Per definire l'interesse dell'entità al nodo A nei confronti dell'entità al nodo B, scriveremo:

```
interest(A,B).
```

Se invece volessimo specificare che il peso dell'arco che va dal nodo A verso il nodo B con l'etichetta *lab* è pari a 1, scriveremo nella base di conoscenza:

```
weight(A,'lab',C,1).
```

Nel caso in cui ci fosse un arco da ignorare durante l'esecuzione dell'algoritmo, come ad esempio l'arco che va dal nodo A al nodo D con l'etichetta *lab*, scriveremo:

```
ignore(A,'lab',D).
```

Infine, è presente un fatto `n_columns(N)` per indicare il numero di colonne del documento contenente la lista dei nodi, dove N è appunto il numero di colonne del file.

Esempio Si consideri la rete semantica in figura 1.2, che rappresenta la visita del turista Marco a Firenze. Marco visiterà le Gallerie degli Uffizi, situate a Firenze, dato che è interessato ad un'opera esposta nel museo, la Primavera di Sandro Botticelli. Vogliamo, perciò, conoscere in che misura può essere interessato alle altre opere presenti nel museo, sapendo che ha mostrato interesse per la Primavera.

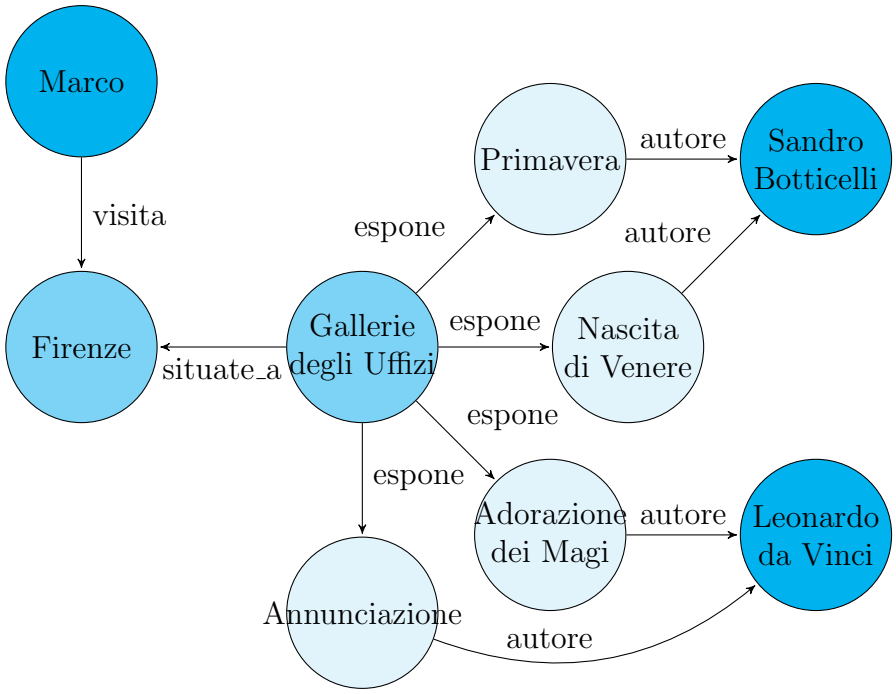

Figura 1.2: Esempio di rete semantica

Le informazioni riguardo le quattro opere inserite nella rete sono le seguenti:

Opera	**Genere**	**Periodo**
Adorazione dei Magi	Arte cristiana, Pittura storica	Tardo gotico
Annunciazione	Arte cristiana, Pittura storica	Rinascimento, Alto Rinascimento
Nascita di Venere	Pittura storica	Rinascimento
Primavera	Pittura storica	Rinascimento

La base di conoscenza scritta in Prolog è composta dai seguenti fatti:

```
interest('Marco','Primavera').
weight('Marco','visita','Firenze',1).
weight('Gallerie degli Uffizi','situato_a','Firenze',1).
weight('Primavera','autore','Sandro Botticelli',1).
weight('Nascita di Venere','autore','Sandro Botticelli',1).
weight('Adorazione dei Magi','autore','Leonardo da Vinci',1).
weight('Annunciazione','autore','Leonardo da Vinci',1).
ignore('Gallerie degli Uffizi','espone','Annunciazione').
```

L'esecuzione dell'algoritmo, partendo dal nodo *Marco*, dopo 3 iterazioni e con parametro F uguale a 1, riporta i seguenti risultati, filtrati per i soli nodi di tipo "opera":

Opera	**Attivazione\Interesse**
Primavera	1
Nascita di Venere	0.45
Adorazione dei Magi	0.1
Annunciazione	0

L'attivazione del nodo *Primavera* è pari a 1, dato che Marco è interessato a quest'opera. L'attivazione del nodo *Nascita di Venere* è pari a 0.45, poiché condivide due attributi e una coppia arco-nodo (*autore-Sandro Botticelli*) con l'opera Primavera, che è di interesse per Marco. Al contrario, per il nodo *Adorazione dei Magi* l'attivazione è uguale a 0.1, visto che condivide un solo attributo con l'opera Primavera. Infine, l'attivazione del nodo *Annunciazione* è 0 perché nella base di conoscenza abbiamo specificato di ignorare, durante l'esecuzione dell'algoritmo, l'arco che va da *Gallerie degli Uffizi* verso *Annunciazione* con l'etichetta *espone*.

Sperimentazione

2.1 Aggiornamento della rete Gr@phBRAIN

All'interno della rete di conoscenza Gr@phBRAIN sono state inserite informazioni attinenti al dominio del turismo.
Si è scelto di aggiungere nel database dati su luoghi d'interesse turistico presenti nella città di Bari, ma, soprattutto, informazioni dettagliate riguardo una collezione di quadri esposta presso la Pinacoteca "Corrado Giaquinto" di Bari. La collezione scelta è stata la collezione Grieco composta da 49 quadri, per la maggior parte di artisti Macchiaioli. I dati sulle opere sono stati estratti direttamente dal sito web ufficiale della Pinacoteca[1].
I luoghi di interesse turistico sono stati trattati come entità di tipo *PointOfInterest* nella rete di Gr@phBRAIN. La collezione è stata inserita come entità di tipo *Collection*, mentre i quadri come entità di tipo *Attraction*. Ogni quadro è stato collegato al suo autore, entità di tipo *Person*, ed eventualmente alla corrente artistica in cui è inquadrabile, entità di tipo *Category*.
Nelle tabelle 2.1, 2.2, 2.3, 2.4, 2.5 sono quindi elencate le informazioni inserite all'interno della rete Gr@phBRAIN, suddivise per tipo di entità, con indicazione dell'identificativo attribuito ad ogni entità:

ID	**Name**	**Kind**
810	Basilica di San Nicola	Church
812	Castello Normanno-Svevo	Monument
811	Cattedrale di San Sabino	Church
816	Museo Archeologico di Santa Scolastica	Museum
817	Museo Nicolaiano	Museum
791	Pinacoteca Corrado Giaquinto	Museum
815	Teatro Margherita	Theater
813	Teatro Petruzzelli	Theater

[1] http://www.pinacotecabari.it/

814	Teatro Piccinni	Theater

Tabella 2.1: Entità di tipo *PointOfInterest*

ID	**Name**	**Type**
804	Collezione Grieco	Series

Tabella 2.2: Entità di tipo *Collection*

ID	**Name**	**Type**	**Date**	**Technique**
801	Al pianoforte	Painting	1867	oil on paper
830	Alberi o Bosco alle Cascine	Painting	1870 ca.	oil on panel
843	Angolo di giardino (con sfondo di paesaggio)	Painting	1912	oil on canvas
793	Angolo veneziano	Painting	1869	oil on panel
855	Antica Via del Fuoco (Mercato Vecchio)	Painting	1881	oil on canvas
825	Barconi in laguna	Painting	1890	oil on panel
866	Burano	Painting	1945	oil on canvas
862	Case nel bosco	Painting	1931	oil on canvas applied on cardboard
865	Cavalli	Painting	1927-1935 ca.	gouache on paper pasted on cardboard
846	Cavalli al pascolo	Painting	1865 ca.	oil on canvas
832	Contadina senese che fa l'erba	Painting	1885-1890	oil on canvas
796	Contadinella al sole	Painting	1859-1865	oil on panel
857	Donna che cuce	Painting	1875 ca.	oil on canvas
860	Donne al mare	Painting	1934	oil on canvas
797	Donne sulla terrazza	Painting	1860-1862 ca.	oil on panel
859	Fanciulla con un fascio di fiori	Painting	1903	oil on canvas
845	Giovinetta	Painting	1880-1890	oil on panel
837	Il lavoro della terra	Painting	1867-1873	oil on cardboard
874	Il sonaglio	Painting	1912	oil on canvas
841	In villeggiatura	Painting	1885 ca.	oil on panel
840	L'Arno a Rovezzano	Painting	1880 ca.	oil on panel
829	L'Arno a San Rossore	Painting	1860-1864 ca.	oil on panel

795	L'Arno alla Casaccia	Painting	1862-1863 ca.	oil on canvas
827	La Porta Saint-Denis a Parigi	Painting	1875 ca.	oil on cardboard
838	La lettura	Painting	1864-1867 ca.	oil on pasteboard applied to panel
835	La maestrina	Painting	1876	oil on cardboard
828	La pesca sul fiume	Painting	post 1855-1856	oil on panel
852	La raccolta delle olive	Painting	1862-1865 ca.	oil on canvas
850	Marina di Castiglioncello	Painting	1864	oil on canvas applied on panel
873	Montagne	Painting	1940-1945 ca.	oil on canvas
868	Natura morta	Painting	1945	oil on panel
869	Paesaggio	Painting	1942	oil on canvas
871	Paesaggio	Painting	1927	oil on canvas
875	Paesaggio apuano	Painting	1820-1823	oil on cardboard
849	Paesino o Strada di campagna	Painting	1862-1863	oil on panel
798	Passeggiata sotto la pioggia	Painting	1880 ca.	oil on panel
805	Porta a San Frediano	Painting	1862-1863 ca.	oil on panel
803	Porta fiorentina (Antica Porta alla Croce)	Painting	post 1877	oil on panel
864	Ragazza sulla poltrona	Painting	1939-1941 ca.	oil on cardboard applied on panel
833	Ritorno della cavalleria	Painting	1888	oil on canvas
824	Ritratto d'uomo	Painting	ante 1920	oil on canvas
799	Ritratto del pittore Lanfredini	Painting	1866	oil on panel
858	Ritratto della figlia Emma	Painting	1882-1883	oil on canvas
839	San Prugnano	Painting	1875 ca.	oil on panel
842	Servetta	Painting	1885-1890	oil on canvas
853	Strada a Ravenna	Painting	1875	oil on plastered cardboard
847	Una via a Volpedo 1903	Painting	1903	oil on canvas
792	Venezia	Painting	1867	oil on cardboard

831	Viale alle Cascine	Painting	1875-1880	oil on panel

Tabella 2.3: Entità di tipo *Attraction*

ID	**Surname**	**Name**	**KnownAs**	**Gen.**	**BirthDate**	**DeathDate**
876	Abbati	Giuseppe	-	M	1836/01/13	1868/02/21
782	Banti	Cristiano	-	M	1824/01/04	1904/12/04
904	Boldini	Giovanni	-	M	1824/12/31	1931/01/11
909	Borrani	Odoardo	-	M	1833/08/22	1905/09/14
916	Cabianca	Vincenzo	-	M	1827/06/20	1902/03/22
923	Cammarano	Michele	-	M	1835/02/23	1920/09/21
930	Ciardi	Guglielmo	-	M	1842/09/13	1917/10/05
937	De Nittis	Giuseppe	-	M	1846/02/25	1884/08/21
944	De Tivoli	Serafino	-	M	1825/02/22	1892/11/01
951	Fattori	Giovanni	-	M	1825/09/06	1908/08/30
957	Favretto	Giacomo	-	M	1849/08/11	1887/06/12
964	Fontanesi	Antonio	-	M	1818/02/23	1882/04/17
971	Lega	Silvestro	-	M	1826/12/08	1895/11/21
978	Mancini	Antonio	-	M	1852/11/14	1930/12/28
985	Morbelli	Angelo	-	M	1854/07/18	1919/11/07
992	Nono	Luigi	-	M	1850/12/08	1918/10/17
999	Palizzi	Filippo	-	M	1818/06/16	1899/09/11
1005	Pellizza	Giuseppe	Pellizza da Volpedo	M	1868/07/28	1907/06/14
1011	Sernesi	Raffaello	-	M	1838/12/29	1866/08/11
1017	Signorini	Telemaco	-	M	1835/08/18	1901/02/10
1023	Toma	Gioacchino	-	M	1836/01/24	1891/01/12
1028	Zandomeneghi	Federico	-	M	1841/06/02	1917/12/31
1034	Ihlenfeldt	Max	Massimo Campigli	M	1895/07/04	1971/05/31
1040	Carrà	Carlo Dalmazio	Carlo Carrà	M	1881/02/11	1966/04/13
1046	Casorati	Felice	-	M	1883/12/04	1963/03/01
1053	de Chirico	Giorgio	-	M	1888/07/10	1978/11/20
1058	Tibertelli de Pisis	Luigi Filippo	Luigi de Pisis	M	1896/05/11	1956/04/02
1065	Mafai Volpe	Mario	-	M	1902/02/12	1965/03/31
1071	Morandi	Giorgio	-	M	1890/07/20	1964/06/18
1077	Rosai	Ottone	-	M	1895/04/28	1957/05/13

1083	Sironi	Mario	-	M	1885/05/12	1961/08/13
1090	Spadini	Armando	-	M	1883/07/29	1925/03/31
1096	Viani	Lorenzo	-	M	1882/11/01	1936/11/02

Tabella 2.4: Entità di tipo *Person*

ID	**Name**	**Type**
807	Divisionismo	Trend
808	Impressionismo	Trend
809	Naturalismo	Trend
806	Pittura dei Macchiaioli	Trend

Tabella 2.5: Entità di tipo *Category*

Per quanto riguarda le relazioni inserite nel grafo, i luoghi di interesse sono collegati all'entità che rappresenta la città di Bari tramite relazioni di tipo *wasIn*. Lo stesso tipo di relazione collega la collezione Grieco alla Pinacoteca Corrado Giaquinto. Tutti i dipinti sono connessi alla collezione attraverso relazioni di tipo *belongsTo*. Per quanto concerne gli autori dei dipinti, questi sono collegati alle loro opere tramite connessioni del tipo *developed*. Infine, per i quadri che sono stati inquadrati in delle correnti artistiche, esistono delle relazioni di tipo *belongsTo* che collegano le opere alle correnti artistiche.
Nella tabella 2.6 è riportato l'elenco dei dipinti appartenenti alla collezione Grieco, con indicazione dell'autore e dell'eventuale corrente artistica del quadro. Come si può notare, la maggior parte dei dipinti è ascrivibile alla pittura dei Macchiaioli: la collezione, infatti, comprende quasi tutti i macchiaioli toscani, alcuni artisti veneti e napoletani dell'Ottocento e un'ottima scelta di grandi artisti del primo Novecento.

Opera	**Autore**	**Corrente artistica**
Porta a San Frediano	Giuseppe Abbati	Pittura dei Macchiaioli
L'Arno alla Casaccia	Giuseppe Abbati	Pittura dei Macchiaioli
Contadinella al sole	Giuseppe Abbati	Pittura dei Macchiaioli
Donne sulla terrazza	Cristiano Banti	Pittura dei Macchiaioli
Passeggiata sotto la pioggia	Cristiano Banti	Pittura dei Macchiaioli
Ritratto del pittore Lanfredini	Giovanni Boldini	Pittura dei Macchiaioli
Al pianoforte	Giovanni Boldini	Pittura dei Macchiaioli
Porta fiorentina (Antica Porta alla Croce)	Odoardo Borrani	Pittura dei Macchiaioli
Venezia	Vincenzo Cabianca	Pittura dei Macchiaioli

Angolo veneziano	Vincenzo Cabianca	Pittura dei Macchiaioli
Ritratto d'uomo	Michele Cammarano	-
Barconi in laguna	Guglielmo Ciardi	Pittura dei Macchiaioli
La Porte Saint-Denis a Parigi	Giuseppe De Nittis	Pittura dei Macchiaioli Impressionismo
La pesca sul fiume	Serafino De Tivoli	Pittura dei Macchiaioli
L'Arno a San Rossore	Serafino De Tivoli	Pittura dei Macchiaioli
Alberi o Bosco alle Cascine	Giovanni Fattori	Pittura dei Macchiaioli
Viale alle Cascine	Giovanni Fattori	Pittura dei Macchiaioli
Contadina senese che fa l'erba	Giovanni Fattori	Pittura dei Macchiaioli
Ritorno della cavalleria	Giovanni Fattori	Pittura dei Macchiaioli
La maestrina	Giacomo Favretto	-
Il lavoro della terra	Antonio Fontanesi	Pittura dei Macchiaioli
La lettura	Silvestro Lega	Pittura dei Macchiaioli
San Prugnano	Silvestro Lega	Pittura dei Macchiaioli
L'Arno a Rovezzano	Silvestro Lega	Pittura dei Macchiaioli
In villeggiatura	Silvestro Lega	Pittura dei Macchiaioli
Servetta	Antonio Mancini	Naturalismo
Angolo di giardino (con sfondo di paesaggio)	Angelo Morbelli	Divisionismo
Giovinetta	Luigi Nono	-
Cavalli al pascolo	Filippo Palizzi	-
Una via a Volpedo, 1903	Giuseppe Pellizza Da Volpedo	Divisionismo
Paesino o Strada di campagna	Raffaello Sernesi	Pittura dei Macchiaioli
Marina di Castiglioncello	Raffaello Sernesi	Pittura dei Macchiaioli
La raccolta delle olive	Telemaco Signorini	Pittura dei Macchiaioli
Strada a Ravenna	Telemaco Signorini	Pittura dei Macchiaioli
Antica Via del Fuoco (Mercato Vecchio)	Telemaco Signorini	Pittura dei Macchiaioli
Donna che cuce	Gioacchino Toma	-
Ritratto della figlia Emma	Gioacchino Toma	-
Fanciulla con un fascio di fiori	Federico Zandomeneghi	Impressionismo
Donne al mare	Massimo Campigli	-
Case nel bosco	Carlo Carrà	-
Ragazza sulla poltrona	Felice Casorati	-
Cavalli	Giorgio De Chirico	-
Burano	Filippo De Pisis	-

Natura morta	Mario Mafai	-
Paesaggio	Giorgio Morandi	-
Paesaggio	Ottone Rosai	-
Montagne	Mario Sironi	-
Il sonaglio	Armando Spadini	-
Paesaggio apuano	Lorenzo Viani	-

Tabella 2.6: Autori e correnti artistiche dei quadri della collezione Grieco

2.2 Esempio di applicazione dell'algoritmo di Spreading activation intelligente

Vediamo ora una possibile applicazione dell'algoritmo di spreading activation intelligente sul grafo di Gr@phBRAIN nel dominio del turismo.
Immaginiamo che ci sia un turista in visita a Bari, amante dell'arte, che vuol visitare la pinacoteca presente in città. Egli è interessato ad una particolare opera esposta nella pinacoteca. Si vuol capire quale potrebbe essere il suo interesse verso gli altri dipinti presenti nel museo. L'algoritmo di spreading activation intelligente può aiutarci in questo senso.
Per lo scopo del seguente esempio e delle successive sperimentazioni, si è inserito nel grafo di conoscenza un'entità di tipo *Person* per rappresentare il turista:

ID	**Surname**	**Name**	**Gen.**
1102	Maiorano	Federico	M

Questa entità rappresentante il turista è connessa solo con l'entità della città di Bari tramite una relazione di tipo *wasIn*, che indica la visita del turista nella città.
Dopo aver scaricato il database di Gr@phBRAIN, ed aver eliminato tutte le entità e le relazioni non attinenti al dominio del turismo [2], si sono preparati i dataset da fornire in input allo script Prolog per la spreading activation intelligente.
Sappiamo che il turista è interessato al dipinto "Viale alle Cascine" (831) realizzato da Giovanni Fattori. Inoltre, fissiamo a 1 il peso degli archi che collegano il turista (1102) alla città di Bari (6), la pinacoteca (791) a Bari, e

[2]Sono state mantenute solo le entità del dominio *tourism* di tipo *Attraction*, *Category*, *Collection*, *Event*, *Multimedia*, *Period*, *Person*, *Place*, *PointOfInterest*, *Visit*

la collezione Grieco (804) alla pinacoteca. Tra parentesi sono indicati gli id delle varie entità, dato che questi serviranno per fare riferimento alle stesse entità nella base di conoscenza.
Nella base di conoscenza Prolog vengono inseriti quindi i seguenti fatti:

```
interest(1102,831).
weight(1102,'wasIn',6,1).
weight(791,'wasIn',6,1).
weight(804,'wasIn',791,1).
ignore('','','').
```

Il nodo di origine per la spreading activation deve essere il nodo che indica il turista (1102), mentre, per poter raggiungere i nodi dei dipinti bisogna effettuare 4 iterazioni nel ciclo della spreading activation. Viene quindi eseguita in Prolog la seguente regola, con F soglia di attivazione fissata a 0.7:

```
spreading_activation([1102],0.7,4,Result)
```

I risultati dell'esecuzione dell'algoritmo, filtrati per le sole entità di tipo *Attraction* e ordinati per attivazione crescente, sono i seguenti:

ID	**Opera**	**Attivazione**
831	Viale alle Cascine	1
830	Alberi o Bosco alle Cascine	0.7
832	Contadina senese che fa l'erba	0.6
833	Ritorno della cavalleria	0.6
793	Angolo veneziano	0.45
796	Contadinella al sole	0.45
797	Donne sulla terrazza	0.45
798	Passeggiata sotto la pioggia	0.45
799	Ritratto del pittore Lanfredini	0.45
803	Porta fiorentina (Antica Porta alla Croce)	0.45
805	Porta a San Frediano	0.45
825	Barconi in laguna	0.45
828	La pesca sul fiume	0.45
829	L'Arno a San Rossore	0.45
839	San Prugnano	0.45
840	L'Arno a Rovezzano	0.45
841	In villeggiatura	0.45
849	Paesino o Strada di campagna	0.45
792	Venezia	0.35
795	L'Arno alla Casaccia	0.35

801	Al pianoforte	0.35
827	La Porta Saint-Denis a Parigi	0.35
837	Il lavoro della terra	0.35
838	La lettura	0.35
850	Marina di Castiglioncello	0.35
852	La raccolta delle olive	0.35
853	Strada a Ravenna	0.35
855	Antica Via del Fuoco (Mercato Vecchio)	0.35
845	Giovinetta	0.2
868	Natura morta	0.2
824	Ritratto d'uomo	0.1
835	La maestrina	0.1
842	Servetta	0.1
843	Angolo di giardino (con sfondo di paesaggio)	0.1
846	Cavalli al pascolo	0.1
847	Una via a Volpedo 1903	0.1
857	Donna che cuce	0.1
858	Ritratto della figlia Emma	0.1
859	Fanciulla con un fascio di fiori	0.1
860	Donne al mare	0.1
862	Case nel bosco	0.1
864	Ragazza sulla poltrona	0.1
865	Cavalli	0.1
866	Burano	0.1
869	Paesaggio	0.1
871	Paesaggio	0.1
873	Montagne	0.1
874	Il sonaglio	0.1
875	Paesaggio apuano	0.1

Possiamo vedere come i dipinti con maggiore attivazione, cioè quelli verso cui il turista può avere un certo interesse, sono gli altri lavori di Giovanni Fattori (escludendo il dipinto "Viale alle Cascine", la cui attivazione è fissata a 1). I quadri "Alberi o Bosco alle Cascine", "Contadina senese che fa l'erba" e "Ritorno della cavalleria", riportano rispettivamente i valori di attivazione 0.7, 0.6 e 0.6. Questi risultati sono dovuti al fatto che queste opere condividono con il dipinto "Viale alle Cascine" l'autore, il tipo di opera e la corrente artistica ($0.25 + 0.1 + 0.25 = 0.6$). Il dipinto "Alberi o Bosco alle Cascine" riporta un valore leggermente maggiore poiché è stato eseguito con la stessa

tecnica pittorica ($0.1 + 0.6 = 0.7$).
Diversi quadri riportano un valore di 0.45: queste opere appartengono tutte alla pittura dei Macchiaioli, che è la stessa corrente artistica in cui è stato inserito il dipinto "Viale alle Cascine", e sono state eseguite con la stessa tecnica. Altri lavori mostrano un valore di attivazione pari a 0.35: sono altri dipinti appartenenti alla pittura dei Macchiaioli, ma realizzati con tecniche differenti.
I due quadri "Giovinetta" e "Natura morta" condividono con "Viale alle Cascine" il tipo di opera e la tecnica di realizzazione ($0.1 + 0.1 = 0.2$). Infine, le rimanenti opere presentano un valore di attivazione uguale a 0.1, dato che hanno in comune con il quadro di interesse per il turista solo il tipo di opera.

2.3 Confronto tra Spreading activation tradizionale e intelligente

Si vuole ora confrontare l'algoritmo sviluppato con un classico algoritmo di spreading activation, come quello presentato nel capitolo 1 nella sezione 1.1. Si è creato un altro programma Prolog per l'esecuzione dell'algoritmo di spreading activation tradizionale. Ad entrambi i programmi sono stati forniti gli stessi dataset di input. Per il programma Prolog di spreading activation intelligente, la base di conoscenza è identica a quella dell'esempio presentato precedentemente.
Il primo confronto realizzato è in termini di tempo d'esecuzione dei programmi. Per valutare il tempo di esecuzione e il numero di inferenze effettuate da una regola si è utilizzato il predicato integrato `time\1`, che in input necessita del goal da valutare.
Le regole da valutare sono le seguenti:

- `spreading_activation(SN,F,D,C,Result)` per la spreading activation tradizionale;
- `spreading_activation(SN,F,C,Result)` per la spreading activation intelligente;

dove SN equivale alla lista di nodi d'origine, F alla soglia d'attivazione, D al fattore di decadimento, C al numero di iterazioni della diffusione d'attivazione e $Result$ alla lista risultato di output.

SN equivale a `[1102]` come nel esempio precedente. Mentre F è fissato a 0.1 e il fattore di decadimento D a 0.9. Il parametro C verrà invece aumentato

nelle diverse esecuzioni, per vedere come influisce sul tempo d'esecuzione. I risultati sono mostrati nella tabella 2.8

C	**Spread. Act. tradizionale**		**Spread. Act. intelligente**	
	tempo	**inferenze**	**tempo**	**inferenze**
1	0.008 sec.	18 913	0.013 sec.	17 660
2	0.014 sec.	32 154	0.031 sec.	44 578
3	0.030 sec.	59 113	0.013 sec.	40 952
4	0.047 sec.	168 968	0.181 sec.	1 404 956
5	0.057 sec.	436 753	0.227 sec.	1 773 017
6	0.075 sec.	533 784	0.216 sec.	1 775 311
7	0.071 sec.	536 330	0.217 sec.	1 777 605
8	0.105 sec.	538 876	0.220 sec.	1 779 899
9	0.093 sec.	541 422	0.237 sec.	1 782 193
10	0.097 sec.	543 968	0.239 sec.	1 784 487

Tabella 2.8: Tempi di esecuzione e numero di inferenze al variare del numero di iterazioni

Si può notare dai risultati come, con entrambi gli algoritmi, i tempi di esecuzione e le inferenze aumentino con il crescere del fattore C.
Per quanto riguardo lo spreading activation tradizionale, i tempi di esecuzione aumentano costantemente fino a $C = 6$, mentre non c'è aumento passando da 6 a 7; per C pari a 8,9 e 10, il tempo è praticamente lo stesso. Andando a considerare il numero di inferenze, si nota come cresca con l'incremento di C: l'aumento maggiore si ha passando da $C = 4$ a $C = 5$, mentre dopo il valore 5 la crescita del numero di inferenze è meno accentuata. Il notevole aumento ottenuto cambiando il valore di C da 4 a 5, può essere dovuto al fatto che in tale modo si raggiungano molti nodi attivabili per la diffusione dell'attivazione.
Guardando i risultati ottenuti nelle esecuzioni dell'algoritmo di spreading activation intelligente, si nota come i tempi di esecuzione sono molto bassi fino a $C = 3$; a partire da $C = 4$ i tempi aumentano, rimanendo pressoché costanti fino a $C = 10$. Lo stesso scarto visto nei tempi tra $C = 3$ e $C = 4$, si osserva nel numero di inferenze: passando con C da 3 a 4, le inferenze da essere 40 952 diventano 1 404 956. Con 4 iterazioni del ciclo di diffusione dell'attivazione, infatti, riusciamo a raggiungere i nodi che rappresentano i 49 dipinti esposti nella pinacoteca di Bari, nodi di cui si vuol conoscere l'attivazione (intesa come interesse) e per i quali è stata approntata la base di

conoscenza.

In un secondo esperimento il parametro cambiato nelle varie esecuzioni è stato F, la soglia di attivazione oltre la quale un nodo viene attivato, cosicché gli archi a esso connessi vengano esplorati. I fattori SN e D sono identici a quelli del primo esperimento, mentre il valore del parametro C è fissato a 4, in modo da raggiungere, con l'algoritmo di spreading activation intelligente, i nodi rappresentanti i quadri della pinacoteca. I risultati di queste prove sono riassunti nella tabella 2.9.

F	**Spread. Act. tradizionale**	**Spread. Act. intelligente**
	numero di nodi	**numero di nodi**
0.1	73	53
0.2	73	53
0.3	73	53
0.4	73	53
0.5	73	53
0.6	73	53
0.7	73	53
0.8	16	53
0.9	14	53
1	2	53

Tabella 2.9: Numero di nodi con attivazione maggiore di 0 al variare della soglia di attivazione

Per quanto concerne l'algoritmo di spreading activation intelligente, il numero di nodi con attivazione maggiore di 0 è pari a 53 per qualsiasi valore di F: 53 è infatti il risultato della somma di 49 nodi dei dipinti (nessuno riporta attivazione pari a 0) con i nodi che rappresentano il turista, la città di Bari, la pinacoteca e la collezione Grieco. Se andassimo a diminuire il valore di C, che era stato posto uguale a 4, non riusciremmo a diffondere l'attivazione verso i nodi che rappresentano i quadri; se invece andassimo ad aumentare il parametro C con valori maggiori di 4, otterremmo che il numero di nodi con attivazione maggiore di 0 sia sempre pari a 53, visto che, per come è impostata la base di conoscenza, la diffusione dell'attivazione non si propaga verso nodi che non siano di tipo *Attraction*.
Per l'algoritmo di spreading activation tradizionale il numero di nodi decresce con l'aumento del fattore F. Tuttavia, per F compreso tra 0.1 e 0.7, il

numero di nodi con attivazione maggiore di 0 è sempre pari a 73: questo numero rappresenta il limite massimo di nodi che si possono raggiungere con 4 cicli di iterazione di diffusione dell'attivazione partendo dal nodo di origine del turista. Aumentando il numero di iterazioni si riescono a raggiungere più nodi.

2.4 Confronto tra Spreading activation in Prolog e Neo4j

In un ultimo esperimento si è voluta confrontare l'implementazione dell'algoritmo di spreading activation tradizionale realizzata in Prolog con un'altra realizzazione dello stesso algoritmo, in termini di tempo d'esecuzione al variare del numero di iterazioni della spreading activation.
Si è scelto di implementare l'algoritmo su Neo4j, software specificatamente progettato per la manipolazione di grafi, usando per la definizione delle query il linguaggio Cypher. Per la realizzazione dell'algoritmo si è preso spunto dalla guida su GitHub *Spreading activation in Neo4j*[3], adattando l'algoritmo in modo che fosse identico a quello in Prolog. Ci si attende che l'algoritmo realizzato in Prolog sia più lento dello stesso lanciato in Neo4j, che è un tool creato proprio per la gestione di database a grafo.
Le query sono state poi eseguite sull'interfaccia browser di Neo4j, sulla quale è possibile misurare il tempo di esecuzione di ogni singola query. Il tempo totale di esecuzione dell'algoritmo è il risultato della somma dei tempi delle singole query.
Come parametri per il lancio degli algoritmo si è scelto F pari a 0.1, in modo da includere più nodi possibile, il fattore di decadimento fissato a 0.9, e l'insieme dei nodi da cui far propagare l'attivazione composto dal solo nodo turista dei precedenti esperimenti. Il peso degli archi è posto uguale a 1.

[3]https://gist.github.com/shprman/0e246f7082d5f06b00d2065df44a84bd

Cicli	Spreading Act. Prolog	Spreading Act. Neo4j
	tempo di esecuzione	tempo di esecuzione
1	0.010 sec.	0.012 sec.
2	0.010 sec.	0.016 sec.
3	0.028 sec.	0.020 sec.
4	0.047 sec.	0.026 sec.
5	0.083 sec.	0.032 sec.
6	0.096 sec.	0.032 sec.
7	0.096 sec.	0.032 sec.
8	0.096 sec.	0.032 sec.
9	0.097 sec.	0.032 sec.
10	0.073 sec.	0.032 sec.

Tabella 2.10: Tempi di esecuzione al variare del numero di iterazioni (sotto-grafo turismo)

Come si può vedere dalla tabella 2.10 i tempi di esecuzione delle due esecuzioni sono simili fin quando i cicli di spreading activation sono massimo 3. All'aumentare del numero di cicli si nota come in Neo4j il tempo di esecuzione rimanga costante, mentre tende a crescere in Prolog. Superati i 5 cicli di attivazione, infatti, non ci sono più nodi da attivare ed archi da esplorare: ciò si nota soprattutto in Neo4j con il tempo di esecuzione che rimane costante a 32 millisecondi, mentre sono poco più alti in Prolog.
I risultati sono comunque simili e i tempi sono sempre al di sotto del centesimo di secondo per entrambe le realizzazioni. Ciò può essere dovuto al fatto che si sta analizzando un grafo di ridotte dimensioni. Quindi nel prossimo esperimento si lanceranno gli algoritmi sull'intero grafo di Gr@phBRAIN (974 nodi e 1 351 archi), e non sul solo sotto-grafo relativo al dominio del turismo (480 nodi e 466 archi). I risultati sono riportati nella tabella 2.11.

Cicli	**Spreading Act. Prolog**	**Spreading Act. Neo4j**
	tempo di esecuzione	**tempo di esecuzione**
1	0.012 sec.	0.007 sec.
2	0.018 sec.	0.014 sec.
3	0.062 sec.	0.023 sec.
4	0.101 sec.	0.035 sec.
5	0.327 sec.	0.042 sec.
6	0.575 sec.	0.061 sec.
7	0.843 sec.	0.076 sec.
8	1.284 sec.	0.109 sec.
9	1.264 sec.	0.113 sec.
10	1.320 sec.	0.119 sec.

Tabella 2.11: Tempi di esecuzione al variare del numero di iterazioni

Andando ad analizzare il grafo completo di Gr@phBRAIN, si osserva una netta differenza in termini di tempo d'esecuzione tra l'algoritmo in Prolog e la realizzazione in Neo4j, con l'aumentare dei cicli di spreading activation. L'algoritmo scritto in Prolog, come ci si aspettava, è molto più lento della sua controparte lanciata su Neo4j, arrivando ad impiegare più di 1 secondo per completare l'esecuzione. In Neo4j, infatti, si rimane sempre entro il decimo di secondo, che si supera solo con un numero di iterazioni maggiore di 7.

Conclusioni

Con il seguente progetto si è cercato di creare un algoritmo per l'analisi di reti semantiche che sfrutti una forma di conoscenza che si ha sui dati. Si è partiti da un algoritmo di spreading activation, a cui è stata poi aggiunta una componente intelligente che calcolasse dinamicamente il peso degli archi in base alla conoscenza. Il peso degli archi indica il grado di interesse o affinità che l'insieme di nodi sorgente dell'algoritmo ha nei confronti del nodo verso cui l'attivazione viene diffusa.
L'algoritmo è stato sviluppato in Prolog, mentre per le sperimentazioni si è utilizzata la rete semantica Gr@phBRAIN, in cui sono state aggiunte informazioni riguardo il dominio del turismo. Nelle sperimentazioni si è confrontato l'algoritmo di spreading activation intelligente realizzato con l'implementazione tradizione, valutando tempi d'esecuzione, numero di inferenze e numero di nodi ottenuti con attivazione maggiore di zero. In un successivo esperimento si sono comparate le implementazioni in Prolog e Neo4j dell'algoritmo tradizionale di spreading activation, e si è constatato come la realizzazione in Prolog fosse più lenta di quella in Neo4j con il crescere della dimensionalità della rete e del numero di iterazioni nella diffusione dell'attivazione.

L'algoritmo implementato, comunque, è una prima possibile soluzione per l'obiettivo che si era posti. Tuttavia, la modalità di calcolo del fattore intelligente è migliorabile. Nella soluzione proposta, infatti, la similarità tra nodi viene stabilita considerando la semplice intersezione degli attributi. Si potrebbe pensare a soluzioni alternative meno restrittive. Considerando ad esempio le datazioni dei dipinti, queste sono uguali solo se l'anno di produzione è lo stesso. Potrebbe essere più utile, invece, considerare degli intervalli o dei periodi di tempo.
Inoltre, al momento, l'interesse delle entità verso altre è fissato ad 1. Si potrebbe immaginare però che l'interesse possa assumere valori variabili e non necessariamente uguali ad 1. Ad esempio, un turista potrebbe avere un

grado di interesse di 1 per un dipinto, mentre per una scultura un grado di interesse pari a 0.5. Questa situazione è più vicina a quella di contesti reali. L'algoritmo potrebbe essere modificato per considerare anche questa possibilità.