IA-Lab 2019/2020
Relazione Progetto Reti Bayesiane

Bushaj Aldo: 847091
Bushaj Antonino: 847013
Frisullo Silvestro Stefano: 832813

1. INTRODUZIONE

Per la prima parte del progetto è stata estesa la libreria aima-core tramite l'implementazione di strategie volte al miglioramento dell'efficienza computazionale della Variable Elimination, al fine di rendere migliori le prestazioni del sistema di calcolo dell'inferenza su reti Bayesiane. In particolare sono state applicate tre strategie di ottimizzazione:
-Pruning di nodi irrilevanti (Pruning semplice e M-Separation)
-Pruning di Archi irrilevanti (Pruning Edge)
-Implementazione di algoritmi di ordinamento dei fattori

Abbiamo suddiviso il progetto in parti diverse a seconda della consegna. Sotto la cartella reti Bayesiane troviamo tutte le classi principali:
La prima parte è "Pruning" che tratta l'esercizio del pruning dei nodi irrilevanti in cui si eliminano tutti i nodi non 'ancestor' di nodi rilevanti.
La seconda è "M-Separation" che riguarda l'eliminazione dei nodi m-separati, ossia i nodi che non sono raggiungibili partendo da una variabile di query senza passare dall evidenza.
La terza classe è "Pruning Edge" che riguarda l'eliminazione degli archi irrilevanti, il tutto senza ripercussioni nel risultato della query.
La quarta classe è "Sort " che tratta l'ordinamento topologico delle reti Bayesiane, tutti gli esercizi citati finora si avvalgono della classe "Sort" che si occupa di implementare l'ordinamento scelto, selezionabile dall'utente al momento dell'avvio.
Abbiamo implementato inoltre una classe "Utils" che contiene tutti i metodi utilizzati dalle diverse classi al fine di rendere il codice più fluido e meno ridondante.

2. SVOLGIMENTO

I vari esercizi fanno uso di metodi in comune:

- getQuery(String variables, List variablesList, List DomainList) : dati i nomi delle variabili, le variabili e la lista dei loro domini, questo metodo si occupa di interagire con l'utente per chiedere le variabili di query, quelle di evidenza con i relativi assegnamenti e il tipo di ordinamento desiderato. restituisce una Lista di Liste, in

posizione 0 c'è la lista di query variables, in posizione 1 le variabili di evidenza, in posizione 2 le AssignmentProposition e in posizione 3 il tipo di ordinamento scelto.

- *getVariableFromList(List<RandVar> variableList, String name):*
  restituisce una RandomVariable data una lista in cui è contenuta e il suo nome.
- *nodeToRandomVariableList(BayesianNetwork Network, List<Node> listNode):*
  converte lista di nodi in lista di RandomVariable
- *randomVariableToNodeList(BayesianNetwork Network, List<RandomVariable> listRand)*: converte lista di RandomVariable in lista di Nodi
  Node getNodeByName(String name): Nodo dato il suo nome
- *getNodeByName(Set<String> names)*: Ritorna il set di nodi dato il loro nome

2.1 Pruning

In questo esercizio dopo aver caricato la rete Bayesiana su cui andremo a lavorare eseguiamo il metodo getQuery e istanziamo tutte le variabili necessarie, calcoliamo gli ancestors delle variabili di query, eliminiamo dalla rete i nodi rimanenti, quindi cancelliamo tutti i riferimenti ai nodi rimossi nei nodi rimanenti (parents e children). Gli ancestors sono memorizzati nella lista di nodi Ancestors, mentre nella lista total memorizziamo i nodi della rete da rimuovere.
Per modificare la rete bayesiana occorre cambiare il metodo di BayesNetExampleFactory invocato a riga 36, subito dopo la dichiarazione del main.

2.2 M-Separation

La seconda strategia di ottimizzazione è stata ottenuta andando ad eliminare tutti i nodi irrilevanti per il calcolo della VE, andando a ridurre le CPT dei rispettivi nodi collegati senza modificare il risultato della query.
Prima di eliminare gli archi che collegano nodi non rilevanti, facciamo il marry tra i "parents" di uno stesso nodo tramite il metodo Marry(BayesianNetwork Network).
Marry

Il metodo ha lo scopo di sposare tutti i nodi che abbiano un figlio in comune. Per poter esprimere questa proprietà abbiamo aggiunto nella classe FullCPTNode una lista di nodi marriedNodes. Dato che dobbiamo lavorare con la classe FullCPTNode, innanzitutto convertiamo la lista di variabili della rete in FullCPTNode.

In seguito, per ogni nodo recuperiamo la lista dei suoi parents e se ha almeno due genitori aggiungiamo a ognuno di essi tutti gli altri alla lista marriedNodes.
Abbiamo deciso di implementare il metodo clone in modo da poter clonare tutti i figli dei nodi da dover rimuovere poiché in determinati casi i "parents" da eliminare erano solo alcuni

e questo inconveniente reca problemi nel momento in cui andiamo a calcolare la CPT del figlio i cui genitori NON sono stati eliminati tutti dato che nel calcolare la CPT si prendono tutti i genitori che compaiono nella CPT del nodo figlio e dato alcuni genitori non esistono più questo dava errore. Si è pensato quindi di creare un clone di tale figlio e assegnargli gli stessi 'parents' e 'children', effettuare le opportune modifiche della rete Bayesiana eliminando i nodi richiesti e successivamente "sganciare" dalla rete il nodo figlio originale cancellando ogni riferimento alla rete e sostituirlo con il clone che avrà invece come 'parents' solo i nodi ancora presenti nella rete.
Facciamo utilizzo di una HashMap in modo da avere un riferimento univoco tra il nodo originale ed il suo clone, in modo da poterlo sostituire successivamente.
Una volta avviato il programma si prende da input i valori delle variabili di query e di istanza e fatto il controllo sul dominio dei valori inseriti.

IsMSeparated

IsMSeparated(BayesianNetwork Network, List<RandomVariable> queryList, List<RandomVariable> evidenceList) ritorna la lista di variabili da rimuovere perchè m-separate data la rete, le variabili di query e quelle di evidenza.
Consiste in un algoritmo di esplorazione di un grafo basato su un'idea fondamentale: un nodo è m separato se non trovo nessun percorso a partire da una variabile di query che mi porti ad esso senza passare da una variabile di evidenza, o in altre parole un nodo non è m-separato se posso arrivarci partendo da una variabile di query e senza passare dall'evidenza.
Abbiamo definito tre liste di nodi, white, grey e black. I nodi white sono nodi ancora da visitare, i grey sono nodi esplorati i cui neighbours non sono tutti black e i black sono nodi visitati con tutti i neighborus black o grey.
In altre parole un nodo diventa black solo se tutti i suoi vicini non sono white.
La lista White conterrà all'inizio tutti i nodi della rete meno le variabili di query e di evidenza

DepthFirstSearch

depthFirstSearch(BayesianNetwork Network, FullCPTNode node, List<FullCPTNode> Path, List<Node> white, List<Node> grey, List<Node> black, List<RandomVariable> queryList, List<RandomVariable> evidenceList)
Questo metodo è responsabile di creare una lista contenente tutti i nodi esplorabili nella rete Bayesiana senza passare dalla evidenze per ciascuna delle variabili di query. Tutti i nodi contenuti nella lista sono quindi rilevanti, ossia non m-separati.

Per prima cosa aggiungiamo il nodo corrente ai nodi grey e lo rimuoviamo dai nodi white, dopodiché calcoliamo i suoi neighbours ad esclusione delle variabili di evidenza. In seguito controlliamo se tutti i neighbours del nodo sono o grey o black, in tal caso impostiamo il nodo come black e lo rimuoviamo dalla lista dei grey.

Una volta effettuati questi controlli, iteriamo su tutti in neighbours del nodo, li aggiungiamo alla lista e li visitiamo ricorsivamente con la DepthFirstSearch, dando la precedenza ai nodi white sui grey.

2.3 Pruning Edge

Infine, questa strategia di ottimizzazione è stata ottenuta andando ad eliminare gli archi figli del/i nodo/i evidenza, alterando di conseguenza le CPT del/i figli mantenendo, per quanto riguarda i nodi evidenza, solamente i valori assegnati.
Come precedentemente accennato nell'esercizio M-Separation, anche in questo caso , sono stati clonati tutti i nodi in cui si doveva modificare la CPT (in questo caso i figli dei nodi evidenza) in modo da non avere problemi nel caso in cui si modificasse il numero dei parents.
Per la rimozione degli archi è stato utilizzato il metodo *removeEdgePruning.*

RemoveEdgePruning

removeEdgePruning(List<RandomVariable> Evidence,BayesianNetwork Network)
Questo metodo data la rete Bayesiana "Network", iterando su un ciclo for, per ogni elemento della lista Evidence, elimina tutti gli archi che vanno ai figli, quindi viene aggiornato il numero dei figli(le evidenze non avranno più figli), e i padri dei figli (ai figli delle evidenze verrà tolta dai "parents" l'evidenza stessa).

2.4 Sort

Questa classe implementa gli algoritmi di sorting , in particolare *minFillOrder(BayesanNetwork n)*, *MinDegree(BayesanNetwork n)*. Entrambe queste funzioni fanno uso di un metodo ausiliario *deepCopy()*, il quale crea una *deepCopy* di un oggetto. Per *deepCopy* si intende una copia ricorsiva di tutti gli oggetti contenuti nell'oggetto principale e si differenzia proprio per questo dalla *shallow copy*, che si limita a copiare l'oggetto principale ma mantiene gli stessi riferimenti in memoria per gli oggetti contenuti al suo interno. La funzione *getNeighbours* restituisce tutti i vicini di un nodo e viene usata per calcolare sia il MinDegree che il MinFill.

3. TEST

Abbiamo effettuato dei test su diverse reti Bayesiane in modo da avere risultati differenti, le reti si differenziano nella loro complessità e struttura, la rete "GetRichNetwork" ad esempio ha una struttura 'chain' in quanto ogni 'parent' ha esattamente un singolo 'children' e così via, abbiamo scelto inoltre reti con un numero diverso di nodi .

I risultati sono stati confrontati tra loro a seconda dell'ordinamento utilizzato e sono forniti di seguito.

3.1 Reti Bayesiane su cui si effettuano i test

Nome	**Nodi**
GetRichNetwork	4
John Junior Tornado	8
Fly Safe Network	15
GetHailfinderNet	55

3.2 Rete Get Rich

- ***Pruning***

Query	**Topologico Inverso**	**MinFill**	**MinDegree**
P(Rich|Invest)	0.0464	0.0542	0.0468
P(Job|Invest)	0.0651	0.0527	0.0500
P(Job|Rich,Money)	0.0445	0.0757	0.0753
P(Rich,Job|Money)	0.0597	0.0574	0.0642

- ***M-Separation***

Query	**Topologico Inverso**	**MinFill**	**MinDegree**
P(Rich|Invest)	0.0401	0.0701	0.0569
P(Invest|Rich)	0.0563	0.0490	0.0446
P(Job|Rich,Money)	0.0520	0.0647	0.0638

P(Rich,Job|Money)	0.0567	0.0739	0.0573

❖ ***Pruning Edge***

Query	Topologico Inverso	MinFill	MinDegree
P(Rich|Invest)	0.0351	0.0487	0.0401
P(Invest|Rich)	0.0609	0.0516	0.0439
P(Job|Rich,Money)	0.0765	0.0667	0.0522
P(Rich,Job|Money)	0.0463	0.0381	0.0378

3.3 Rete JohnJunior

❖ ***Pruning***

Query	Topologico Inverso	MinFill	MinDegree
P(JohnJuniorCalls|JohnCalls)	0.0556	0.0795	0.0644
P(Earthquake|JohnJuniorCalls)	0.0719	0.0498	0.0474
P(Sleep,JohnJuniorCalls|Mary Calls)	0.0764	0.0609	0.0722
P(Tornado|JohnJunioCalls,Alarm)	0.0820	0.0583	0.0668

❖ ***M-Separation***

Query	Topologico Inverso	MinFill	MinDegree
P(JohnJuniorCalls|JohnCalls)	0.0558	0.0532	0.0767
P(Earthquake|JohnJuniorCalls)	0.0695	0.0386	0.0397

P(Sleep,JohnJuniorCalls|Mary Calls)	0.0388	0.0414	0.0411
P(Tornado|JohnJunioCalls,Alarm)	0.0837	0.0574	0.0666

❖ ***Pruning Edge***

Query	Topologico Inverso	MinFill	MinDegree
P(JohnJuniorCalls|JohnCalls)	0.0696	0.0592	0.0583
P(Earthquake|JohnJuniorCalls)	0.0344	0.0529	0.0556
P(Sleep,JohnJuniorCalls|Mary Calls)	0.0434	0.0551	0.0458
P(Tornado|JohnJunioCalls,Alarm)	0.0440	0.0577	0.0519

3.3 Rete Fly Safe Network

❖ ***Pruning***

Query	Topologico Inverso	MinFill	MinDegree
P(Fly|Turbolence)	0.1069	0.3547	0.3678
P(Poor|Panic)	0.0468	0.0470	0.0700
P(Panic,Turbolence|Fly)	0.2973	0.3247	0.4325
P(Gamble|Happy,Eredity)	0.0893	0.3301	0.372

❖ ***M-Separation***

Query	Topologico Inverso	MinFill	MinDegree
P(Fly|Turbolence)	0.1633	0.1861	0.0483
P(Poor|Panic)	0.0681	0.0945	0.0531
P(Panic,Turbolence|Fly)	0.1655	0.1783	0.1792
P(Gamble|Happy,Eredity)	0.2210	0.1646	0.1549

❖ ***Pruning Edge***

Query	Topologico Inverso	MinFill	MinDegree
P(Fly|Turbolence)	0.1632	0.2346	0.1776
P(Poor|Panic)	0.0842	0.0727	0.0898
P(Panic,Turbolence|Fly)	0.1533	0.1629	0.1710
P(Gamble|Happy,Eredity)	0.0590	0.0712	0.0575

3.4 Rete GetHailfinderNet

❖ ***Pruning***

Query	Topologico Inverso	MinFill	MinDegree
P(Morning_Bound|Mountain_Fcst)	0.0843	0.389	0.4122
P(Latest_Cin|Outflow_Frmt)	0.1012	0.5437	0.4248

P(Vis_Cloudcov,An_instab_Mt|Lliw)	0.3986	0.4490	0.4344
P(Date|Boundaries,Ins_Change)	0.3706	0.3999	0.5761

❖ ***M-Separation***

Query	Topologico Inverso	MinFill	MinDegree
P(Morning_Bound|Mountain_Fcst)	0.5771	0.3749	0.2731
P(Latest_Cin|Outflow_Frmt)	0.3171	0.3081	0.3798
P(Vis_Cloudcov,An_instab_Mt|Lliw)	0.2664	0.3090	0.2719
P(Date|Boundaries,Ins_Change)	0.2918	0.2880	0.2449

❖ ***PruningEdge***

Query	Topologico Inverso	MinFill	MinDegree
P(Morning_Bound|Mountain_Fcst)	0.3082	0.2607	0.2677
P(Latest_Cin|Outflow_Frmt)	0.2989	0.2857	0.2974
P(Vis_Cloudcov,An_instab_Mt|Lliw)	0.3351	0.2985	0.0.2882
P(Date|Boundaries,Ins_Change)	0.2621	0.2635	0.2707

4. ANALISI DEI RISULTATI

Dai test effettuati è emerso che al crescere dei nodi della rete gli algoritmi più efficienti sono quelli di pruning dei nodi in quanto apportano un considerevole risparmio di spazio nella JVM, che può rivelarsi fondamentale nel processamento di reti molto complesse.
Riguardo il Pruning dei nodi semplice, possiamo concludere che la quantità di nodi eliminati, a seconda della query e della topologia della rete, risulta in media maggiore rispetto a quella di M-Separation.
Tuttavia, a seconda della posizione e del numero delle variabili di evidenza, M-Separation può eliminare molti più nodi del Puring semplice, per esempio nel caso in cui le variabili di evidenza corrispondono al Markov Blanket della variabile di query tutti i possibili percorsi sono bloccati e quindi gran parte della rete viene eliminata.
Anche il Pruning degli archi risulta migliorare il tempo di computazione e lo spazio in quanto riduce la dimensione delle CPT, soprattutto nel caso in cui le variabili di evidenza hanno molti figli.
Per quanto riguarda gli algoritmi di ordinamento è possibile osservare un discreto impatto sui tempi di esecuzione e variabile a seconda della topologia della rete e della posizione delle variabili di query e di evidenza.
Possiamo concludere che questi algoritmi riducono in maniera significativa le richieste computazionali in termini di spazio e tempo della Variable Elimination.

5. RETI BAYESIANE DINAMICHE

Le classi principali di questo esercizio sono CustomDynamicBayesNet, che consente di rappresentare una rete bayesiana dinamica e CustomEliminationAsk che implementa l'inferenza per reti dinamiche.

CustomDynamicBayesNet contiene il metodo forward(), il quale consente di far avanzare la rate dinamica all'istante successivo.
Inoltre utilizziamo il Rollup Filtering considerando solamente due slice alla volta, in questo modo avremo un notevole incremento delle prestazioni.

5.1 Test

L'esercizio è stato testato su tre reti di prova, applicando due forward e nelle reti Complex e Big ogni variabile al tempo $t+1$ è fatta dipende da tutte le altre al tempo t

Rete	**Nodi**	**Tempo**
Complex	2	0.4364

Simple	2	0.4768
Big	3	0.4663

5.2 Considerazioni finali

Si evince come il Rollup Filtering porti ad un miglioramento nelle prestazioni temporali del processo di inferenza sulle reti Bayesiane dinamiche perché per effettuare l'inferenza al tempo t+1 e sufficiente conoscere lo stato della rete al tempo t e l'evidenza al tempo t+1.

Questo dipende principalmente dall' assunzione markoviana, ossia che le evidenze al tempo t+1 dipendono solamente dalle variabili di stato al tempo t e dalla staticità del processo , per cui né il modello sensoriale né quello di transizione cambiano nel tempo. Questa inferenza dal tempo t al tempo t+1 ha tempo costante ed indipendente dalla grandezza di t.