IALAB 19/20

Progetto Uncertainty

Davide Rubinetti 832763  
Davide Di Luccio 837927

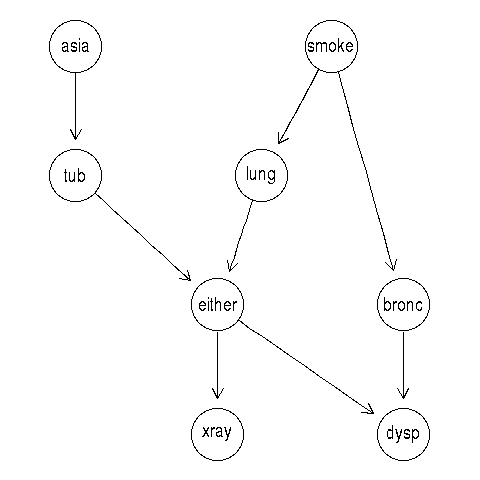
# 

# TEST SU RETI BAYESIANE

## PRUNING NODI

Il primo metodo di pruning dice che una variabile Y è irrilevante a meno che Y Ancestors({X} U **E**). È facile dunque intuire che, se X ed **E** sono molto in basso nella rete, i loro antenati saranno molto numerosi, e dunque per Y è più difficile essere irrilevante. Viceversa, se X ed **E** sono molto in alto, essi avranno molti pochi antenati e dunque per Y è più semplice essere irrilevante.   
Inoltre, si aggiunge il fatto che per far sì che il numero di antenati di {{X} U **E**} sia ristretto, il numero di variabili di evidenza e di query deve ovviamente essere piccolo, oltre al fatto che la rete non deve avere tanti archi.

Nel secondo metodo di pruning, viceversa, è facile trovare molte variabili irrilevanti qualora abbiamo tante variabili di evidenza e poche di query. Infatti, dopo aver costruito il moral graph della rete, il metodo afferma che Y è irrilevante se è separata da X attraverso **E** nel moral graph. Dunque, più è alto il numero delle variabili di **E** e, allo stesso tempo, meno è alto il numero di variabili di query, più sarà difficile collegare Y ad X senza passare da una variabile di evidenza.   
In aggiunta, con questo metodo è più facile trovare nodi irrilevanti se la rete è poco connessa, cioè non presenta un alto numero di archi, e se le evidenze sono posizionate in modo sparso nella rete.

Proviamo su un esempio semplice (rete Asia   
<https://www.bnlearn.com/bnrepository/discrete-small.html#asia>):

P(Asia|Either = Yes)

In questo caso, come nodo irrilevante risulta solo XRay. Infatti nel moralgraph, le variabili tub e lung sono state “sposate”.

Cambiando quindi query :

P(Asia|Either = Yes, Lung = Yes), le variabili irrilevanti diventano XRay, Smoke, Bronc e Dysp in quanto non esiste più un percorso che collega queste variabili a quella di query.

Dunque, abbiamo fatto opportuni test con la seguente rete **Insurance**(disponibile a <https://www.bnlearn.com/bnrepository/discrete-medium.html#insurance>):

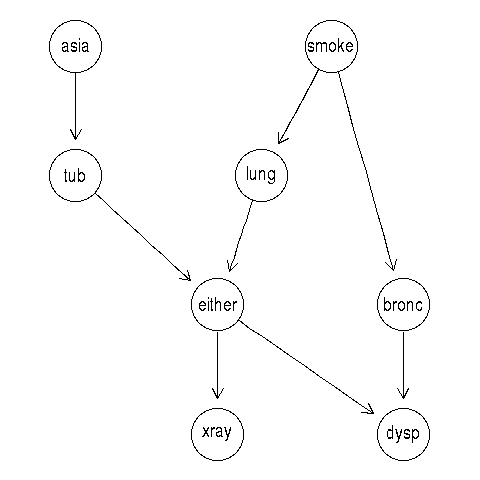


Le nostre intuizioni hanno effettivamente trovato luogo nei test che abbiamo effettuato.  
Consideriamo ad esempio la seguente query, sulla rete Insurance:  
P(Cushioning|ThisCarDam = Mild, MedCost = Thousand, ILiCost = Thousand, OtherCarCost = Thousand, ThisCarCost = Thousand, Theft = False)

abbiamo molte variabili di evidenza, tutte posizionate in basso nella rete. Come abbiamo descritto prima, il primo metodo di pruning non trova molti nodi irrilevanti, per la precisione ne trova solo 4.   
Consideriamo invece una query in cui abbiamo poche evidenze e poche variabili di query e, inoltre, sono posizionate molto in alto nella rete.

Ad esempio P(Age | SocioEcon = Prole, GoodStudent = True): come ci si aspettava, i nodi irrilevanti in questo caso sono molti di più, addirittura 24.

Focalizziamoci adesso sul secondo metodo di pruning. Con una query in cui si hanno poche evidenze, ad es. P(Age | SocioEcon = Prole, GoodStudent = True), troviamo pochi nodi irrilevanti, soltanto uno per l’esattezza. Viceversa, se abbiamo tante evidenze, i nodi irrilevanti diventano molti di più.



## PRUNING ARCHI

Un arco U -> X è irrilevante se U è in **E**; inoltre la CPT di X può essere sostituita da una CPT più piccola tenendo conto del valore u del padre U nell’evidenza **e**.   
Chiaramente questo metodo funziona quando le evidenze sono molte, perché possono esserci più archi del tipo appena descritto ma, soprattutto, è utile quando le variabili di evidenza hanno tanti figli, perché si andrà a ridurre un gran numero di CPT.

# Rete Insurance, con query : P(Cushioning | ThisCarDam = Mild, MedCost = Thousand, ILiCost = Thousand, OtherCarCost = Thousand, ThisCarCost = Thousand, Theft = False)

V.E. Russel & Norvig e ordine topologico inverso

|  |  |  |
| --- | --- | --- |
| Metodo di pruning | #Irrilevanze | Tempo di esecuzione(ms) |
| Niente | / | 2200 |
| Nodi Irrilevanti | 4 | 1639 |
| M-Separation | 1 | 1755 |
| Archi Irrilevanti | 4 | 2000 |

Si vede subito come i metodi di pruining diminuiscono il tempo di esecuzione della VE.

# \* i tempi calcolati sono indicativi

# Rete Insurance, con query : P(Age | SocioEcon = Prole, GoodStudent = True)

V.E. Russel & Norvig e ordine topologico inverso

|  |  |  |
| --- | --- | --- |
| Metodo di pruning | #Irrilevanze | Tempo di esecuzione(ms) |
| Niente | / | 5173 |
| Nodi Irrilevanti | 23 | 79 |
| M-Separation | 1 | 4468 |
| Archi Irrilevanti | 7 | 4106 |

Esempio molto significativo che mostra un caso ideale per il primo metodo di pruning, poche variabili di query e di evidenza: addirittura si possono ignorare 23 nodi e il tempo di esecuzione della V.E. scende da 5 secondi a una manciata di millisecondi.

# Rete polytree50nodi (BNGenerator), con query :

# P(Node10 | Node15 = State0, Node48 = State0)

V.E. Russel & Norvig e ordine topologico inverso

|  |  |  |
| --- | --- | --- |
| Metodo di pruning | #Irrilevanze | Tempo di esecuzione(ms) |
| Niente | / | 152 |
| Nodi Irrilevanti | 48 | 50 |
| M-Separation | 1 | 233 |
| Archi Irrilevanti | 1 | 160 |

## Possiamo notare come, con una rete singly connected (polytree), le differenze di prestazione vengono a diminuire, dato che di per sé, con un polytree, la VE può avere una complessità temporale meno elevata, motivo per il quale si può anche decidere di non compiere nessun metodo di pruning. Possiamo altresì notare come il metodo di m-separation, nonostante sia un metodo valido per ridurre il tempo di esecuzione della VE (almeno in linea generale), può risultare, in questo caso, in cui abbiamo un polytree, addirittura uno spreco di lavoro.

## ORDINAMENTO

Per testare l’ordinamento delle variabili, ci siamo ispirati alla Variable Elimination della versione di Darwiche, che riceve come parametro in input un ordinamento delle variabili su cui effettuare il calcolo. Invece, l’algoritmo EliminationAsk() proposto da Russel & Norvig, esegue il calcolo esclusivamente con l’ordinamento topologico inverso delle Random Variable (l’ordinamento è hard-coded nell’algoritmo).

Utilizzando dunque la Variable Elimination con versione di Darwiche, abbiamo provato a calcolare delle query per notare come, in linea del tutto generale, le euristiche che suggeriscono un preciso ordinamento delle variabili siano effettivamente molto utili per diminuire il tempo di esecuzione dell’algoritmo.

Infatti, la Variable Elimination ha complessità lineare nel numero di variabili ma esponenziale nella treewidth della rete. Chiaramente un corretto ordinamento delle variabili può far sì che il fattore più grande che viene generato durante l’algoritmo sia di dimensione il più possibile contenuta.

Nella fattispecie, il MinDegreeOrder suggerisce un ordinamento in cui si elimina per prima la variabile che, nell’interaction graph, ha meno vicini e, dunque, che sembra condurre alla costruzione del fattore più piccolo.

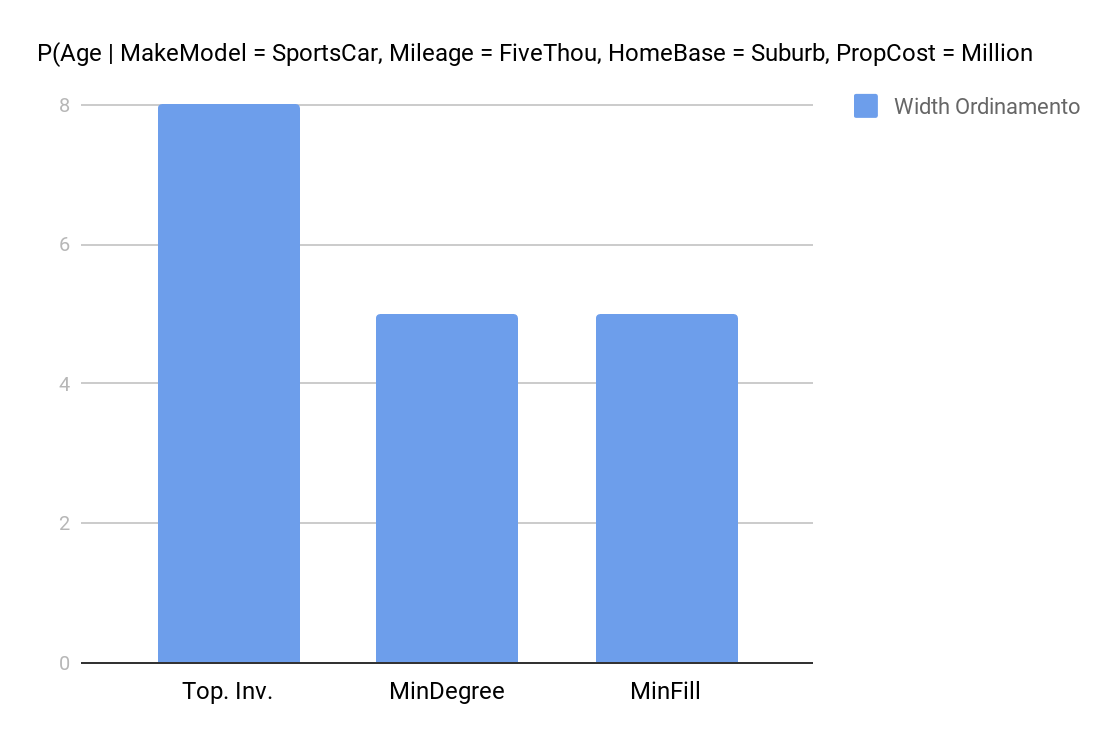
L’altro ordinamento implementato, MinFillOrder, va a considerare la variabile che, nell’algoritmo di MinDegreeOrder, aggiungerebbe il minor numero di archi tra i suoi vicini. Ad esempio se consideriamo una variabile che, nell’interaction graph, ha già tutti i vicini collegati, potremmo avere già esaminato i fattori di tutti i suoi vicini e dunque non verrà costruito un fattore più grande.

Facendo alcuni approfondimenti durante lo sviluppo, abbiamo scoperto che costruire tali ordinamenti può essere fatto in tempo O(n lgn).

Ebbene, considerando la query P(Age | MedCost = Thousand) nella rete Insurance, il tempo di esecuzione con questi ordinamenti si è notevolmente ridotto. Il tempo di esecuzione è anche minore rispetto alla versione della VE di Russel & Norvig.

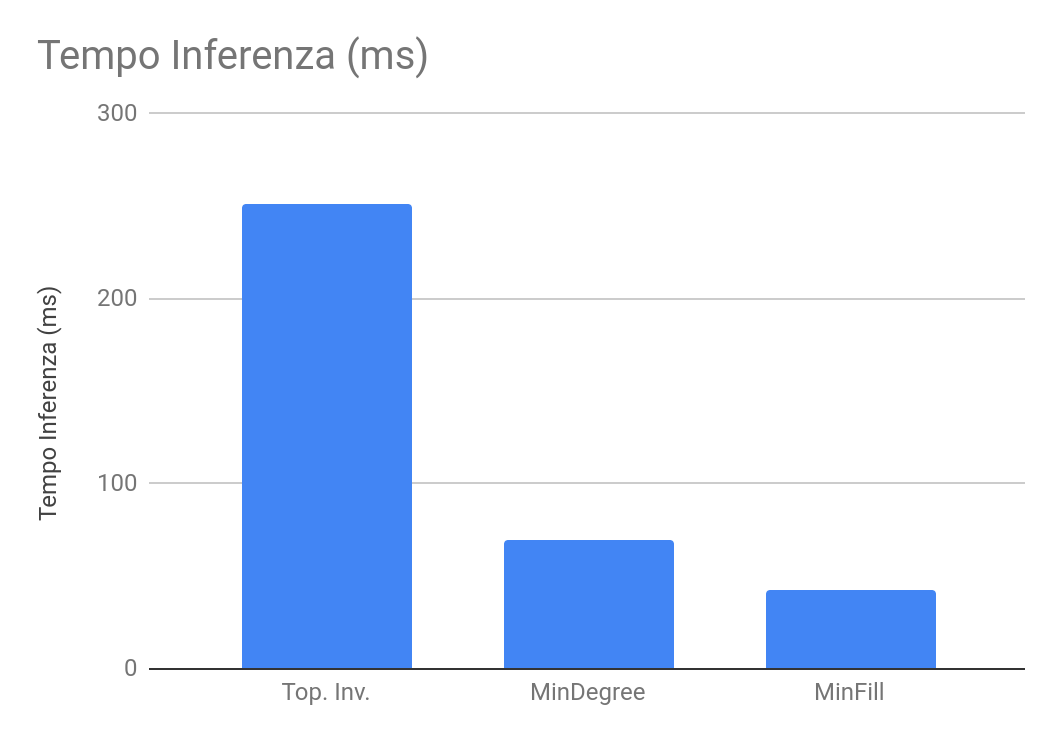
|  |  |  |
| --- | --- | --- |
| EliminationAsk R&N | MinDegreeOrder | MinFillOrder |
| 3109ms | 186ms | 112ms |

Si conclude che le euristiche che suggeriscono gli ordinamenti delle Random Variable sono molto utili, non solo perché garantiscono un tempo di esecuzione minore, ma perché di fatto affrontare il problema della ricerca dell’ordinamento migliore è un problema NP-complete.



Il primo grafico mostra la width nella V.E. in base all’ordinamento utilizzato.

Notiamo che quello topologico inverso in questo caso è sfavorevole, rispetto ai due ordinamenti implementati.



Anche per quanto riguarda il tempo, ovviamente, i dati rispecchiano quanto emerge tramite la width. Infatti abbiamo una notevole differenza tra il topologico inverso che con la width più alta e la natura esponenziale dell’algoritmo porta a questa differenza di valori. Inoltre, tra i due ordinamenti, MinFill impiega meno tempo.

# 

# 

# Rete insurance, con query :

# P(Cushioning | ThisCarDam = Mild, MedCost = Thousand, ILiCost = Thousand, OtherCarCost = Thousand, ThisCarCost = Thousand, Theft = False)

|  |  |  |
| --- | --- | --- |
| Ordinamento | Width | Tempo di esecuzione(ms) |
| Topologico Inverso (R&S) | 4 | 1498 |
| Topologico Inverso (Drw) | 9 | 1102 |
| MinDegree | 7 | 196 |
| MinFill | 6 | 118 |

Si noti come MinDegree e MinFill sono euristiche che consentono un ordinamento delle Random Variable con cui la VE riduce notevolmente il tempo d’esecuzione. Nelle conclusioni viene brevemente descritto un test identico ma con una rete molto più complessa.

# 

# Rete polytree 50 nodi (generata con BNGenerator)

# P(Node10 | Node15 = State0, Node48 = State0)

|  |  |  |
| --- | --- | --- |
| Ordinamento | Width | Tempo di esecuzione(ms) |
| Topologico Inverso (R&S) | 5 | 173 |
| Topologico Inverso (Drw) | 8 | 66 |
| MinDegree | 5 | 122 |
| MinFill | 5 | 65 |

# 

# Stessa query con eliminazione dei nodi irrilevanti

|  |  |  |
| --- | --- | --- |
| Ordinamento | Width | Tempo di esecuzione(ms) |
| Topologico Inverso (R&S) | 2 | 26 |
| Topologico Inverso (Drw) | 2 | 20 |
| MinDegree | 2 | 22 |
| MinFill | 2 | 15 |

Le differenze di prestazioni utilizzando un polytree non sono così marcate.

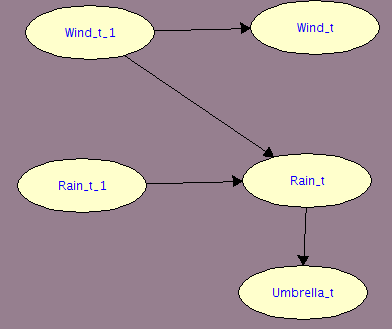
# Test su Reti Bayesiane Dinamiche

INTRODUZIONE

In questa seconda parte del progetto abbiamo lavorato sulle reti bayesiane dinamiche, le quali sono come delle reti bayesiane statiche che sono modellate e organizzate in sequenze temporali. Ciascun time slice è condizionalmente dipendente dal precedente.

Per poter utilizzare le DBN abbiamo bisogno di:

1. P(X0) -> Probabilità a priori sulle variabili di stato
2. P(Xt+1 | Xt) -> Modello di transizione
3. P(Et | Xt) -> Modello sensoriale

Quindi per poter lavorare sulle DBN, siamo partiti da alcune reti statiche utilizzate nella prima parte del progetto che abbiamo “dinamizzato”.

Ecco quindi un esempio :

I nodi Wind\_t\_1 e Rain\_t\_1, al tempo precedente, condizionano le variabili al tempo successivo.

Per fare ciò la libreria aima.core offre l’implementazione della classe DynamicBayesNet. Ci siamo accorti però che questa gestisce le transizioni tra un tempo ed il successivo tramite una Map<RandomVariable, RandomVariable>. Quindi abbiamo ipotizzato che non è possibile, per una variabile, condizionarne più di una al tempo successivo (come Wind\_t\_1 nell’esempio). Perciò abbiamo modificato questa classe affinchè per ogni variabile si possano gestire una lista di variabili che condiziona al passo successivo.

INFERENZA

L’inferenza sulle reti bayesiane dinamiche può essere eseguita attraverso l’uso dell’algoritmo Rollup Filtering.

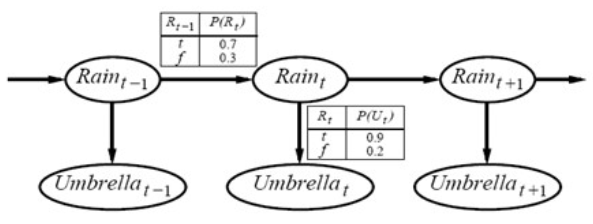
Questo ci permette di risolvere il problema chiamato “unrolling” di cui soffrono le dbn: dato che queste possono essere costruite su un numero illimitato di intervalli, mantenere in memoria tutta la rete nel corso della query porterebbe ad un eccessivo uso della stessa.

Invece, con il metodo del Rollup Filtering, possiamo focalizzarci solo su due slice di tempo per volta. Quindi una volta calcolati i fattori necessari tra i tempi t-1 e t, possiamo proseguire e “dimenticare” i nodi precedenti.

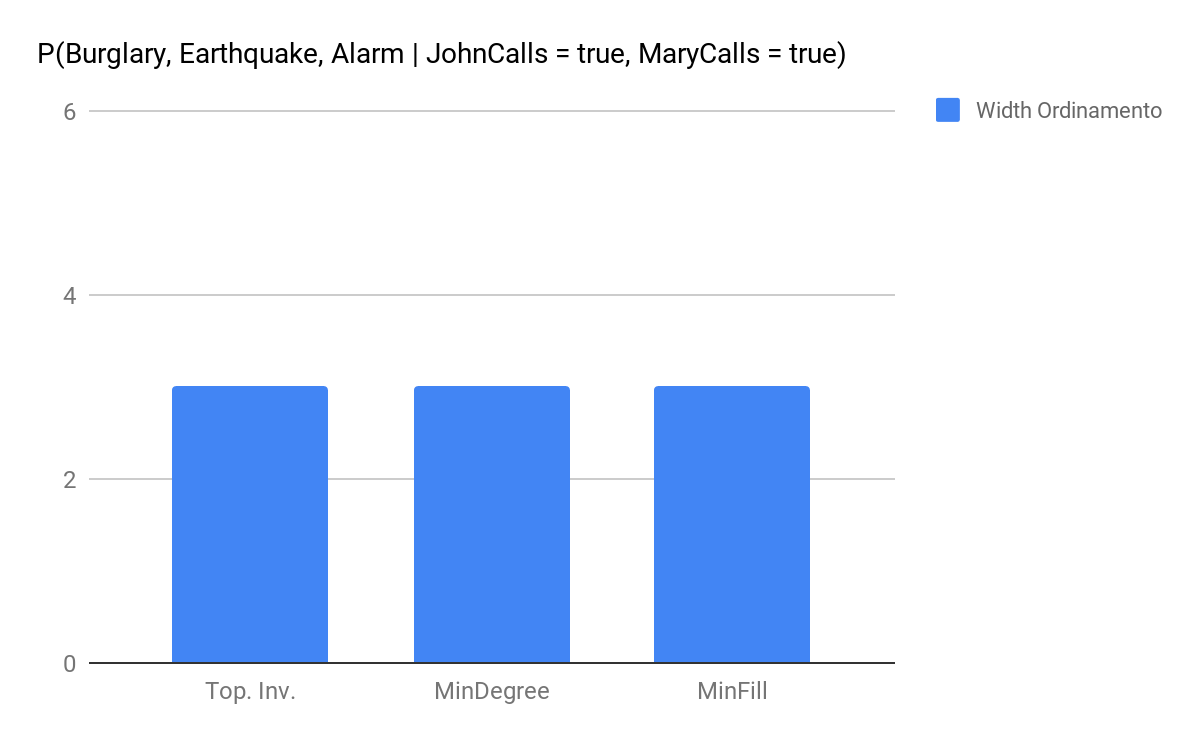
L’implementazione del Rollup Filtering è iniziata con la modifica dell'algoritmo della Variable Elimination (versione Darwiche) implementato per la prima parte del progetto.

Questo è stato necessario per far sì che, anziché restituire un unico fattore, restituisse un insieme di fattori. Infatti in questo caso non ci serve un fattore unico, il quale viene costruito nell’ultimo passo dell’implementazione originale in cui tutti i fattori vengono moltiplicati e il quale corrisponde alla distribuzione di probabilità richiesta dalla query.

Al primo passo di esecuzione, l’insieme dei fattori in input all’algoritmo è vuoto e dunque si prende la probabilità a priori P(X0), mentre nei successivi sarà il risultato dell’esecuzione al passo precedente.



TEST



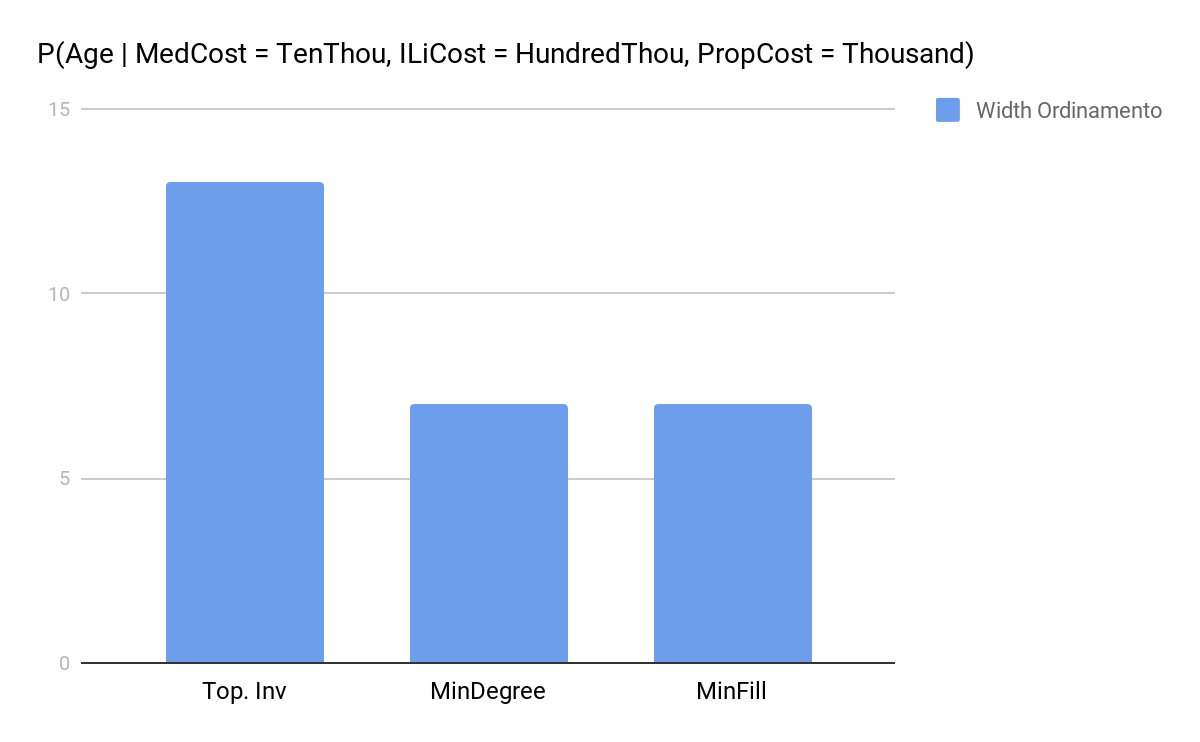
Rete Alarm :

Come già analizzato nella prima parte, in questa semplice rete, non troviamo sostanziali differenze tra i vari ordinamenti e/o pruning.

Questo perché la rete, di fatto, è piccola (5 nodi) e dunque non permette di trovare delle differenze di prestazione significative.

Rete Insurance :

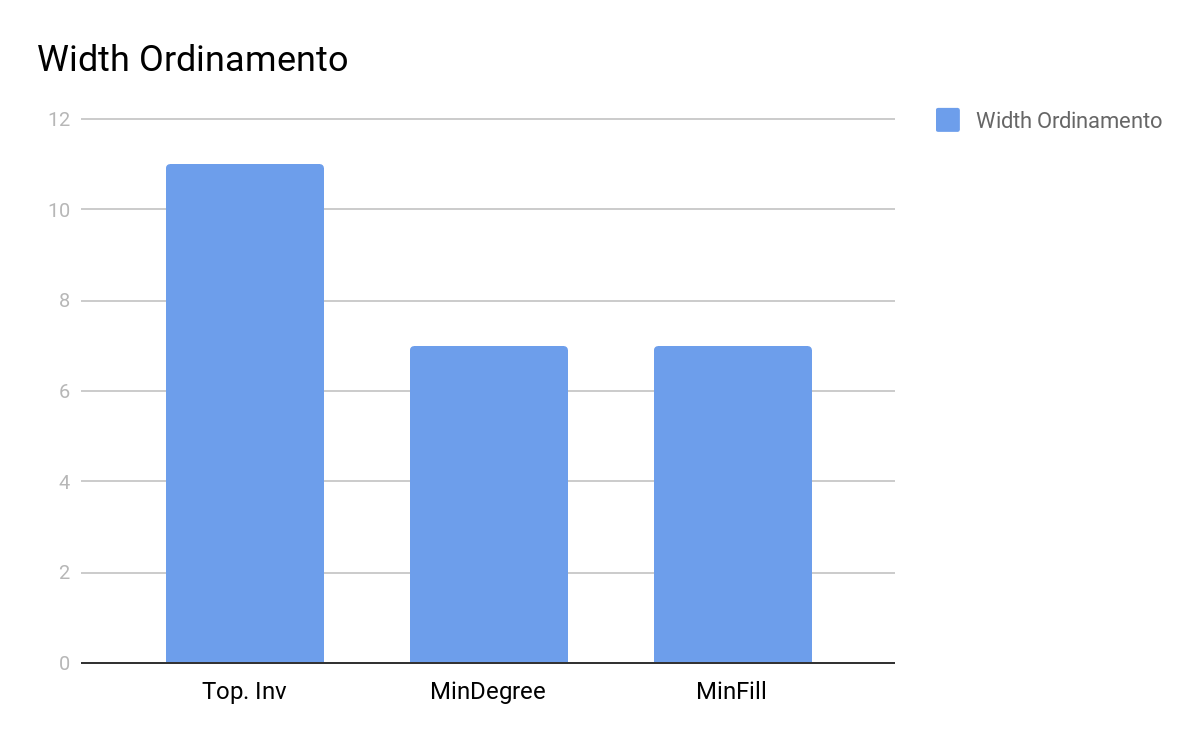
In questa rete, composta da 27 nodi, invece riusciamo ad evidenziare alcuni risultati.



Come si evince dal grafico, la width dell’ordinamento, è superiore per l’ordinamento topologico inverso utilizzato di base dalla libreria aima. I due ordinamenti da noi implementati, invece, in questo caso si equivalgono.

I tempi di esecuzione inoltre sono molto diversi.

Il topologico inverso per ogni slice di tempo impiega circa qualche secondo ad effettuare la query, contro i pochi millisecondi degli altri due ordinamenti.

Utilizzando il pruning dei nodi irrilevanti, notiamo un notevole miglioramento delle prestazioni anche per quanto riguarda l’ordinamento hard-coded in aima.

Infatti anche dal punto di vista del tempo di esecuzione dell’inferenza questo viene abbattuto. Mentre per i restanti due non vi sono ulteriori miglioramenti.

# 

# Rete insurance, con query : t = 5

# P(Age, MakeModel, CarValue |MedCost = Thousand, ILiCost = TenThou, PropCost = Million)

|  |  |  |
| --- | --- | --- |
| Ordinamento | Width | Tempo di esecuzione(ms) |
| Topologico Inverso (Drw) | 11 | 3500 |
| MinDegree | 7 | 154 |
| MinFill | 7 | 90 |

Conclusioni

Da come si evince dai test, i metodi di pruning sono validi e garantiscono un’efficienza della Variable Elimination, almeno nella stragrande maggioranza dei casi. Solo in reti molto semplici (polytree) le differenze di prestazioni sono molto meno marcate, ci sono anche rari casi in cui compiere il pruning può andare solo peggiorare il tempo di esecuzione. Abbiamo provato la Variable Elimination su una rete molto grande e complessa (Andes, <https://www.bnlearn.com/bnrepository/discrete-verylarge.html#andes>): senza nessun pruning, la Variable Elimination, dopo 10 minuti, non ha terminato l’esecuzione, motivo per il quale è stato ragionevole interromperne l’esecuzione. Utilizzando dei metodi di pruning, invece, l’esecuzione è stata di pochi secondi.

Per quanto riguarda le reti dinamiche, in generale, vale quanto detto sui metodi di pruning. Inoltre, il rollup filtering, dato che effettua il calcolo su due slice per volta, permette di ridurre al minimo la complessità spaziale sul numero di nodi della rete, per via del fatto che la rete non viene effettivamente “srotolata” fino al tempo t.