Intelligenza Artificiale e Laboratorio

Relazione progetto Prolog anno 2019/2020

Studenti:

Sanfilippo Paolo

Giuseppe Biondi

Antonio Surdo

Traccia:

- **Reti statiche:**
  Implementare i seguenti tipi di Pruning:
    - Nodi irrilevanti ancestor
    - Nodi irrilevanti m-separated
    - Archi irrilevanti

  Implementare i seguenti tipi di ordinamento:

    - Reverse topological order
    - Min-degree order
    - Min-fill order

  Eseguire eserimenti su diverse Bayesian network confrontando il tempo di Variable Elimination.

- **Reti dinamiche:**

  Modificare l'algoritmo di Variable Elimination per implementare il Rollup Filtering su reti dinamiche.

Progetto reti statiche

Pruning

E' stata implementata una classe Pruning.java la quale offre tre metodi statici per applicare dei specifici pruning:

- pruningNodeAncestors
- pruningNodeMSeparated
- pruningEdge

Questi metodi prendono in input una rete bayesiana, un array RandomVariable contenente le nostre query e un array di AssignmentProposition con le evidenze. Il risultato è una nuova rete bayesiana contenente esclusivamente i nodi rilevanti in base al tipo di pruning scelto.

Si è scelto di ricreare la rete bayesiana per evitare di dover modificare le librerie Aima in quanto gestiscono liste di nodi non modificabili. L'approccio generale dunque consisterà nella creazione di una lista contenente le RandomVariable dei nodi rilevanti che sarà passata ad un metodo per la creazione della nuova rete.

Vediamoli più nel dettaglio:

- Nodi irrilevanti ancestor:

  Considerare esclusivamente i nodi antenati delle variabili passate in input come query e evidenze.
  A livello implementativo raccogliamo la lista dei nodi antenati, essa sarà ottenuta tramite una chiamata ricorsiva che collezionerà i genitori di ogni nodo fino al raggiungimento di una radice.
  La nuova rete sarà creata contenendo solo i nodi di tale lista.

- Nodi irrilevanti m-separated:

  Rimuovere quei nodi che sono m-separated da un nodo query tramite un nodo evidenza.
  Per m-separated intendiamo che costruito il grafo morale, un nodo è m-separato e quindi irrilevante se è possibile raggiungerlo esclusivamente passando da un nodo evidenza.
  A livello implementativo eseguiamo una visita in profondità sulla rete bayesiana per raccogliere in una lista i nodi rilevanti.
  Tale visita deve tenere conto dei collegamenti di un grafo morale ma questo è garantito grazie alle funzionalità di Aima dove da ogni nodo possiamo risalire ai fratelli cercando i genitori di ogni figlio.
  La chiamata ricorsiva avviene soltanto se non ci troviamo in un nodo evidenza, questo ci permette di creare il taglio che escluderà solo i percorsi raggiungibili attraverso le evidenze.

- Archi Irrilevanti:

  Rimuovere tutti gli archi U->X, originati da un nodo U contenuto nelle evidenze.
  A livello implementativo consiste nel ricostruire la rete bayesiana escludendo gli archi uscenti dalle evidenze e modificando la CPT dei figli con solo i valori per il valore dell'evidenza.

La modifica della CPT in base al pruning è stata implementata nei seguenti modi:

- M-separated: Eseguiamo la somma delle variabili rimosse e normalizziamo il fattore.
- Archi irrilevanti: Estraiamo le righe corrispondenti al valore dell'evidenza genitore e normalizziamo il fattore

Ordinamento

E' stata implementata una classe Order.java che prende una rete bayesiana come input e offre i seguenti tipi di ordinamento:

- reverseTopologicalOrder
- minDegreeOrder
- minFillOrder

Questi metodi partendo dalla rete bayesiana globale salvata nell'istanziazione di Order restiùtuiranno una lista di RandomVariable come output.

Nel caso di min degree order e min fill order l'idea generale consiste nel costruire un grafo dove ogni vertice è una RandomVariable della nostra rete e può essere riassunta nei seguenti punti:

1) scegliere un nodo dal grafo in base al criterio scelto
2) collegare tra di loro tutti i nodi ad esso adiacenti
3) rimuoverlo dal grafo

Vediamo ora in dettaglio come verranno create queste liste:

- Reverse topological order:
  Questo metodo inverte il risultato del metodo di Aima che restituisce l'elenco delle RandomVariable nell'ordine topologico.

- Min degree order:
  Ad ogni iterazione il nodo scelto da rimuovere sarà quello con il minor numero di archi.
  La lista risultante sarà dunque un ordinamento di RandomVariable crescente sul numero di archi.

- Min fill order:
  Ad ogni iterazione il nodo scelto sarà quello che se rimosso genererebbe il minor numero di nuovi collegamenti tra i suoi adiacenti.
  La lista risultante sarà dunque un ordinamento di RandomVariable crescente sul numero di collegamenti generati.

Variable Elimination

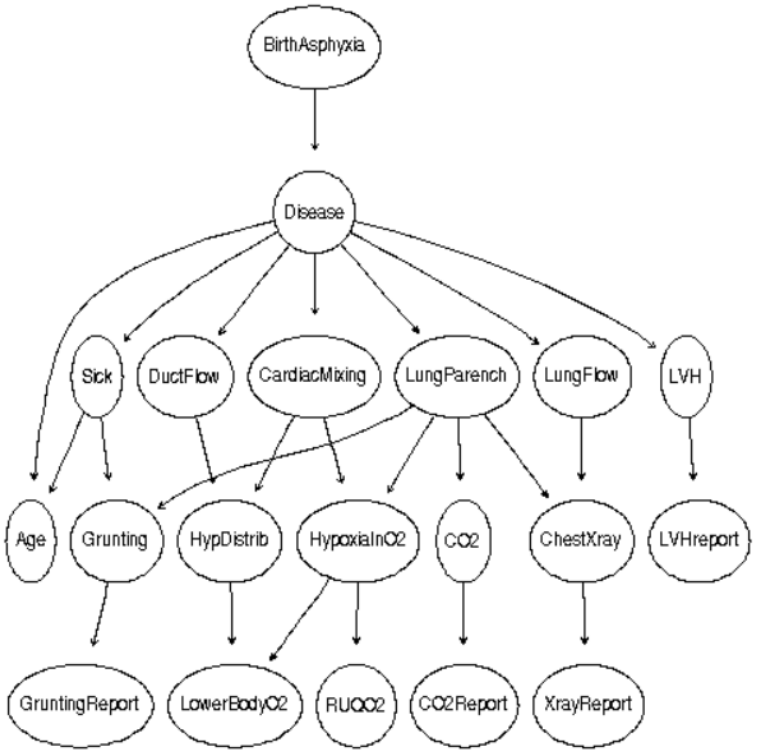
Per l'implementazione del task di VE è stata creata una classe EliminationDarwiche.java seguendo il pseudocodice fornito a lezione sull'algoritmo di Darwiche

Risultati

Per testare il nostro algoritmo di Variable Elimination, abbiamo utilizzato 4 reti selezionate dal repository del sito “bnlearn”. Abbiamo scelto delle reti di diversa grandezza, in modo da misurare l’andamento dell’algoritmo. Nello specifico abbiamo preso due parametri di paragone:

1. Tempo di esecuzione.
2. Cardinalità del fattore di grandezza massima

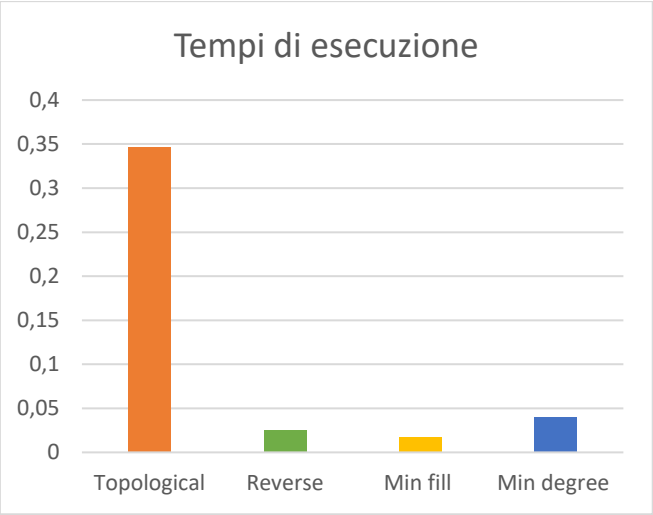
Test su rete Child (20 nodi):

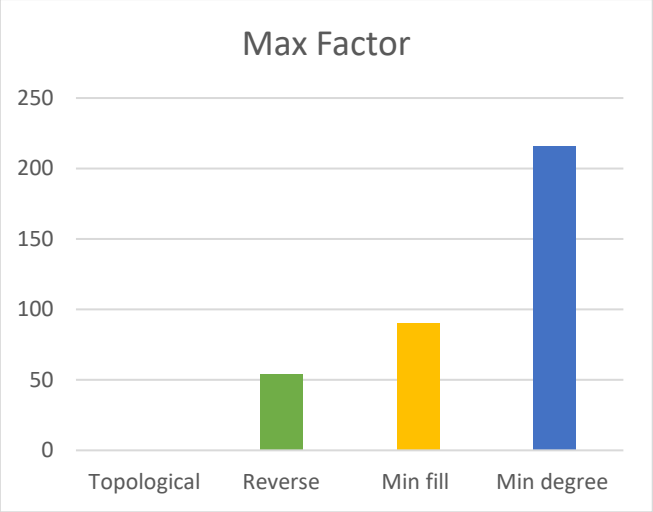


Descrizione:

Number of nodes: 20
Number of arcs: 25
Number of parameters: 230
Average Markov blanket size: 3.00
Average degree: 1.25
Maximum in-degree: 2

Esempio usando query CO2 e evidenza Age=0-3_days:





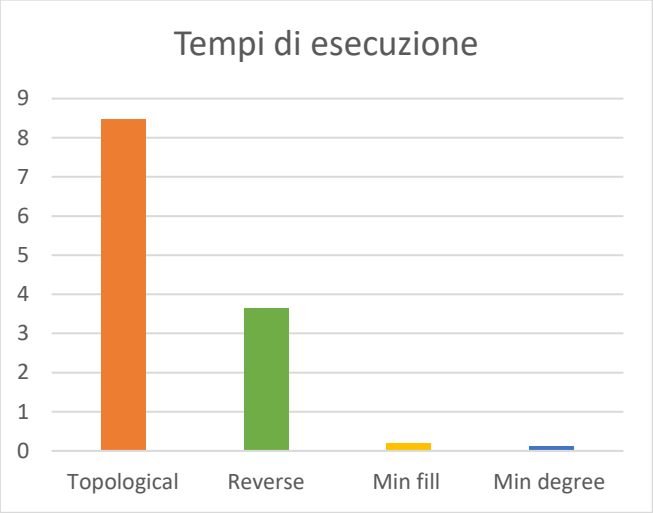
L'ordine topologico ottiene un factor di 12960.

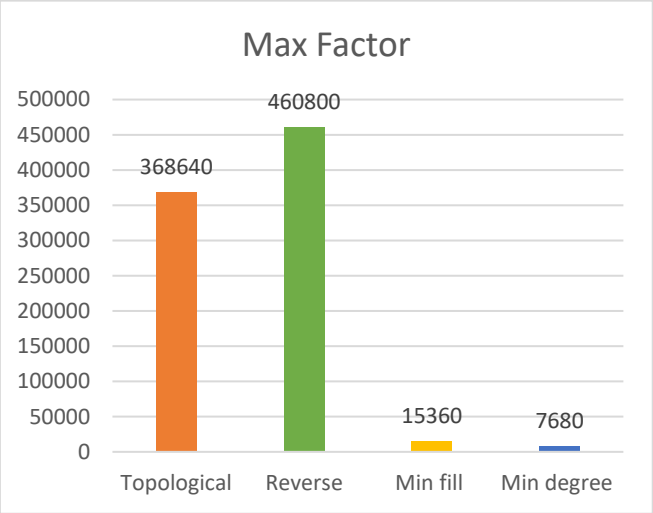
Test su rete Insurance (27 nodi):

Descrizione:

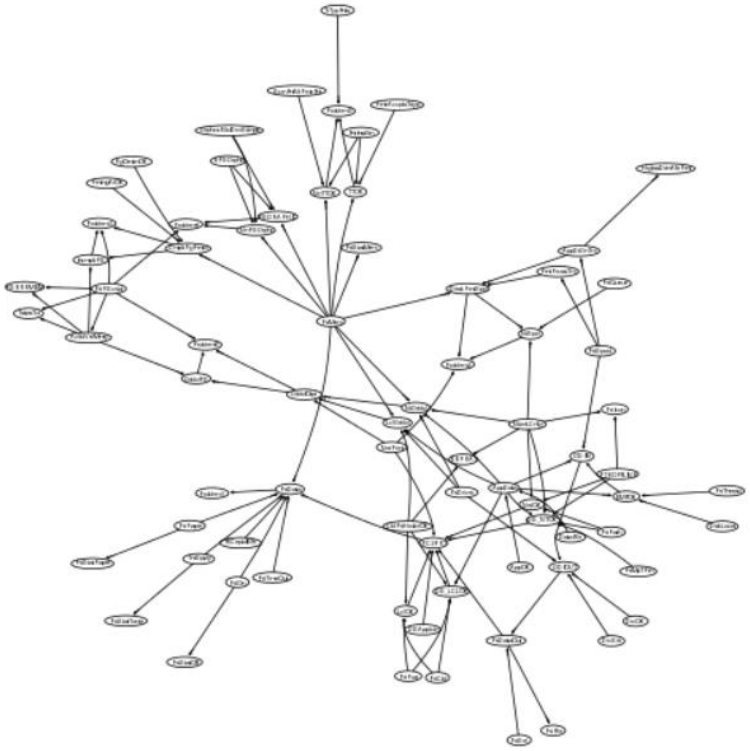
Number of nodes: 27
Number of arcs: 52
Number of parameters: 984
Average Markov blanket size: 5.19
Average degree: 3.85
Maximum in-degree: 3

Esempio usando query PropCost e evidenza DrivingSkill=Normal:





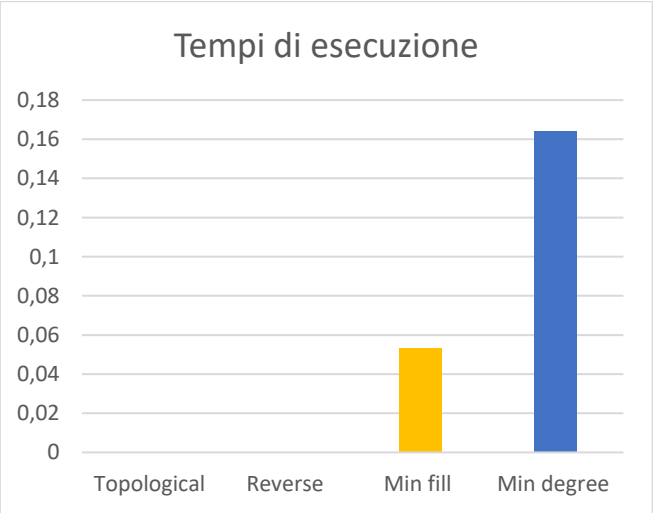
Test su rete win95pts (76 nodi) :

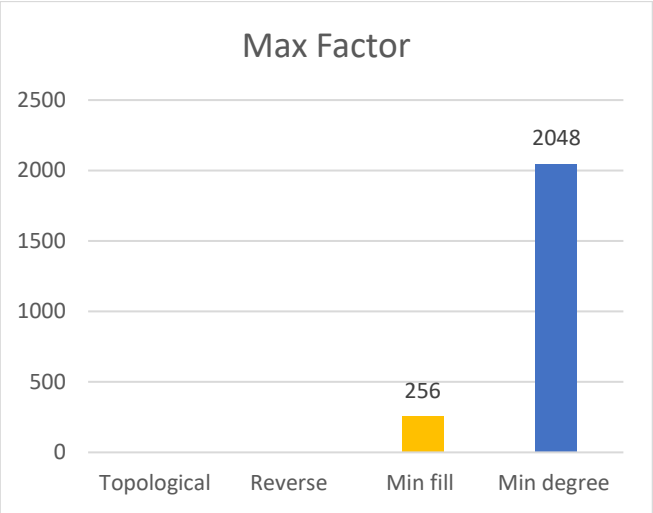


Descrizione:

Number of nodes: 76
Number of arcs: 112
Number of parameters: 574
Average Markov blanket size: 5.92
Average degree: 2.95
Maximum in-degree: 7

Esempio usando come query EPSGrphc e evidenza GDIOUT = No





Topological e Reverse order risultano non applicabili, perché hanno un costo computazionale troppo elevato.

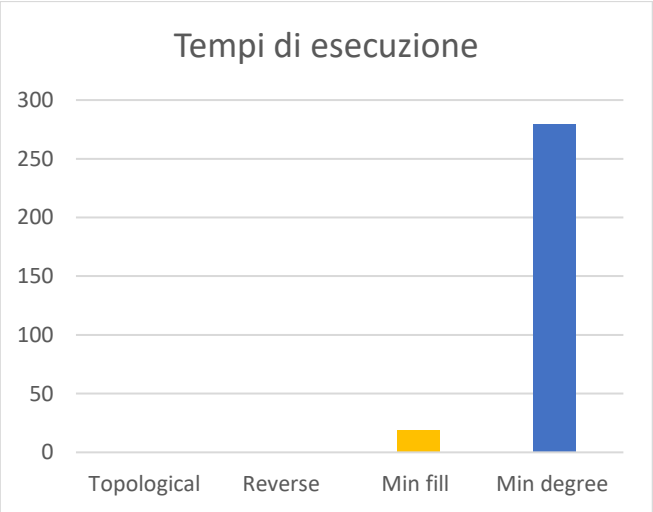
Test su rete pigs (411 nodi) :

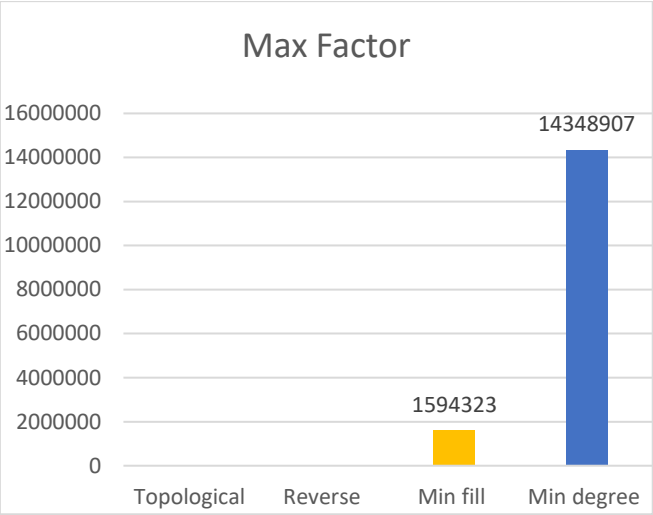
Link alla rete: https://www.bnlearn.com/bnrepository/discrete-verylarge.html#pigs

Descrizione:

Number of nodes: 441
Number of arcs: 592
Number of parameters: 5618
Average Markov blanket size: 3.66
Average degree: 2.68
Maximum in-degree: 2

Esempio usando come query P48084891 ed evidenza P543551889 = 2:





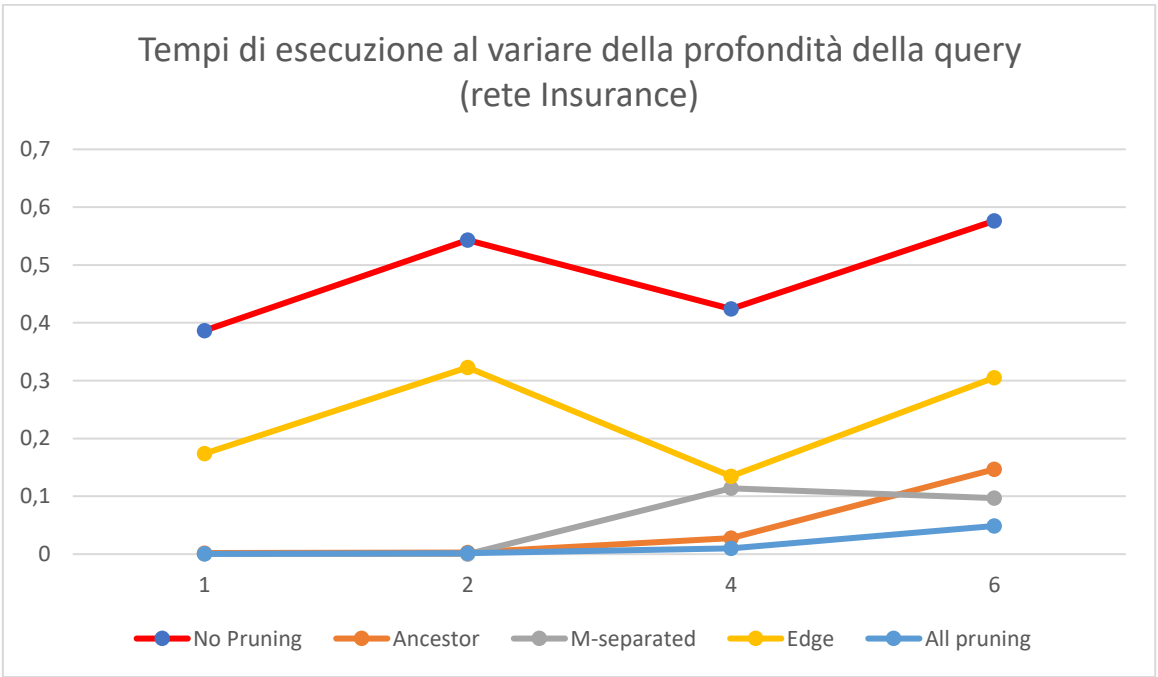
Data la grandezza della rete, anche qui Topological e Reverse order non sono applicabili.

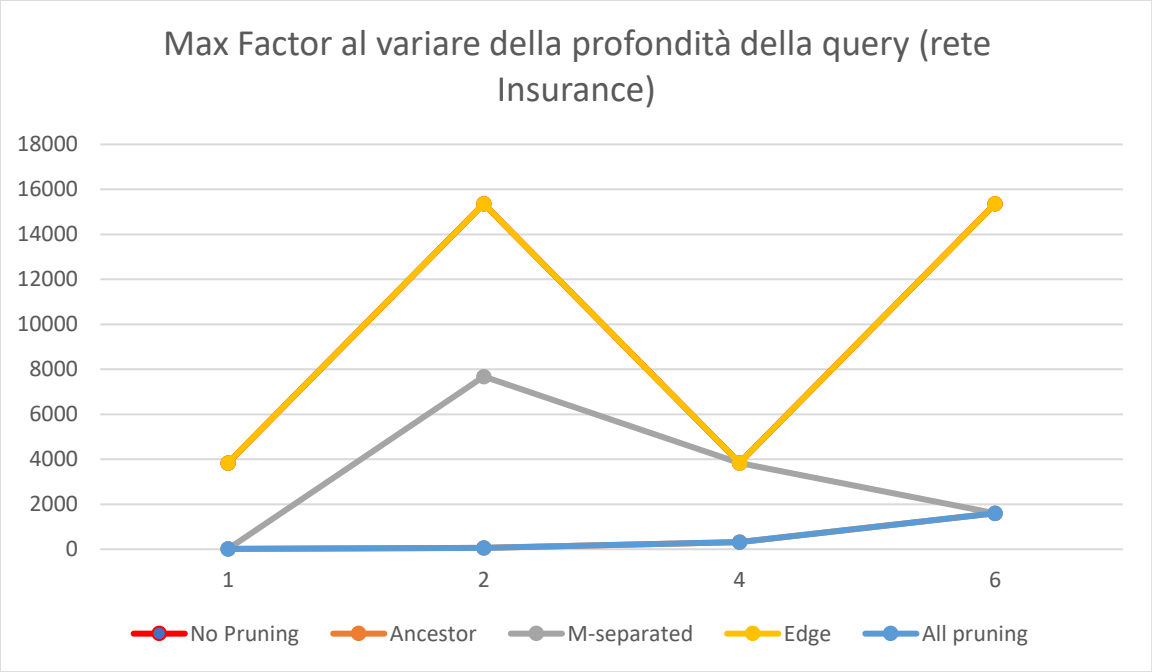
I seguenti esempi considerano il variare dei tempi di esecuzione in base al tipo di pruning effettuato.

Il primo esempio mostra come variano tempo e max factor per ogni Pruning in base al livello di profondità della query data la stessa evidenza.

Si basa sulla rete Insurance vista precedentemente e usa come query le seguenti:

- Livello 1: RiskAversion => P (RiskAversion | Age=Adult)
- Livello 2: HomeBase => P (HomeBase | Age=Adult)
- Livello 4: Theft => P (Theft | Age=Adult)
- Livello 6: PropCost => P (PropCost | Age=Adult)

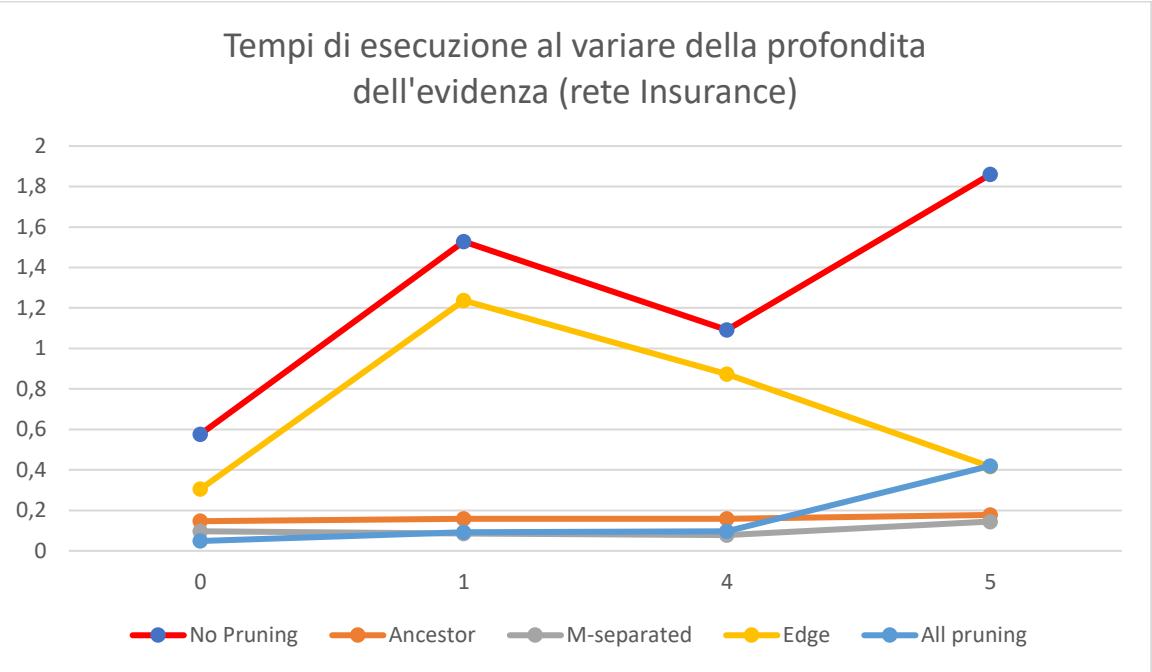


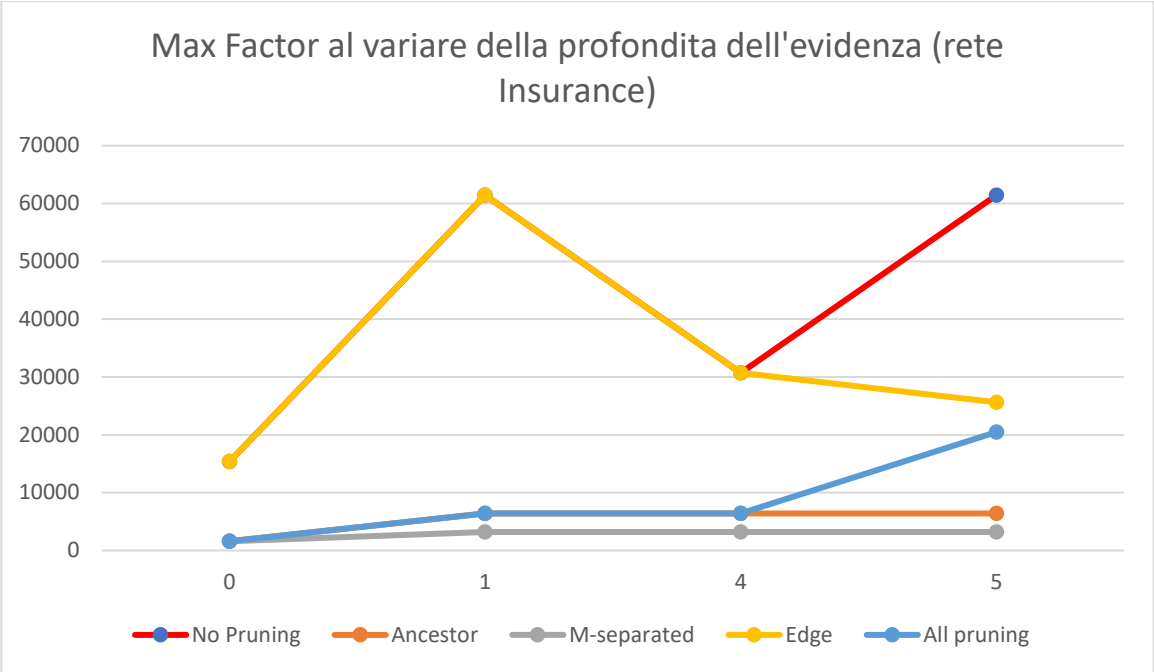


Il prossimo esempio si applica al variare del livello di profondità della evidenza data la stessa query.

Si basa sulla rete Insurance e usa come query le seguenti:

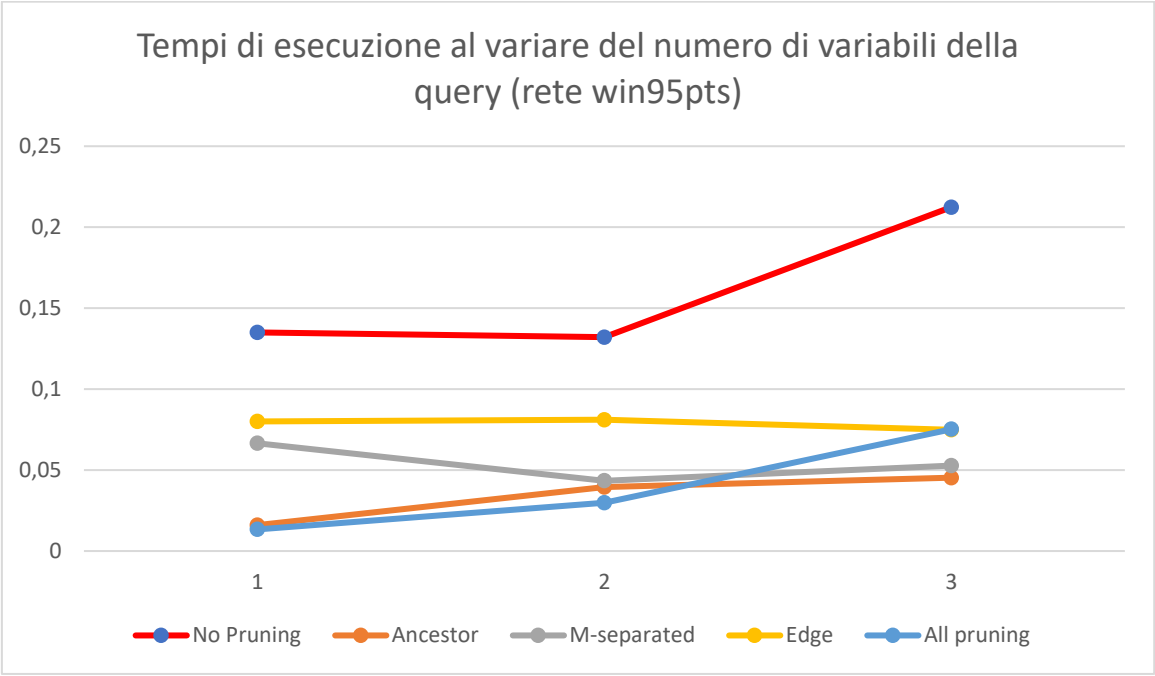
- Livello 0: RiskAversion => P (PropCost | Age=Adult)
- Livello 1: GoodStudent=False => P (PropCost | GoodStudent=False)
- Livello 4: Theft= True => P (PropCost | Theft= True)
- Livello 5: PropCost => P (PropCost | Age=Adult)





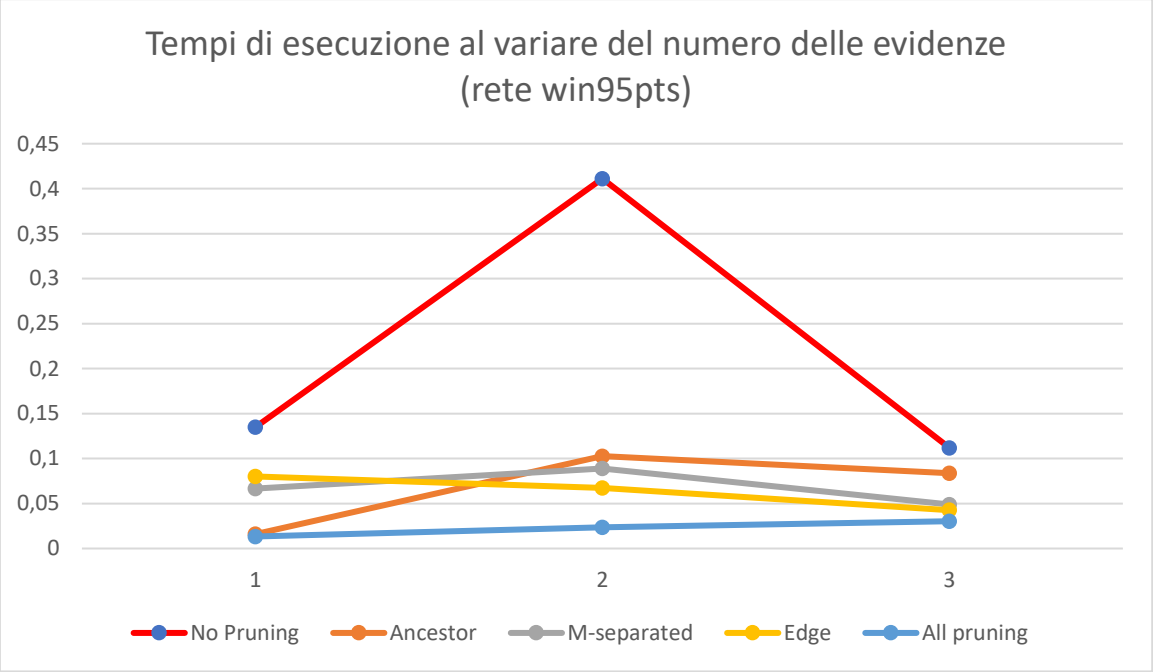
Il seguente esempio si applica al variare del numero di query richieste su rete win95pts (76 nodi):

1) P (PrtData | DataFile=Correct)
2) P (PrtData, NtGrbld | DataFile=Correct)
3) P (PrtData, NtGrbld, DeskPrntSpd | DataFile=Correct)



L’ultimo grafico mostra l’andamento dei pruning al variare nel numero di evidenze su rete win95pts (76 nodi):

1) P (PrtData | DataFile=Correct)
2) P (PrtData | DataFile=Correct, GrbldOtpt = No)
3) P s PrtData | DataFile=Correct, GrbldOtpt = No, GDIOUT = Yes)



Progetto reti dinamiche

RollingUp Filtering

La tecnica RollingUp Filtering nasce come risposta al problema computazionale e di consumo della memoria dell'operare su una rete dinamica con un numero elevato di istanze temporali. Tramite questo filtering riusciamo ad evitare di "srotolare" interamente la rete così da gestire ogni volta solo una coppia di istanti temporali.

Questo filtering lavora appunto su due istanti per volta, inizieremo quindi con uno slice contenente il tempo zero e il tempo uno. Alla successiva iterazione si traslerà lo slice di un istante avendo quindi tempo uno e il successivo, ripetendo tale operazione per le istanze di tempo richieste dal quesito.

Ad ogni iterazione dell'algoritmo sarà applicata la variable elimination per calcolare il risultato dei nodi di interesse dello slice corrente. Tale risultato sarà utilizzato per il calcolo della VE con lo slice successivo.

Dal momento che le reti dinamiche son rappresentate con un solo slice, con i nodi t-1 e t, e una variabile numerica che indica il numero di cicli da effettuare sarà necessario implementare una corrispondenza tra i risultati della VE precedente, calcolata sui nodi rilevanti in t, e i nodi in t-1 dell'iterazione successiva.

Implementazione

Per l'implementazione è stata creata una classe RollingUpFiltering.java la quale riceve in input una rete bayesiana dinamica.

Questa classe offre il metodo **rollUp** che ricevuto in input una lista di evidenze per l'istante in questione e un tipo di ordinamento esegue la nuova versione di Variable Elimination e restituisce il risultato normalizzato.

Come per il ParticleFiltering di Aima, questo metodo dovrà essere invocato all'interno di un ciclo da 0 a N dove N son gli istanti di tempo di cui vogliamo calcolare il risultato.

Dynamic Variable Elimination

Per lavorare su reti dinamiche è stato aggiunto un nuovo metodo di VE anche questo basato sull'algoritmo di Darwiche con due modifiche:

- Riceve un ulteriore input contenente una lista di fattori dell'iterazione precedente da aggiungere in fase di calcolo.
- Restituisce una lista di fattori invece di calcolare un risultato finale, questo perché tale lista ci servirà per l'iterazione successiva.

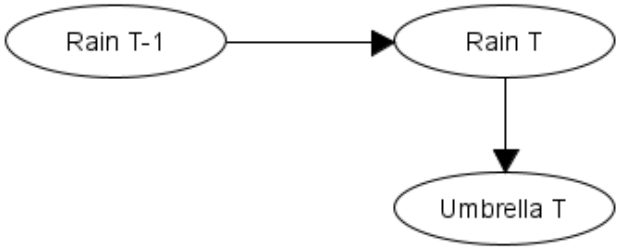
Per quanto riguarda la query da passare al nostro metodo di VE, la classe di Aima "DynamicBayesianNetwork" memorizza in una mappa X0 -> X1 tutti i collegamenti tra un nodo e il suo rispettivo del tempo successivo. Utilizzeremo i nodi contenuti in X1 per crearla automaticamente per ogni rete.

Il risultato della VE è una lista di fattori da utilizzare nell'istante successivo, tuttavia bisogna modificare le variabili di tali fattori affinché nella successiva chiamata non facciano riferimento allo slice uno ma allo slice zero, infatti a seguito della traslazione il tempo a cui son riferiti tali risultati nella successiva iterazione sarà lo slice zero. Il risultato della modifica è memorizzato in una variabile globale e sarà passato in input alla VE del ciclo successivo.

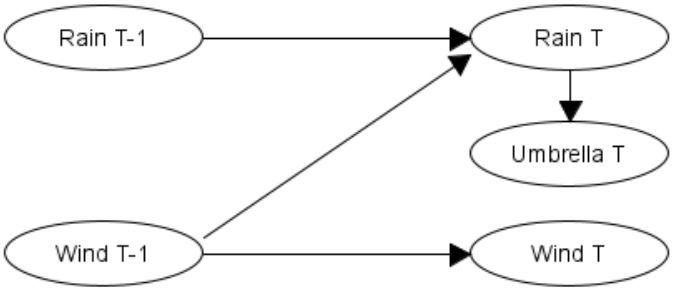
Risultati

Per testare i nostri risultati son state utilizzate le seguenti reti:

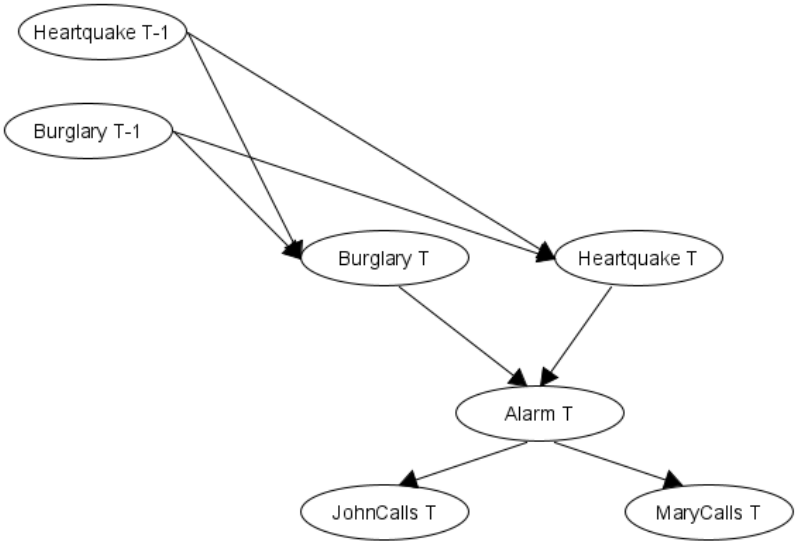
1) Rete Rain



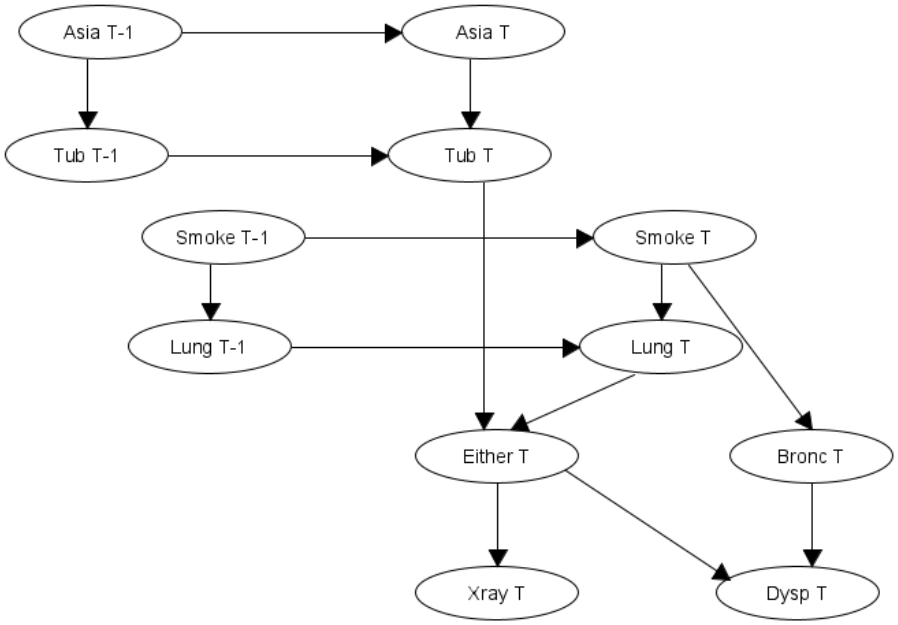
2) Rete Rain-Wind



3) Rete Alarm

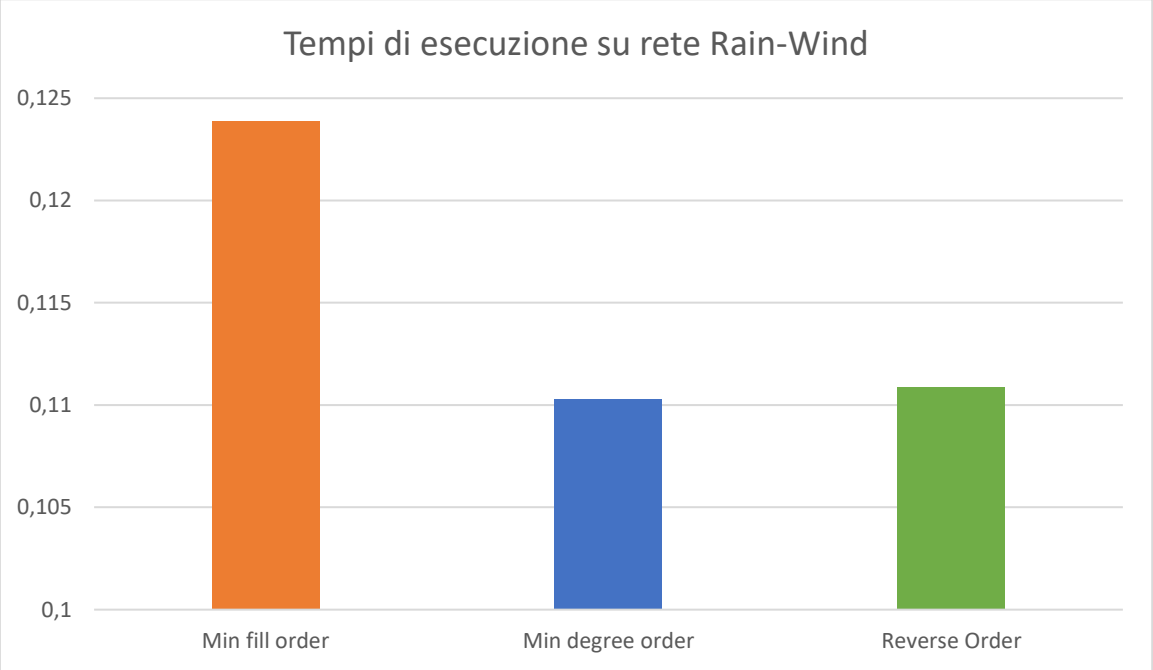


4) Rete Asia

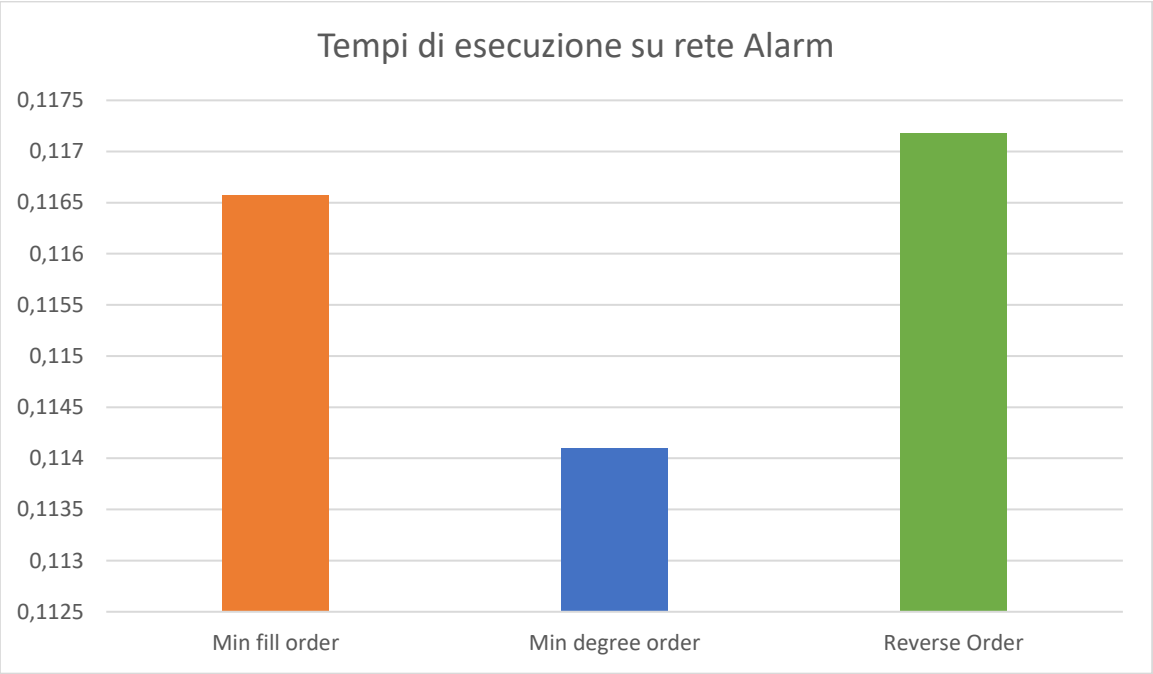


Per i seguenti grafici abbiamo analizzato i tempi del rolling up filtering in base al tipo di ordinamento scelto.

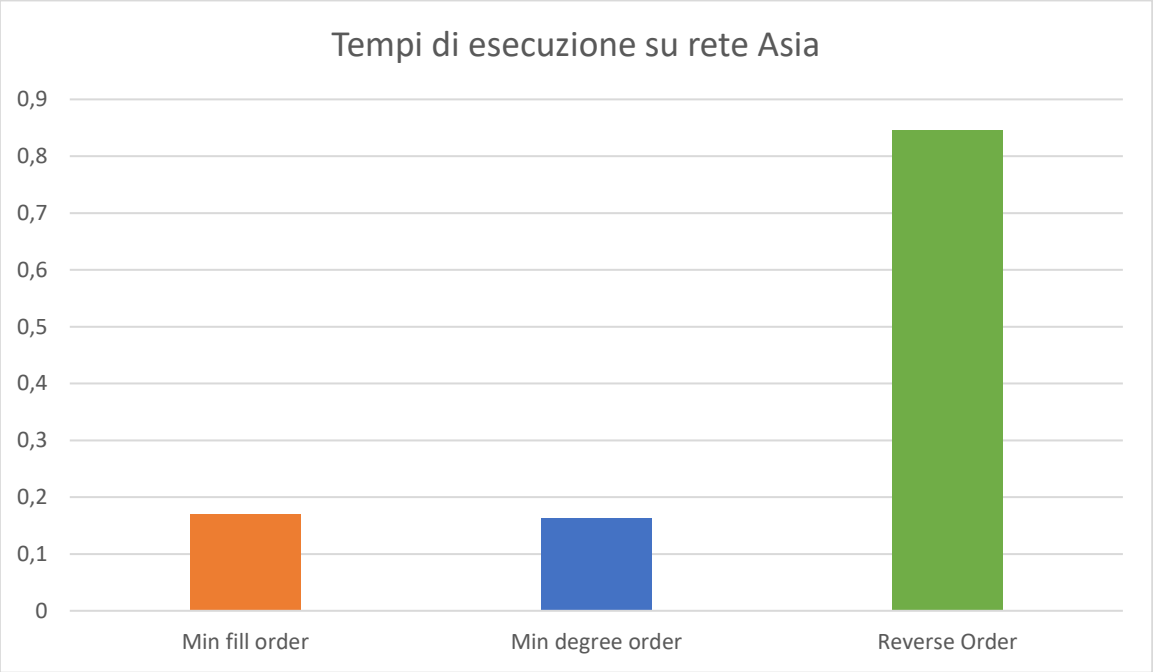
Mappa Rain-Wind:



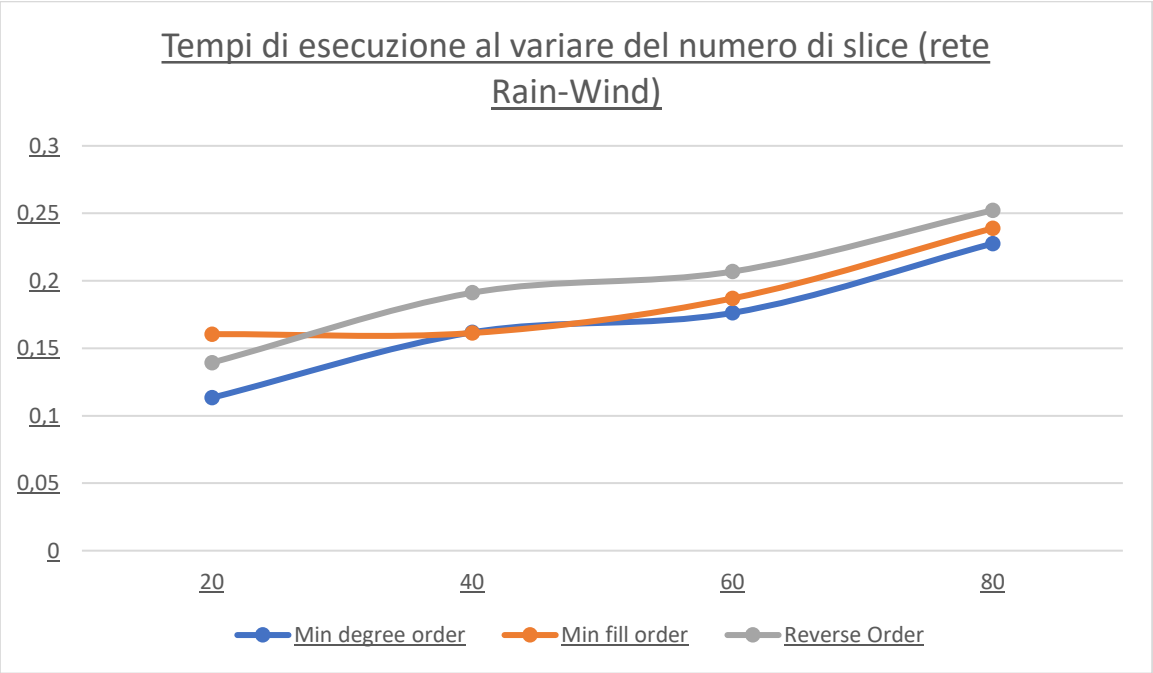
Mappa Alarm:



Mappa Asia:



Infine, nel seguente grafico consideriamo quanto tempo serve a risolvere il filtering in base al numero di istanze temporali nella rete Rain-Wind:



Su reti piccole i tre ordinamenti hanno performance simili, ma al crescere del numero di nodi, il Reverse Order risulta avere tempi notevolmente peggiori. Un discorso analogo può essere fatto sulla crescita degli slice temporali, infatti al crescere degli slice temporali il gap tra gli ordinamenti tende ad aumentare.

L'ordinamento che risulta avere, in generale, un costo computazionale minore e conseguentemente tempi migliori, è il Min Degree Order.

Per il controllo sulla correttezza dell'algoritmo abbiamo effettuato un confronto con i risultati ottenuti usando il ParticleFiltering. Nel caso della mappa rain inoltre son stati confrontati i risultati con le slide viste a lezione per i primi istanti temporali.

Per quanto riguarda il ParticleFiltering si è applicato un numero di predizioni pari a 1000, mentre per il RollingUp abbiamo applicato l'ordinamento min-fill.

Vediamo alcuni esempi:

- Rete Rain-Wind:
  Probabilità sui nodi RainT,WindT data l'evidenza fissa Umbrella = True
  - Con 20 istanti temporali:
    Particle Filtering: < 0.426, 0.043, 0.492, 0.039>
    RollingUp Filtering: <0.417, 0.053, 0.484, 0.046>

  - Con 40 istanti temporali:
    Particle Filtering: < 0.414, 0.049, 0.494, 0.043>
    RollingUp Filtering: <0.417, 0.053, 0.484, 0.046>

- Rete Alarm dopo 20 istanti temporali:
  Probabilità sui nodi Burglary, EarthQuake data l'evidenza fissa MaryCalls = True

- Con 20 istanti temporali:
  Particle Filtering: < 0.214, 0.557, 0.198, 0.031>
  RollingUp Filtering: <0.269, 0.623, 0.095, 0.011>

- Con 40 istanti temporali:
  Particle Filtering: < 0.281, 0.513, 0.187, 0.019>
  RollingUp Filtering: <0.269, 0.623, 0.095, 0.011>

Come vediamo si tratta di risultati molto simili, Il RollingUp tende più velocemente a stabilirsi e notiamo che i risultati di 20 e 40 istanti temporali son sempre gli stessi.