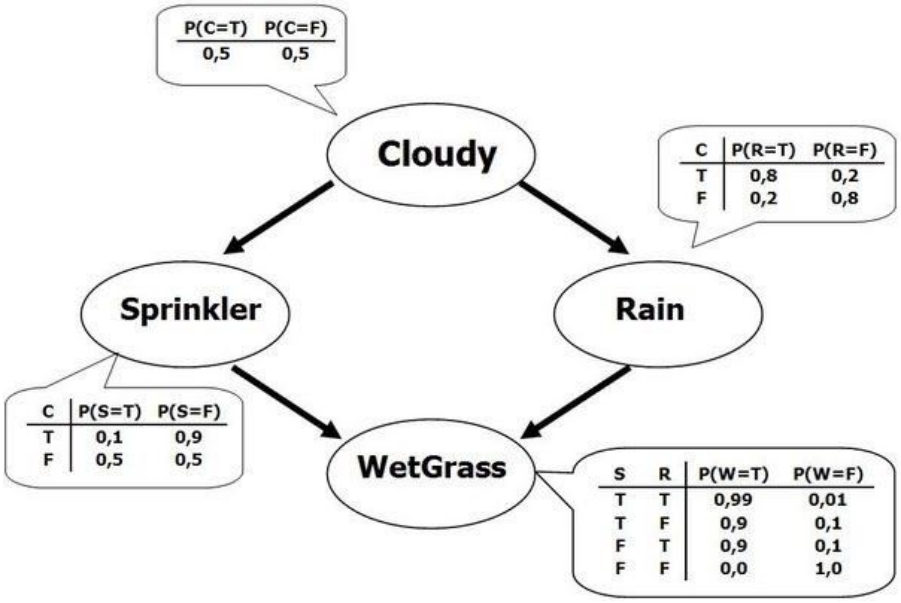
Reti Bayesiane



Il progetto mira a costruire competenze e conoscenze riguardo le Reti Bayesiane. Le Reti Bayesiane sono modelli grafici probabilistici rappresentanti un insieme di variabili aleatorie. La loro peculiarità è rendere visibile graficamente le dipendenze condizionali tra le sue variabili.

Matematicamente, una rete bayesiana è un grafo aciclico orientato in cui:

- I nodi rappresentano le variabili che possono essere osservabili (variabili di evidenza), parametri sconosciuti (hidden) e di query.
- Gli archi rappresentano le relazioni di dipendenza statistica tra le variabili e le distribuzioni locali di probabilità dei nodi figli rispetto al valore dei nodi padre.

Il Progetto

Il progetto richiede l'utilizzo e l'estensione della libreria AIMA-CORE per migliorare l'algoritmo di inferenza esatta Variable Elimination.

Nello specifico le modifiche da applicare sono:

- **Pruning nodi irrilevanti**
  - Pruning nodi non antenati di ({X} U {E}).
  - Pruning nodi m-separati da X tramite E.
- **Pruning archi irrilevanti**: riduzione delle CPT dei nodi contenenti variabili di evidenza E.
- **Ordinamento costruzione dei fattori** seguendo alcune euristiche:

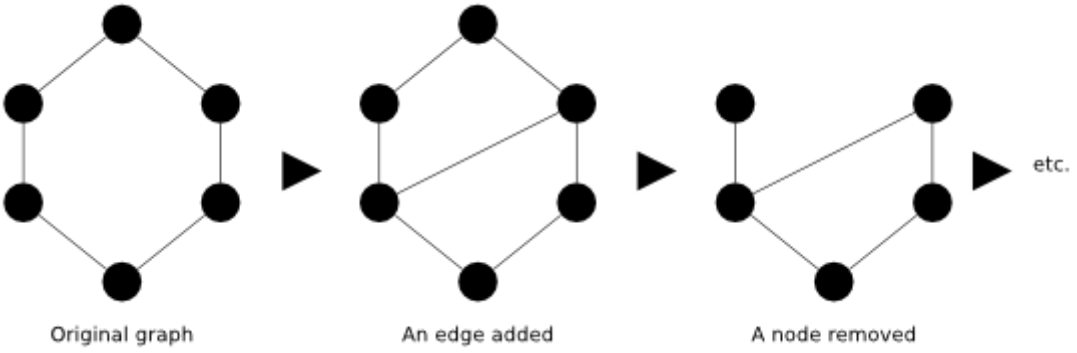
    - MinFill: il costo di un vertice è il numero di archi che devono essere aggiunti al grafo conseguentemente alla sua eliminazione.
    - MinDegree: il costo di un vertice è il numero dei suoi neighbours nel grafo.
    - Reverse: il costo di un vertice è inversamente proporzionale al suo indice nell'ordinamento topologico.

Struttura

- *bayes* contiene tutte le classi che utilizzano ed estendono la libreria AIMA
- *factory* raggruppa le classi che creano le reti bayesiane statiche e dinamiche
- *utils*
    - *bnparser* è il parser che permette di creare oggetti BayesianNetwork a partire da reti in *.xml* o *.xmlbif*
    - *InteractionGraph.java* rappresenta il grafo morale e detta l'ordine delle variabili nell'algoritmo di VE seguendo le euristiche a disposizione

Il progetto utilizza Gradle per l'importazione delle librerie esterne. Seppur poche sono state importate alcune librerie:

- Graphstream, per la visualizzazione del flusso di esecuzione dell'algoritmo di VE sul grafo



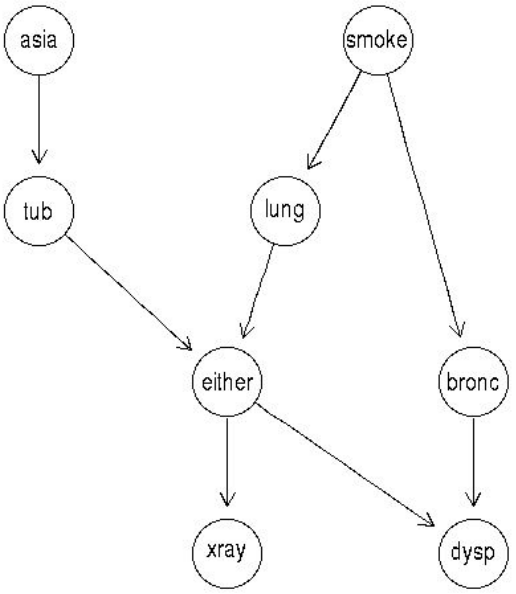
- Protonpack fornisce vari metodi per la gestione di Stream in Java 8

Testing e Risultati Ottenuti

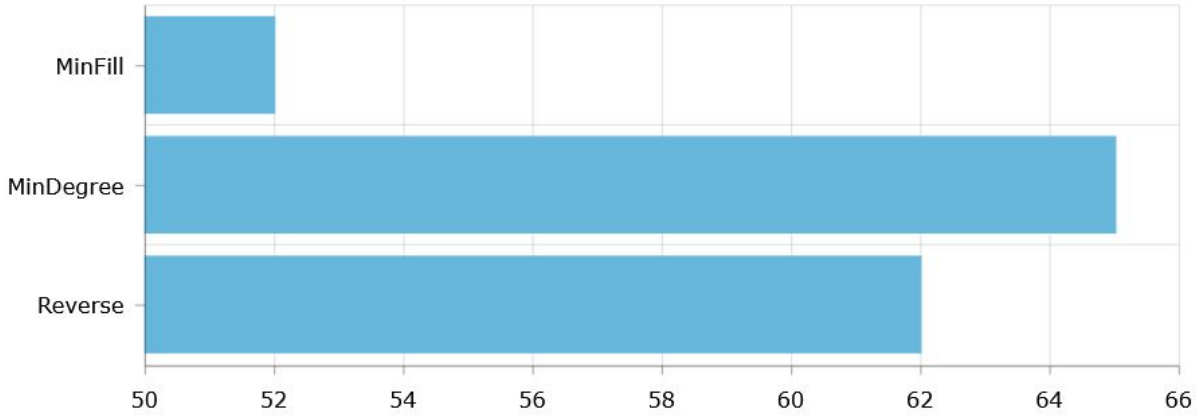
Sono stati effettuati test su varie reti di diversa dimensione, utilizzando diverse variabili di query e di evidenza, in posizioni diverse. Le reti sono prese da BnLearn repository.

I tempi calcolati sono stati misurati in millisecondi (ms) più volte e ne è stata effettuata una media.

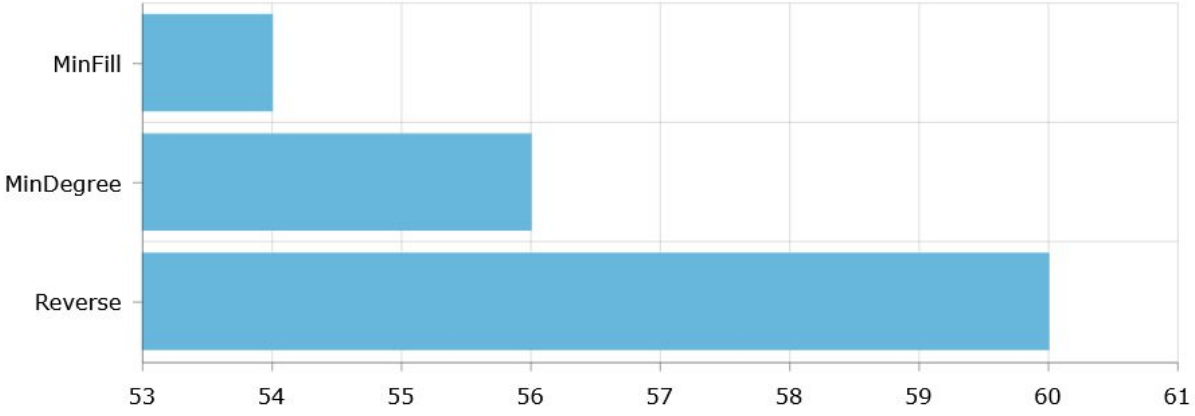
Rete Asia



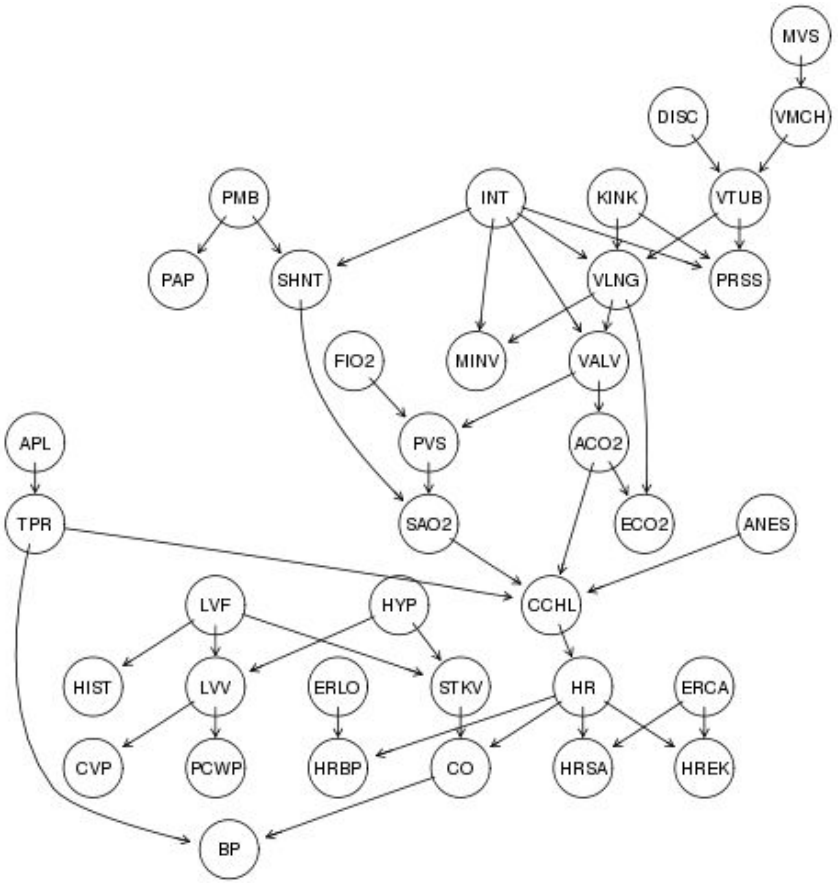
P (Asia | Dysp)



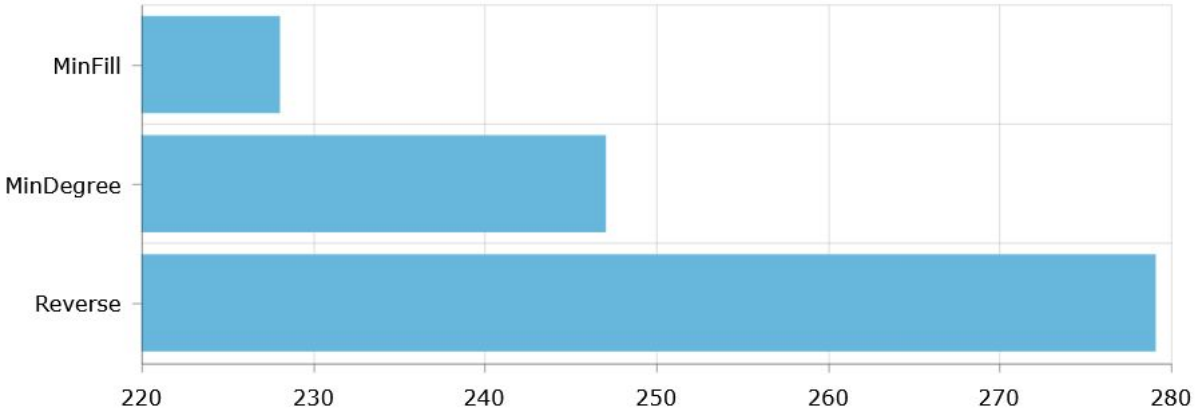
P (Smoke | Dysp, Xray)



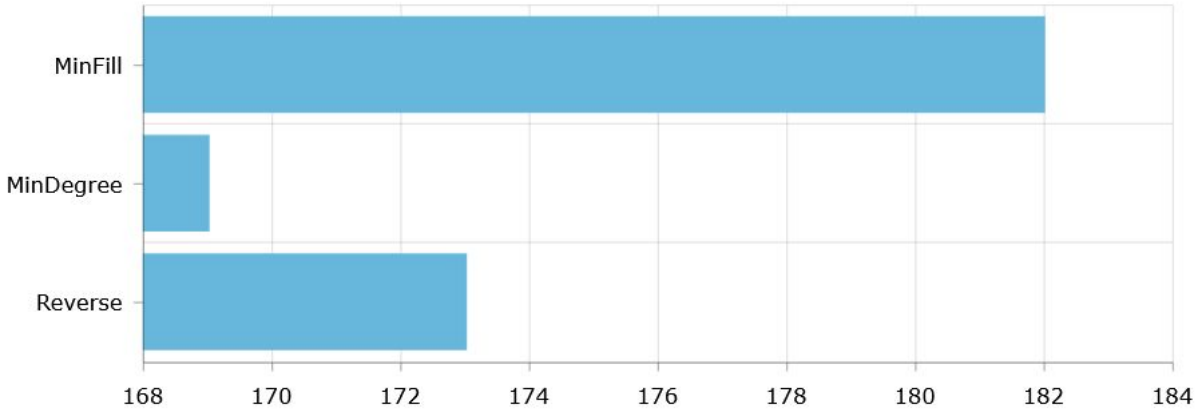
Rete Alarm



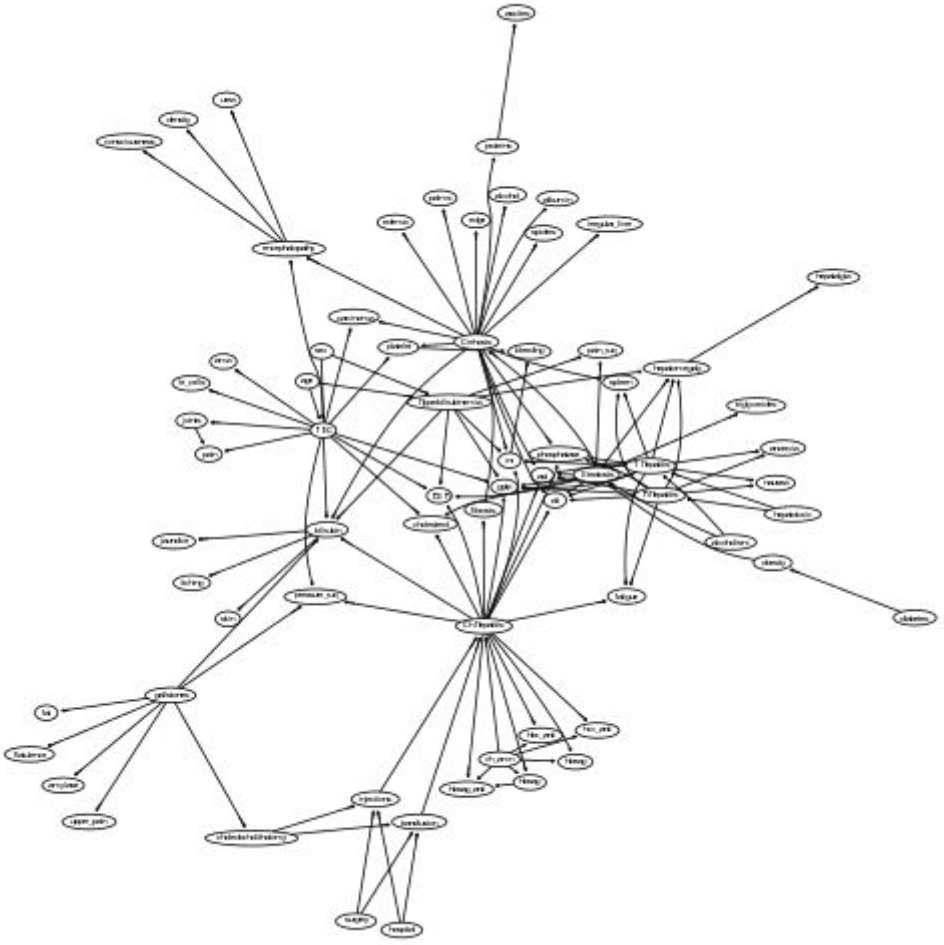
P (FI02 | BP)



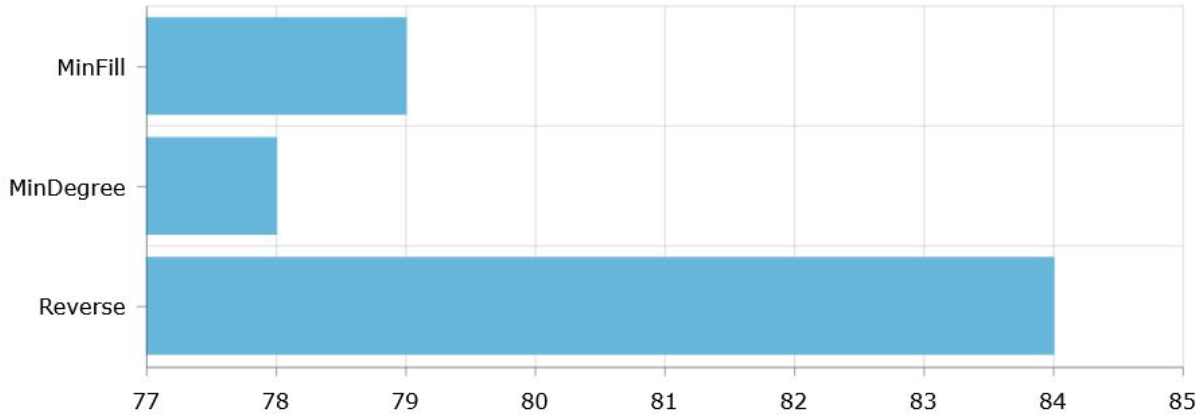
P (FI02, Disconnect | BP, Catechol)



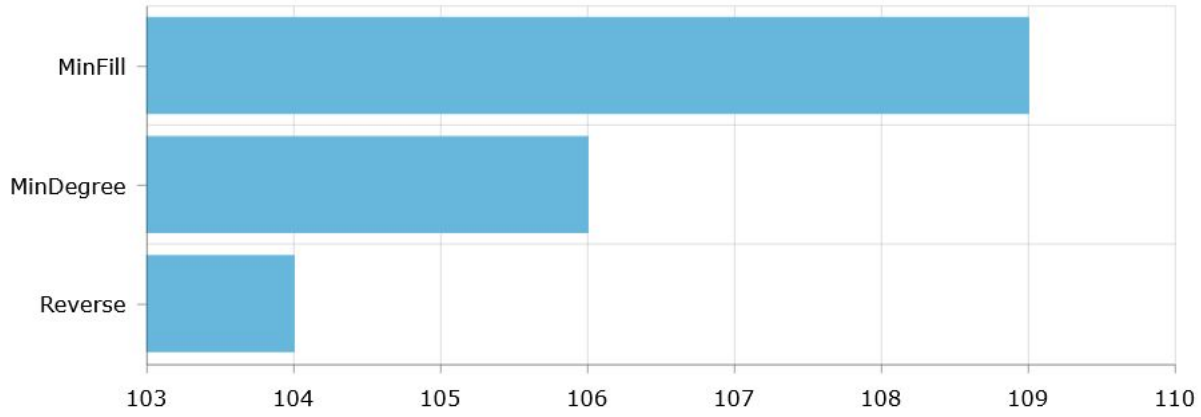
Rete Hepar2



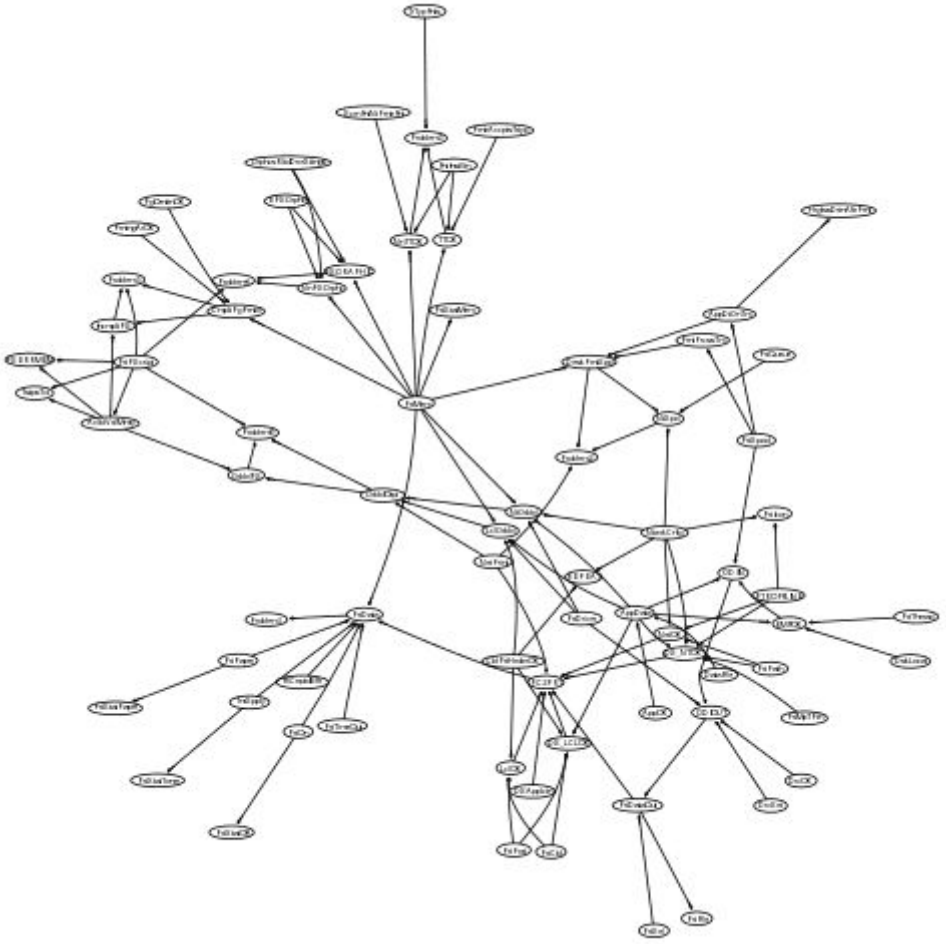
P (Vh_amn | Bleeding)



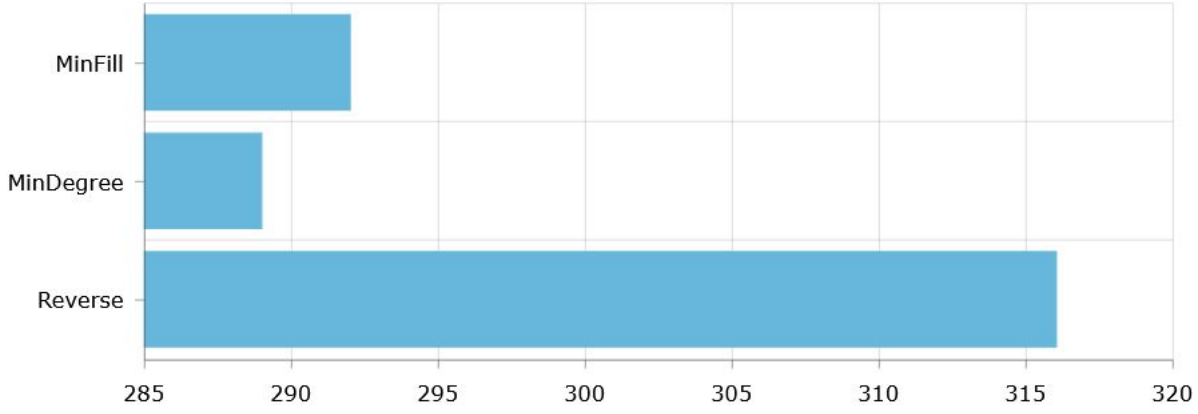
P (Diabetes | Triglycerides && Itching)

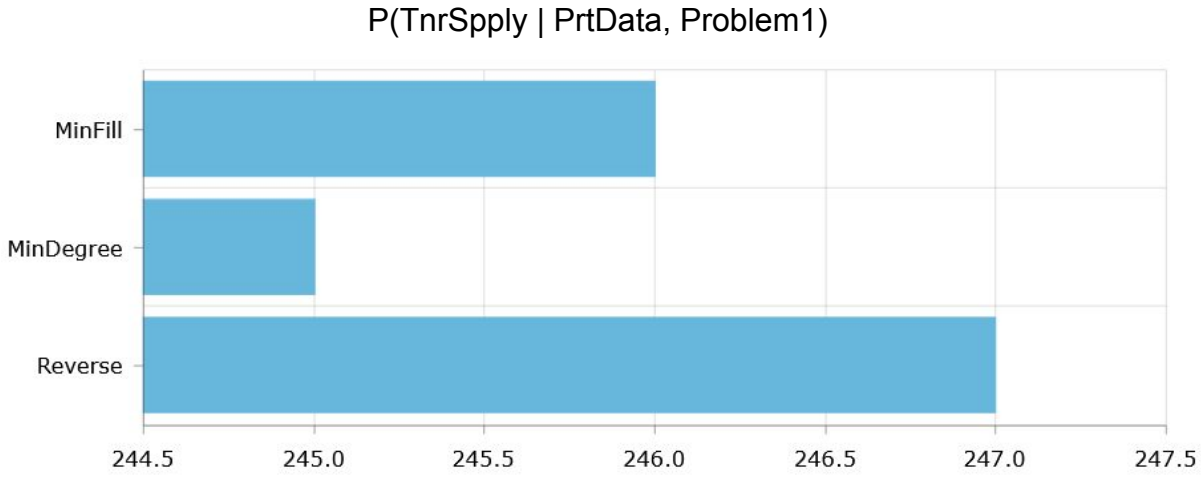


Rete Win95pts



P(TnrSpply | Problem1)





Conclusioni

Dai risultati ottenuti si può vedere che 5 volte su 10 l'ordinamento inverso ha tempi maggiori rispetto ai restanti due.

Poiché il campione di test è esiguo e le differenze sono piccole, per il momento si può ipotizzare che su reti più grandi la disparità possa essere maggiore.

Tuttavia i tempi di query dipendono molto dalla topologia di rete, quindi i risultati ottenuti hanno un peso relativo. Il numero di campioni dovrebbe essere molto più alto per stabilire una correlazione tra ordinamento e tempo di esecuzione dell'algoritmo in maniera significativa.