Reti Bayesiane

Carlo Alberto Barbano, Valentino Di Cianni, Francesco Iodice

1 Introduzione

Lo scopo di questo progetto è implementare alcuni algoritmi che operano su reti bayesiane (BN). In particolare il progetto è diviso in parti: nella prima è esteso l'algoritmo di inferenza Variable elimination con alcune procedure di ricerca euristiche e di pre-processing delle rete. Nella seconda parte è implementato l'algoritmo di Rollup filtering per l'inferenze su reti bayesiane dinamiche (DBN).

2 Variable Elimination

Variable Elimination (VE) è un algoritmo generale di inferenza esatta per modelli probabilistici come le reti bayesiane. Quest'algoritmo consente di effettuare query sul modello, per ottenere la distribuzione di certe variabili aleatorie (VA). Uno dei casi d'uso piú comuni è quello di calcolare la distribuzione di probabilitá di variabili che non sono direttamente osservabili, ad esempio $P(X|E = e)$ dove X ed E sono sottoinsiemi disgiunti di VA, chiamati rispettivamente *variabili di query* ed *evidenza*. L'algoritmo di *Variable Elimination* consente di ottimizzare il calcolo, esprimendo le computazioni ripetute sotto forma di *fattori*. *Variable Elimination* ha una complessità temporale teorica esponenziale, ma in pratica può essere applicato efficientemente a patto di impiegare un ordinamento appropriati. Poichè trovare un ordinamento ottimale per l'eliminazione delle variabili sia effettivamente un problema NP-hard, possono essere impiegate diverse euristiche per ottenere delle soluzioni di ordinamento accettabile. Questa problematica è affrontata in dettaglio nella sezione 2.1.

Lo scopo di questa progetto è di estendere l'algoritmo di *Variable Elimination* , realizzando diverse procedure per ridurre la dimensionalità delle rete attraverso l'utilizzo di diverse tecniche: pruning di archi irrilevanti (sezione 2.2), e pruning di nodi irrilevanti (sezione 2.3).

2.1 Ordinamento dei fattori

L'ordine in cui vengono processati le variabili nell'algoritmo della VE è un fattore che influisce sulla complessità temporale del problema. Di seguito sono illustrati alcuni algoritmi euristici che vanno a trovare un ordinamento delle variabili accettabile, in quanto la complessità per trovare un ordinamento ottimale è NP-Complete e richiederebbe più tempo dell'esecuzione delle VE stessa.

Definizione 2.1 (Interaction graph)**.** Sia $f_1...f_n$ un insieme di fattori. Il grafo di interazione G è un grafo indiretto costruito come segue:

- i nodi di G sono tutte le variabili che appaiono nei fattori $f_1...f_n$
- esiste un arco tra ogni coppia di variabili se esse sono presenti entrambe nello stesso fattore.

Risulta possibile determinare un ordine euristico delle variabili da eliminare durante la VE che si basi sull'interaction graph seguendo diversi approcci. Lo pseudocodice dei due algoritmi implementati *minimum degree* e *minimum fill* è riportato nei seguenti algoritmi:

Algorithm 1 MinDegreeOrder(N, X)

Input: N Bayesian network, X variables in N
Output: an ordering π of variables X
$G \leftarrow$ interaction graph of the CPTs in network N
for $i = 1$ *to number of variables in* X **do**
  $\pi(i) \leftarrow$ a variable X with smallest number of neighbors in G
  add an edge between every pair of non-adjacent neighbors of $\pi(i)$ in G
  delete variable $\pi(i)$ from G and from X
return π

Algorithm 2 MinFillOrder(N, X)

Input: N Bayesian network, X variables in N
Output: an ordering π of variables X
$G \leftarrow$ interaction graph of the CPTs in network N
for $i = 1$ *to number of variables in* X **do**
  $\pi(i) \leftarrow$ a variable X that adds the smalles number of edges in G
  add an edge between every pair of non-adjacent neighbors of $\pi(i)$ in G
  delete variable $\pi(i)$ from G and from X
return π

2.2 Pruning Nodi irrilevanti

Una possibile ottimizzazione per il calcolo della VE si ottiene facendo pruning del grafo di partenza, andando ad eliminare tutti i nodi irrilevanti. Data una BN, per fornire il risultato di un'inferenza, l'algoritmo di VE crea un fattore per ogni nodo presente all'interno della rete. Alcuni nodi irrilevanti possono essere eliminati dalla rete riducendo il numero dei fattori pur mantenendo inalterata la distribuzione di probabilità risultante. In questo progetto, il pruning dei nodi irrelevanti è stato implementato sulla base di due teoremi. Indichiamo con X una variabile di query, con E l'evidenza e con $Ancestor(X)$ tutti i genitori e progenitori della variabile X.

Teorema 2.1. *Data una baeysian network BN, una variabile Y è rilevante se e solo se $Y \in Ancestor(X \cup E)$.*

Prima di enunciare il secondo teorema è necessario definire alcuni concetti complementari.

Definizione 2.2 (Moral Graph)**.** In una di una rete bayesiana un *grafo morale* è un grafo non orientato che si ottiene legando tra loro tutti i genitori che condividono almeno un figlio, tolta la direzionalità degli archi.

Definizione 2.3 (M-Separation)**.** In una di una rete bayesiana due variabili A e B si dicono *m-separate* per una variabile C, se e solo se non esiste un percorso che non contiene C. Due insiemi di variabili $\{A\}$ e $\{C\}$ sono *m-separati* da un insieme di variabili $\{C\}$, se e solo se per ogni coppia di variabili $A \in \{A\}$ e $B \in \{B\}$ non esiste un percorso che non contiene nessuna variabile $C \in \{C\}$.

Di seguito il secondo teorema alla base della rimozione dei nodi irrilevanti. Indichiamo con X la variabile di query e con E l'evidenza.

Teorema 2.2. *Una variabile Y è irrilevante in una rete BN per rispondere ad una query se Y e X sono m-separate da E.*

Il risultato di queste 2 operazioni di pruning è denotato con pruneNodes(BN,e) (Sezione 2.4.1).

2.3 Pruning Archi irrilevanti

Data una rete bayesiana ed una query, è possibile eliminare alcuni archi, e di conseguenza ridurre le entry di alcune CPT, senza intaccare la correttezza del risultato. In particolare, per ogni arco $U \rightarrow X$ che ha origine (è uscente) da un nodo $U \in E$ (gli archi che escono da uno dei nodi di evidenza) è possibile:

1. Rimuovere l'arco $U \rightarrow X$

2. Rimpiazzare la CPT $\Theta_{X|U}$ per il nodo X da una CPT ridotta, ottenuta da $\Theta_{X|U}$ assumendo il valore u del genitore U data l'evidenza e. La nuova CPT corrisponde a $\sum_U \Theta^u_{X|U}$

Questo processo è riassunto dal seguente teorema:

Teorema 2.3 (Pruning archi)**.** *Data una rete bayesiana N ed un'instanziazione di variabili e, denotando con N' la rete bayesiana ottenuta con il pruning degli archi irrilevanti, allora $Pr(Q, e) = Pr'(Q, e)$ dove Pr e Pr' sono le distribuzioni di probabilità indotte da N e N' rispettivamente.*

2.4 Metodo

2.4.1 Implementazione

Dato che nella libreria aima-core molte delle classi relative alla struttura delle BN non permettono la modifica di alcuni campi, sono state implementate le seguente classi per aggirare questa limitazione. In particolare con l'utilizzo delle seguenti classi è possibile eliminare un nodo dalla BN (con il conseguente aggiornamento della CPT del genitore, qualora fosse presente), eliminare archi e cambiare a run-time le CPT dei nodi della rete. Di seguito le classi:

- *FullCPTNodeEd*, simile alle classe *FullCPTNode* con l'unica differenza che associa ad un *FullCPTNode* una *EditableCPT*
- *EditableCPT*, simile alle classe *CPT* con la differenza che permette di eseguire get e set della *ProbabilityTable* associata.
- *EditableNode.java*, simile alle classe *AbstractNode* con la possibilità di eliminare un nodo figlio o un nodo genitore.
- *EditableBayesianNetwork*, simile alla classe *BayesNet* che permette la rimozione di un nodo con il conseguente aggiornamento delle CPT del padre (qualora fosse presente).

Oltre alle classi necessarie per la struttura delle reti sono state implementate anche la classi: *MoralGraph* che, data in input una rete bayesiana, costruisce il relativo grafo morale; e la classe *InterationGraph* che data una lista di fattori costruisce il relativo di grafo di interazione. Per quanto riguarda il parser di reti bayesiane (bifparser) è stata aggiunta la classe *BIFHandlerToEditableBN* che permette di creare da un file XML una rete di tipo *EditableBayesNet*. Dopo aver definito le classi di struttura e di utilità è stata estesa la classe BayesInferences realizzando una nuova classe *CustomEliminationAsk* con tutte le ottimizzazioni sopra elencate che, in base ai parametri in input, ne applica una o più durante il calcolo di VE. Per i test è stato utilizzato il BN-Generator che, in base ai paramatri (Size, MaxDegree, MaxEdges, Types), permette di creare RB casuali utili ai nostri scopi.

2.4.2 Metodologia di sperimentazione

L'algoritmo della VE è stata testato su diverse reti che variano per dimensione e struttura, con combinazioni diverse delle tecniche di ottimizzazione elencate precedentemente. Le architetture di rete sono riportate in tabella 1, dove *size* indica il numero di nodi presenti nella rete, *Max degree* il massimo grado di ciascun nodo, *Max edges* il numero totale massimo di archi e *type* se singly-connected (s) o multy-connected (m).

Tabella 1: Architetture di rete utilizzate

Nome	MaxSize	MaxDegree	MaxEdges	Type
CHAIN	50	1	100	s
S20	20	8	150	s
S50	50	5	36	s
S100	100	5	200	s
S200	200	5	400	s
P20	20	8	83	m

Per ogni architettura è stato testato l'algoritmo di variable elimination, sia nella versione originale sia estesa al pruning dei nodi e archi, e con tutte le euristiche per l'ordinamento topologico implementate.

2.5 Discussione

Sulla rete CHAIN sono stata eseguite le seguenti query:

- $P(X_1|x_{50}) \rightarrow Q1$
- $P(X_1, X_{30}|\neg x_{40}, x_{50}) \rightarrow Q2$
- $P(X_1, X_{25}, X_{50}|x_{20}, \neg x_{30}, x_{40}) \rightarrow Q3$

Tabella 2: Query su CHAIN

Order	MF				MD				TI			
Prune Nodes	v	v	x	x	v	v	x	x	v	v	x	x
Prune Edges	v	x	v	x	v	x	v	x	v	x	v	x
Q1	14826	4352	4055	3710	3432	1993	2241	1625	970	923	910	932
Q2	2828	1307	1839	1504	1324	1531	1715	1439	1582	928	1528	1153
Q3	987	932	7439	1560	1530	1756	2567	2503	985	1162	1232	1715

In tabella 2 sono presentati i risultati sulla rete di tipologia *CHAIN*. I tempi sono riportati in microsecondi. Come si può vedere nella tabella, in una rete a catena, utilizzando l'ordinamento minimum fill (MF) si ottiene un degrado delle prestazioni, in quanto la modalità utilizzata per scegliere l'ordinamento dei fattori dipende dalle relazioni dei fattori nel relativo grafo di interazione. Ad ogni iterazione MF seleziona il nodo con meno vicini, ma in questo esempio specifico risulta trascurabile, in quanto tutti i nodi (eccetto il nodo evidenza e il nodo radice) possiedono tutti lo stesso numero di vicini, dunque è come se la selezione avvenisse in maniera casuale. Lo stesso comportamento viene registrato anche per il minimum degree (MD) in quanto variante di MF. Nelle rete CHAIN l'ordinamento ottimale risulta essere quello topologico inverso (TI), in quanto ogni figlio verrà processato sicuramente prima del proprio padre.

Per quanto riguarda il pruning dei nodi nel caso delle query Q1 e Q2, risulta trascurabile l'apporto dell'ottimizzazione, in quanto tutti i nodi della rete sono necessari per rispondere alla query. Nella query Q3 vengono rimossi 9 nodi dalla rete, portando ad un incremento visibile delle prestazioni. Per questa tipologia di rete l'ottimizzazione più significativa risulta essere il pruning degli archi poiché, eliminando un arco, è come se le rete venisse spezzata in due parti, ottenendo così più indipendenza tra le variabili.

Sulla rete S20 sono stata eseguite le seguenti query:

- $P(N_3|N_1 = S1) \rightarrow Q1$
- $P(N_3, N_8|N_1 = S1, N_{10} = S0) \rightarrow Q2$

Tabella 3: Query su S20

Order	MF				MD				TI			
Prune Nodes	v	v	x	x	v	v	x	x	v	v	x	x
Prune Edges	v	x	v	x	v	x	v	x	v	x	v	x
Q1	1896	1773	1774	2703	1370	1349	1434	1533	3048	3040	2668	1737
Q2	1737	1913	1685	1697	1698	1705	1288	1391	2910	2750	2488	3152

I Risultati su S20 sono presentati in tabella 3. Sia nella query Q1 che nella Q2 il pruning non porta ad un miglioramento significativo. In questa rete il vantaggio maggiore è dato dagli algoritmi di ordinamento, poiché in una rete con un grado massimo pari a 8, c'è molta dipendenza tra le variabili, e quindi i criteri di ordinamento MF e MD portano a risultati migliori rispetto all'ordinamento TI.

Sulla rete S60 sono stata eseguite le seguenti query:

- $P(PT|vh) \rightarrow Q1$
- $P(CL, PAP|vg, hg) \rightarrow Q2$
- $P(CL, PAP, CO|vg, hg) \rightarrow Q3$

Tabella 4: Query su S50

Order	MF				MD				TI			
Prune Nodes	v	v	x	x	v	v	x	x	v	v	x	x
Prune Edges	v	x	v	x	v	x	v	x	v	x	v	x
Q1	151	190	3037	2912	184	273	717	622	296	210	63284	69282
Q2	785	1090	1732	1892	403	415	592	743	1517	1192	19760	25526
Q3	879	717	1215	1479	344	395	648	828	1147	1089	2311	1953

I risultati sulla rete S50 sono presentati in tabella 4. Sulla query Q1 è stato eseguito il pruning di 26 nodi, che si traduce in performance temporali significativamente migliori quando il pruning è attivo. Anche il pruning degli archi comporta un vantaggio, ma poco significativo se paragonato al pruning dei nodi, in quanto un solo arco viene rimosso. Nella query Q2 invece, il vantaggio portato dal pruning dei nodi risulta meno significativo, in quanto vengono eliminati solo 13 nodi irrilevanti.

Sulla rete S100 sono stata eseguite le seguenti query:

- $P(N_{80}|N_{56} = S1) \rightarrow Q1$
- $P(N_{130}, N_{180}|N_1 = S1, N_{100} = S0) \rightarrow Q1$

Tabella 5: Query su S100

Order	MF				MD				TI			
Prune Nodes	v	v	x	x	v	v	x	x	v	v	x	x
Prune Edges	v	x	v	x	v	x	v	x	v	x	v	x
Q1	128	37	2652	2079	34	35	2101	2128	23	23	33410	33400
Q2	51	40	2036	1978	38	34	2196	2323	28	33	20058	19850

Anche in queste rete valgono le stesse considerazioni fatte sulle tabelle 4 e tabella 3, ma il miglioramento è decisamente più significativo viste le dimensioni maggiorate della rete.

Sulla rete S200 sono stata eseguite le seguenti query:

- $P(N_{80}|N_1 = S1)$
- $P(N_{30}, N_{80}|N_1 = S1, N_{10} = S0)$
- $P(N_1, N_{10}|N_30 = S1, N_{80} = S0)$

Tabella 6: Query su S200

Order	MF				MD				TI			
Prune Nodes	v	v	x	x	v	v	x	x	v	v	x	x
Prune Edges	v	x	v	x	v	x	v	x	v	x	v	x
Q1	54	34	5638	5526	53	32	5779	5865	24	92	102087783	100860130
Q2	58	41	4905	4697	64	44	9031	5478	33	52	100824056	100588860
Q3	109	55	6124	6423	77	61	7167	6588	30	40	103511410	217710427

I Risultati sulla rete S200 sono presentati in tabella 6. Questi utlimi sono paragonabili ai risultati ottenuti sulla rete S100. É interessante osservare come l'ordinamento topologico peggiori esponenzialmente senza pruning dei nodi.

Sulla rete P20 sono stata eseguite le seguenti query:

- $P(N_{10}|N_{17} = S1)$
- $P(N_{16}, N_8|N_{12} = S1, N_2 = S0)$

I relativi risultati sono presentati in tabella 7. In una rete multi connected il degrado della performance della VE, senza ottimizzazioni, è evidente, pur essendo una rete di dimensioni contenute. Nel caso della Q1, su un totale di 20 nodi, 16 vengono eliminati e questo porta ad un'esecuzione

Tabella 7: Query su M20

Order	MF				MD				TI			
Prune Nodes	v	v	x	x	v	v	x	x	v	v	x	x
Prune Edges	v	x	v	x	v	x	v	x	v	x	v	x
Q1	158	165	194958	125049	151	152	32713	47295	124	333	64265	32392
Q2	20729	20063	32912	46815	17308	16604	11749	8707	10973	11133	12245	21521

molto veloce della VE, riducendo le l'ordine di grandezza delle tempistiche di tre volte. Sulla query Q2, invece, i nodi eliminati sono solo 3 e le performance non ottengono vantaggi consistenti. Gli algoritmi di ordinamento migliorano nella maggior parte dei casi le performance della VE, anche senza ottimizzazioni.

In conclusione, osservando i risultati ottenuti dai nostri esperimenti, il pruning dei nodi sembra essere l'operazione che porta in media a un vantaggio più significativo sulla performance di VE, anche nel caso di reti multi connesse, anche se a volte il pruning degli archi e dei nodi in combinazioni con gli ordinamenti euristici porta ad un degrado delle performance: questo potrebbe essere dovuto al fatto che il pruning influisce sull'ordinamento delle variabili.

3 Rollup Filtering

Le rete bayesiane dinamiche (DBN) sono reti bayesiane che mettono in relazione variabili tra istanti di tempo successivi. Vengono utilizzate per modellare serie temporali e poichè sono una generalizzazione degli hidden Markov model e dei Kalman filter, ricadono sotto l'assunzione markoviana. Per questo sono spesso chiamate 2TBN (two-time slice bayesian network).

Come per gli altri modelli temporali, una delle operazioni di inferenza possibili su questo tipo di reti è il filtering, ovvero: $P(X_t|E_{1:t})$, dove X_t rappresenta le variabili di stato e $E_{1:t}$ la serie di evidenze nell'intervallo temporale $[1 : t]$. Sarebbe possibile eseguire questo tipo di operazione tramite unrolling completo della rete, considerando la rete ottenuta come un BN statica e facendo un'operazione di VE. Questa operazione però non sarebbe ottimale dal punto di vista del tempo e dell'utilizzo delle risorse; per questo motivo viene spesso utilizzata una tecnica che prende il nome di *Rollup Filtering*: a differenza dell'unrolling completo, con questo approccio si considera la rete in un dato istante (*slice*). Grazie a questa ottimizzazione, le performance del filtering migliorano sensibilmente, come spiegato nel dettaglio nella sezione 3.2.

3.1 Metodo

Per l'implementazione del Rollup Filtering abbiamo deciso di partire da quanto sviluppato con la prima parte del progetto sulle reti bayesiane statiche, riutilizzando frammenti di codice già scritto in precedenza per la creazione di reti dinamiche. Dal punti di vista tecnico, per la creazioni delle rete dinamiche, abbiamo definito una nuova classe java *CustomDBN* che estende la classe *EditableBayesianNetwork* per poter sfruttare le proprietà dei nodi editabili già sviluppati. Abbiamo infatti deciso di non sfuttare la classe fornita da Aima per l'implementazione delle reti dinamiche, rendendo invece dinamiche le reti statiche su cui avevamo lavorato in precedenza, con l'aggiunta di mappe di transizione per indicare le relazioni delle variabili di stato tra istanti di tempo successivi. Inoltre, dal momento che la rete in un dato istante rimane una rete bayesiana statica, non abbiamo avuto bisogno di implementare una nuova versione dell'algoritmo di variable elimination, ma abbiamo potuto sfruttare quella già precedentemente implementata.

Per quanto riguarda l'aspetto temporale delle reti, abbiamo implementato una funzione di forward che fa avanzare lo stato interno della rete andando ad agire sulle prior belief dei nodi root ad ogni passo, basandosi sui fattori ottenuti dal passo precedente.

Dinamizzazione di una rete statica Per ottenere una rete dinamica a partire da una rete statica, l'approccio che abbiamo seguito è il seguente:

1. Viene creata una rete statica N
2. I nodi radice $\{R\}$ di N sono considerati variabili di stato all'istante t.

3. Sia $\{S\}$ l'insieme dei nodi figli di $\{R\}$: ogni nodo $R \in \{R\}$ viene associato a un nodo $S \in \{S\}$, che rappresenterà la corrispondente variabile di stato all'istante $t+1$. Ogni nodo $S \in \{S\}$ è associato al più a un nodo $R \in \{R\}$.
4. Le variabili di evidenza sono costituite dalle variabili dei nodi foglia di N.
5. Le variabili hidden sono costituite dai nodi genitori $\{H\}$ di $\{S\}$, tali che $\{H\} \cap \{R\} = \emptyset$.

Avanzamento di una rete dinamica La procedura *forward* è riassunta di seguito:

i Vengono calcolati i fattori f_{X_t} per ogni variabile di stato X_t, e i fattori f_{E_t} per ogni variabile E_t di evidenza

ii Per ogni variabile di stato X_t, se esiste un fattore $f_{E_t}(X_t)$, allora $f_{X_t} \leftarrow f_{X_t} \cdot f_{E_t}(X)$

iii I fattori risultanti f_{X_t} sono usati per aggiornare le distribuzioni prior delle variabili di stato X_{t+1}.

3.2 Discussione

Tabella 8: Architetture di DBN utilizzate

Nome	N. State	N. hidden	N. evidence	Type
Umbr	1	0	1	*s*
WUmbr	2	0	1	*s*
S20	3	8	1	*s*
S100	34	12	34	*s*
S200	64	30	58	*s*
M20	2	11	3	*m*

In tabella 8 sono riportate le architetture di reti dinamiche utilizzate. Il filtering è stato valutato su una successione di 50 istanti, e per ogni istante le evidenze sono state generate in maniera casuale. Per ogni rete abbiamo misurato il tempo impiegato per effettuare un filtering $P(X_t|E_{1:t})$, dove X_t rappresenta le variabili di stato al tempo t e $E_{1:t}$ la successione delle evidenze raccolte dal tempo 1 al tempo t, sia in versione *naive* (unrolling completo della rete) nell'intervallo temporale $[1;t]$ sia in versione rollup filtering, assumendo lo stato della rete X_{t-1} e partendo dalle evidenze E_t. Inoltre, come effettuato in sezione 2.5, per ogni rete sono stati provati diversi ordinamenti topologici, quali: topologic inverse (*TI*), minimum degree (*MD*) e minimum fill (*MF*).

Tabella 9: Risultati Rollup filtering

Order	**TI**		**MD**		**MF**	
Nome	**t. un**	**t. up**	**t. un**	**t. up**	**t. un**	**t. up**
Umbr	10	1	10	1	10	1
WUmbr	19	1	17	1	17	1
S20	397	3	364	3	460	3
S100	151	4	166	4	154	4
S200	600	8	495	7	325	5
M20	5434	106	3126	56	3272	61

In tabella 9 sono riportati i risultati, misurati in millisecondi, per le diverse combinazioni di rete e ordinamento, dove *t. un* indica il tempo impiegato per effettuare un unroll completo e *t. up* il tempo impiegato con il Rollup Filtering per rispondere alla query $P(X_t|E_{1:t})$. Dai risultati è evidente come un algoritmo basato sulla programmazione dinamica come Rollup Filtering possa portare a un vantaggio nell'inferenza delle reti bayesiane dinamica in termini di tempo. Il metodo naive di unrolling completo è, infatti, assolutamente non necessario - se non per ragioni di comprensione - poiché per effettuare l'inferenza al tempo t, è sufficiente conoscere lo stato della rete al tempo $t-1$: come anticipato nell'introduzione alla sezione 3 questo avviene per due motivi:

- Nei processi stocastici modellati dalle reti bayesiane vale l'assunzione di indipendenza Markoviana

- I processi stocastici modellati dalle reti bayesiane sono stazionari, cioè $P(X_t|X_{t-1}) = P(X_j|X_{j-1})$ $\forall t, j$. Questo descrive in effetti come la dinamica del sistema sia costante nel tempo.

Cambiando il criterio di ordinamento, non abbiamo riscontrato differenze significative nei tempi di esecuzione di rollup filtering. In generale, guardando ai risultati, è chiaro come facendo facendo rollup filtering si ottenga un grande vantaggio in termini di tempo, ma il vantaggio maggiore si ottiene a livello spaziale: infatti lo spazio in memoria utilizzato risulta drasticamente ridotto.

Prendiamo come esempio una semplice markov chain $\{X_t\}_{t\geq 0}$. Allora

$$P(X_{t+1}) = \sum_{x_t} P(X_{t+1}|x_t)P(x_t) \tag{1}$$

Effettuare un unrolling completo, corrisponderebbe a esplicitare $P(X_k)$ ricorsivamente per ogni $k < t$. Ne risulterebbe:

$$\begin{aligned} P(X_{t+1}) &= \sum_{x_t} P(X_{t+1}|x_t)P(x_t) \\ &= \sum_{x_t} P(X_{t+1}|x_t) \sum_{x_{t-1}} P(x_t|x_{t-1})P(x_{t-1}) \\ &= \sum_{x_t} P(X_{t+1}|x_t) \sum_{x_{t-1}} P(x_t|x_{t-1})P(x_{t-1}) \\ &\quad \dots \sum_{x_0} P(x_1|x_0)P(x_0) \end{aligned} \tag{2}$$

Questo corrisponderebbe a memorizzare la CPT di ogni stato X_t, aumentando inoltre lo spazio richiesto con il passare del tempo. É facile intuire come questo problema possa essere ancora più evidente su reti più complesse. Se da una parte, facendo unrolling completo della rete, lo spazio da considerare è pari alla sommatoria per ogni istante di tempo della dimensione della rete più la dimensione degli archi di transizione, dall'altra dobbiamo considerare solo lo spazio necessario ad immagazzinare i dati di uno slice singolo della rete. Inoltre, sfruttando il garbage collector di java, è possibile liberare le parti di memoria occupate dalle cpt dei fattori ad ogni iterazione. Tutto questo si traduce in una efficenza nettamente superiore.