Regole di Associazione: algoritmi

di Rosa Meo

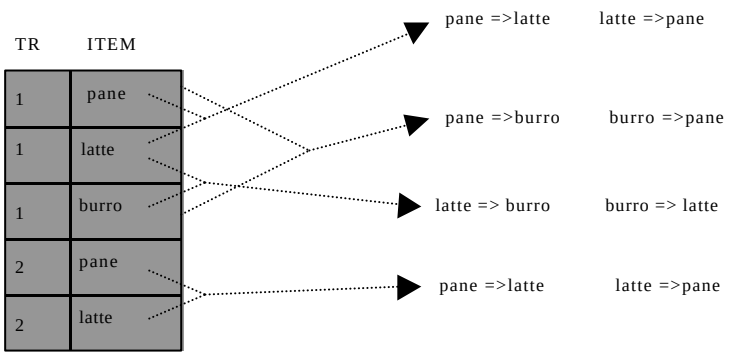
Regole di Associazione

Originariamente proposte nella *market basket analysis* per rappresentare le *regolarità* di comportamento nell'acquisto dei clienti.

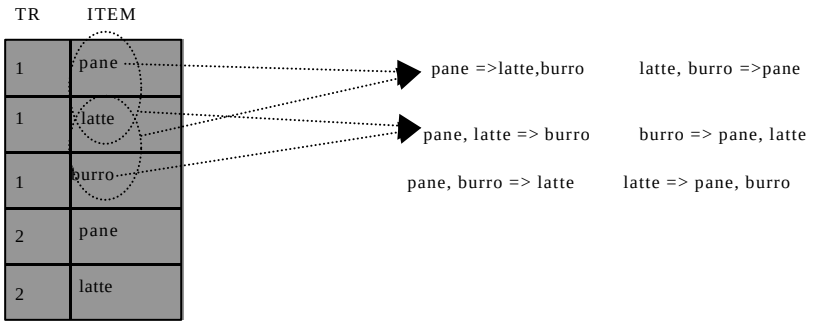
R.Agrawal, T.Imielinski, A.Swami, *Mining Association Rules between Sets of Items in Large Databases*, SIGMOD 1993.

Definizione:
Una regola di associazione è una coppia ordinata di due insiemi di dati, X e Y, estratti da un transazione del database
X={pane} => Y={latte}

Esempio



Esempio



Misure statistiche

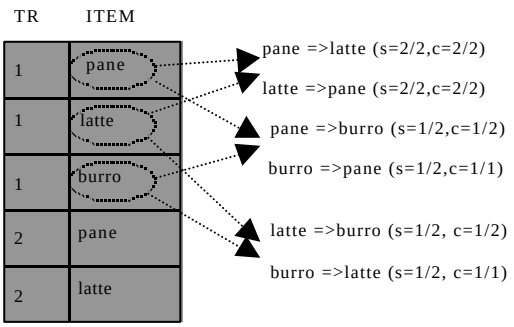
Per estrarre le regole che sono piu' frequenti e significative:

supporto= percentuale di transazioni con X e Y;

confidenza=percentuale delle transazioni con X che hanno anche Y.

L'utente/analista stabilisce due valori minimi.

Esempio



Applicazioni

studio delle abitudini di acquisto;
studio della variabilità delle vendite in assenza di un certo prodotto;
promozione di prodotti, pubblicità mirata;
organizzazione della merce sugli scaffali;
aumento della soddisfazione del cliente;
analisi finanziaria.

Soluzioni proposte

Estrazione di regole di associazione in presenza di:
gerarchie merceologiche;
attributi quantitativi;
condizioni specifiche;
comportamento ciclico nel tempo;
ecc...

Tecnologie adottate

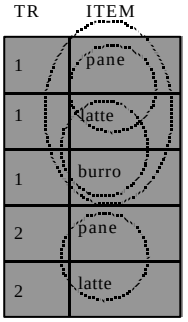
Sequenziale
Con partizionamento della base dati
Con campionamento
Parallela
Incrementale

Algoritmi

Lavorano in due fasi:

1. Individuano tutti gli *itemset* a supporto sufficiente.

Esempio: {pane, latte, burro}



Algoritmi

Lavorano in due fasi:

2. Individuano tutte le regole a confidenza sufficiente suddividendo un itemset Z individuato nella fase 1 in due sottoinsiemi complementari rispetto ad Z.

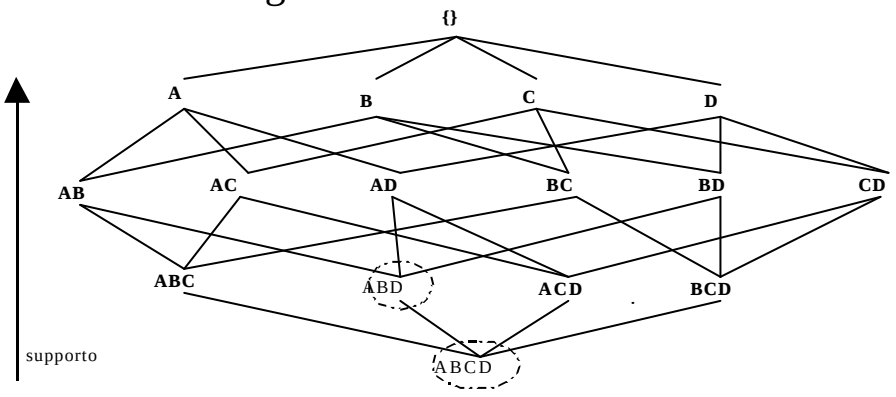
Esempio: Z={pane, latte, burro}

{pane, latte}=> {burro} ,{pane}=> {latte,burro}

Principio di base

Tutti i subset di un itemset a supporto sufficiente (detto *large*) sono itemset large.

Tutti i superset di un itemset che non e' large non sono large.



Fase 2

Dato un itemset Z, e un suo sottoinsieme (proprio) X si individua la regola X=>Z-X.

Il supporto è sufficiente per definizione.

La confidenza viene calcolata come: supporto(Z)/supporto(X).

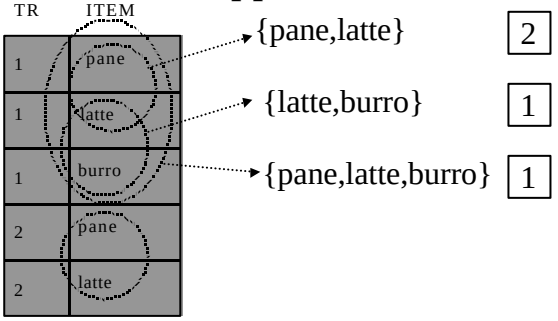
Il processo inizia dai sottoinsiemi X a cardinalitá massima (e quindi con cardinalitá minima della head)

e procede riducendo la cardinalitá di X fino a che la regola ottenuta ha confidenza non sufficiente.

Algoritmi per la fase 1

Fanno almeno una passata sul database.

Allocano un contatore per ciascun itemset al fine di valutarne il supporto.

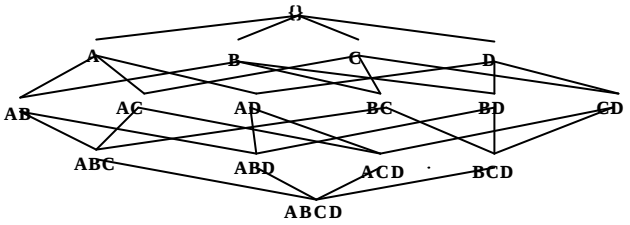


Algoritmi per la fase 1

Allocazione dei contatori

Problema: la memoria principale non è sufficiente per tenere i contatori di tutti i possibili itemset.

Soluzione: si lavora iterativamente e in ciascuna iterazione si considerano solo alcuni itemset.



Tecnologie adottate

Sequenziale

Con partizionamento della base dati

Con campionamento

Parallela

Incrementale

Algoritmo originario (I)

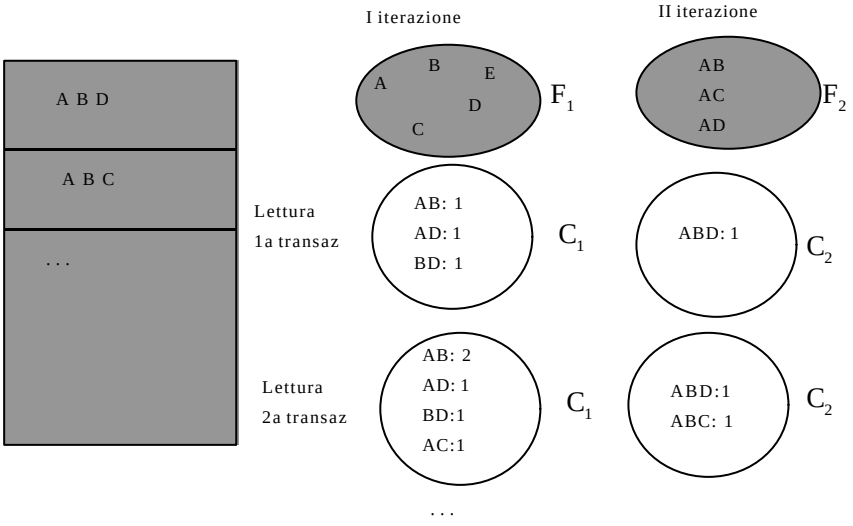
L'algoritmo fa molteplici iterazioni di lettura sulla base dati;

durante l'iterazione k aggiorna il supporto di un insieme C_k di itemset candidati.

Gli itemset in C_k sono determinati da F_k, insieme degli itemset di *frontiera* del passo k.

F_1 contiene dapprima itemset a cardinalitá 1 (singleton) che in ciascuna successiva iterazione k vengono estesi: alla prima iterazione dai sngleton si ottengono le coppie, alla seconda dalle coppie le triplette, ecc...

Esempio



Algoritmo originario (II)

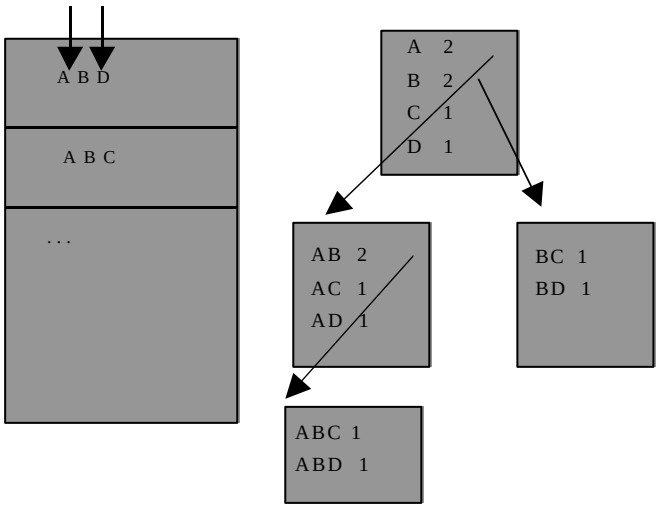
Per ciascuna transazione *t* del db, e per ciascun itemset *i* in F_k, se *t* contiene *i* estendi *i* con item di *t*; gli itemset ottenuti per estensione di *i* sono aggiunti a C_k e se ne mantiene il supporto con un contatore;

alla fine dell'iterazione gli itemset in C_k che risultano davvero large vengono aggiunti a L_k;

aggiorna F_{k+1} con gli itemset di L_k;

ripete una nuova iterazione se $F_{k+1} <> ø$.

Esempio



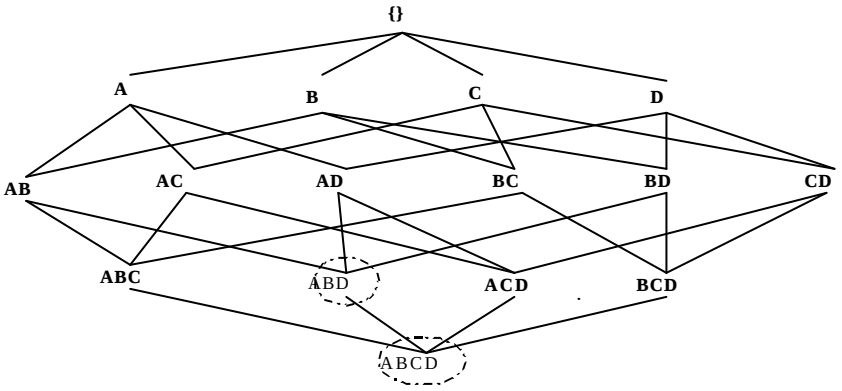
Apriori

L’iterazione k considera come candidati solo itemset a cardinalita’ k+1 (C_{k+1}). Quelli che si dimostreranno large daranno luogo a L_{k+1}.

Ciascun itemset candidato in C_k viene determinato in anticipo rispetto alla lettura del database. Un candidato in C_k si ottiene con il join di due (k-1)-itemset in L_{k-1} aventi lo stesso prefisso di lunghezza (k-2). Esempio: per l’iterazione 2, se AB e AC sono large, (e hanno pari prefisso, A) si genera il candidato ABC.

Apriori (II)

Condizione necessaria perche’ un itemset candidato a cardinalita’ k (k-itemset) sia a supporto sufficiente: tutti e k i suoi sottoinsiemi a cardinalita’ (k-1) devono essere a supporto sufficiente.



Apriori (III)

Un candidato determinato tramite il join (ABC) puó essere eliminato in anticipo, prima della lettura sul database, usando il principio di Apriori.

Se in $L_{(k-1)}$ figurano AB e AC ma non BC ABC puo' essere sicuramente eliminato da C_3 risparmiando CPU-time durante una iterazione sul database e forse anche eventuali nuove iterazioni sul database.

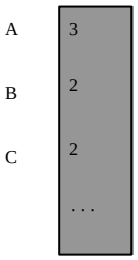
Apriori (IV)

H.Mannila, H.Toivonen, A.I.Verkamo, *Efficient Algorithms for Discovering Association Rules,* AAAI Workshop on Knowledge Discovery in Databases, 1994.

R.Agrawal, R.Srikant, *Fast Algorithms for Mining Association Rules, VLDB 1994.*

Direct Hashing Pruning

Utilizza una hash table per ridurre l'esplosione della cardinalitá nella generazione di C_k a partire da $L_{(k-1)}$* $L_{(k-1)}$ (specialmente nei primi passi)



J.Soo Park, M-S. Chen, P.S.Yu,
An Effective Hash-Based Algorithm for Mining Association Rules,
SIGMOD 1995.

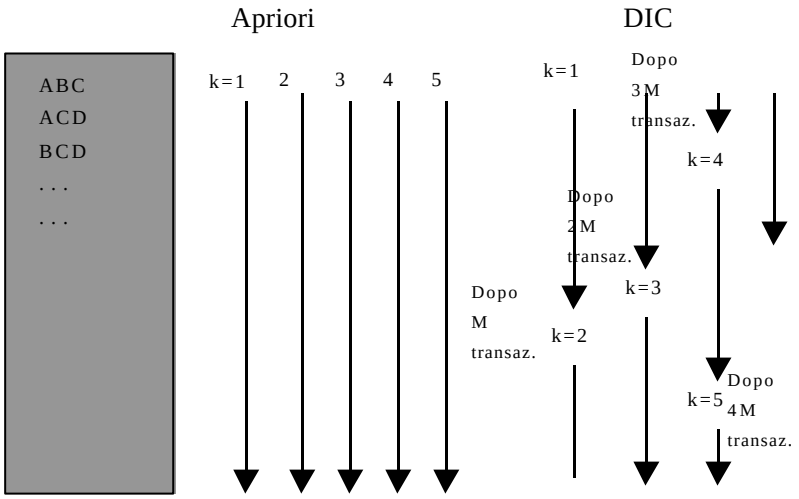
Direct Hashing Pruning (II)

Propone la seguente tecnica per ridurre le dimensioni del database per le scansioni future:

un item i che non é presente in almeno k-1 elementi di $L_{(k-1)}$ viene eliminato da tutte le transazioni che lo contengono (trimming);

ció permette di eliminare anche molte transazioni in toto (pruning).

Dynamic Itemset Counting



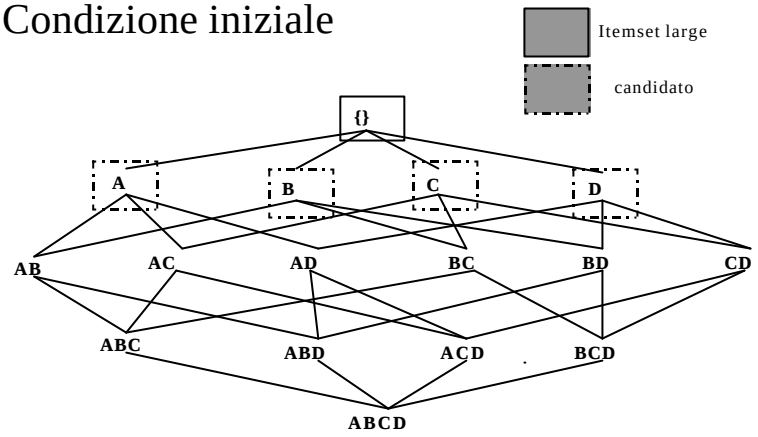
Dynamic Itemset Counting (II)

Aggiunge all'insieme corrente dei candidati nuovi itemset a cardinalitá superiore, basandosi su Apriori e sugli itemset che al momento corrente sono a supporto sufficiente.

Ció avviene in maniera dinamica, ossia successivamente ad un checkpoint, che occorre dopo aver letto un blocco di M transazioni.

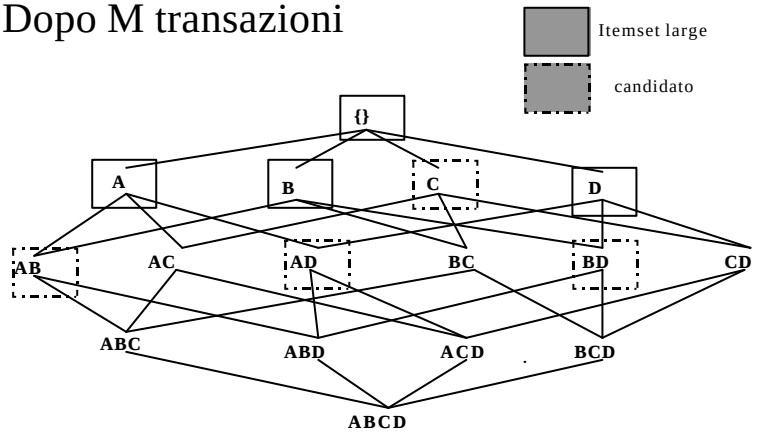
Aggiornamento dell'insieme dei candidati

Condizione iniziale



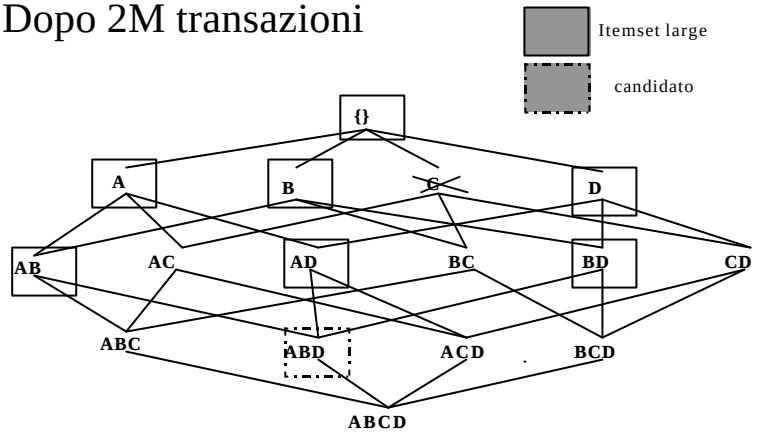
Aggiornamento dell'insieme dei candidati

Dopo M transazioni

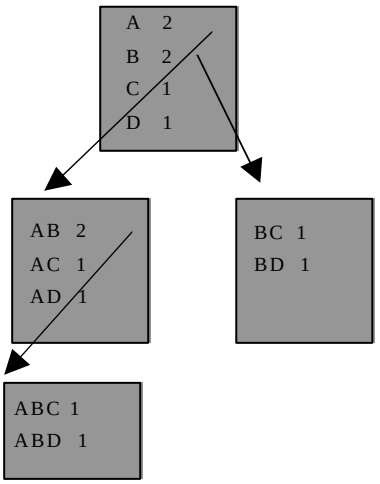


Aggiornamento dell'insieme dei candidati

Dopo 2M transazioni



Dynamic Itemset Counting (III)



Prevede il riordinamento degli item per ridurre la mole di lavoro nell'aggiornamento dei contatori.

Si mettono primi nell'ordine gli item meno frequenti.

Dynamic Itemset Counting (IV)

S.Brin, R.Motwani, J.Ullman, S.Tsur, *Dynamic Itemset Counting and Implication Rules for Market Basket Data*, SIGMOD 1997.

Tecnologie adottate

Sequenziale

Con partizionamento della base dati

Con campionamento

Parallela

Incrementale

Partition

Suddivide il database in partizioni disgiunte in modo che i contatori di tutti i potenziali itemset contenuti in una partizione stiano in memoria centrale.

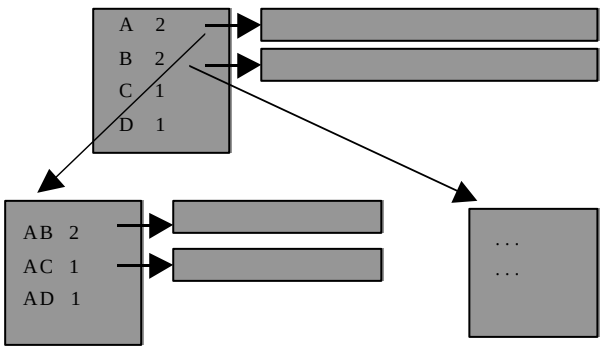
Esegue l'analisi con Apriori separatamente sulle partizioni j e scrive su disco gli insiemi L_{kj} ottenuti per ciascuna di esse.

Fa il merge degli insiemi L_{kj} e fa una seconda passata sul database.

Partition (II)

Fa due sole passate sulla base dati.

Mantiene liste di transazioni per ciascun itemset.



Partition (III)

A.Savasere, E.Omiecinski, S.Navathe, *An Efficient Algorithm for Mining Association Rules in Large Databases*, VLDB 1995.

Tecnologie adottate

Sequenziale

Con partizionamento della base dati

Con campionamento

Parallela

Incrementale

Algoritmi con campionamento

H.Toivonen, *Sampling Large Databases for Association Rules*, VLDB 1996.

Effettua con Apriori un'analisi preliminare dei candidati su un campione del database in memoria e con un supporto minimo meno stringente di quello impostato dall'utente.

Termina la lettura del database calcolando il supporto reale dei candidati cosí individuati.

Campionamento (II)

In una collezione di itemset S la probabilitá che vi sia un itemset il cui supporto è stato stimato con un errore $>e$ sulla base del campione s è al massimo $|S|2^{-(2|s|e^2+1)}$.

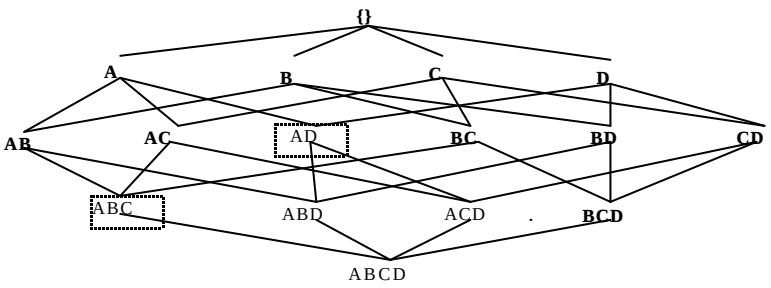
e	P	| *s* |
0.01	0.01	27 000
0.01	0.001	38 000
0.001	0.01	2 700 000
0.001	0.001	3 800 000

Campionamento (III)

Dato un large itemset X, un campione *s* e un limite superiore *d* alla probabilitá che il supporto di X sia stato stimato con errore >*e* la soglia minima di supporto *low_freq* abbassata rispetto a *min_freq* si determina con *low_freq*<*min_freq*- sqrt[1/(2|*s*|)ln(1/d)].

Campionamento (IV)

Data la collezione *S* degli itemset a supporto sufficiente il Negative Border di *S* è l'insieme degli itemset che *non* sono a supporto sufficiente ma tali che *tutti* i loro sottoinsiemi lo sono.



Campionamento (V)

L’algoritmo determina sulla base del campione l’insieme dei candidati *S* e il suo *Negative Border* e di questi determina il supporto reale con la passata sul database.

Se esiste un elemento del *Negative Border* che ha supporto sufficiente, ció indica che potrebbe esserci stato un *miss*.

Si ridetermina il Negative Border includendo il miss nell’insieme S e si ripete una seconda passata sul database.

Tecnologie adottate

Sequenziale

Con partizionamento della base dati

Con campionamento

Parallela

Incrementale

Algoritmi paralleli

R.Agrawal, J.C.Shafer, Parallel Mining of Association Rules, TKDE 1996.

Ipotesi:

- L'architettura ipotizzata é di tipo shared-nothing: ciascun processore ha a disposizione memoria e disco privato.
- I processori comunicano attraverso una rete per mezzo di scambio di messaggi.
- Il database viene partizionato in modo equo rispetto ai processori.

Algoritmi paralleli

Si propongono tre algoritmi (basati su Apriori) che differiscono rispetto a ció che i processori si scambiano e al loro grado di interdipendenza.

- I contatori degli itemset (*Count)* - comunicazione sincrona.
- I candidati e i dati del database (*Data*) - comunicazione sincrona.
- Gli itemset a supporto sufficiente (*Candidate*) - comunicazione asincrona.

Count

Ciascun processore possiede gli stessi candidati C_k al passo k, ma ciascuno determina i propri conteggi sulla base della propria partizione.

Alla fine del passo k i processori si scambiano i valori dei contatori per determinare L_k.

Al passo 1 i processori determinano anche i candidati che devono poi essere scambiati (non è detto che siano gli stessi per tutti).

Count (II)

Vantaggi
I processori realizzano la passata sulla propria partizione in modo indipendente e asincrono.

Svantaggi
Il numero totale di candidati che *Count* puó determinare rimane fissato dalla memoria di ogni singolo processore: crescendo il numero di processori, non cambia.

Data

Al passo k tutti i processori posseggono C_k ma ogni processore deve considerare 1/n del totale e lo determina in modalitá round-robin, a seconda del proprio id.

Ogni processore si scambia i dati delle partizioni (in modo asincrono) per trovare i contatori globali dei candidati ad essi assegnati.

Alla fine del passo k i processori si devono sincronizzare per scambiarsi i candidati in modo che tutti possano determinare L_k.

Data (II)

Vantaggi

Il numero di candidati aumenta col numero di processori ed è determinato dalla memoria totale del sistema.

Svantaggi

Ad ogni passo ogni singolo processore deve scambiare con tutti gli altri i propri dati: è fattibile solo in architetture dalla comunicazione veloce.

Candidate

Partiziona sia l'insieme dei candidati che i dati sfruttando la conoscenza del dominio. In tal modo ogni processore è indipendente dagli altri perché i candidati da considerare sono pertinenti i dati a ciascuno assegnati. (Alcune porzioni del database possono essere replicate in seguito al partizionamento dei candidati.)

Candidate (II)

Fino al passo *l* usa *Count* o *Data*.

Al passo *l* suddivide C_l ai processori in base al prefisso dei candidati e si segna il processore a cui ciascun candidato è stato assegnato.

Ogni processore fa una passata su tutto il database (occorre lo scambio di dati) e si tiene le transazioni a lui pertinenti. Trova il supporto globale dei candidati a lui assegnati.

Comunica in modo asincrono gli itemset large in modo che tutti possano fare pruning dei candidati (con Apriori) al passo successivo.

Candidate (III)

Lo scambio di dati tra processori al passo *l* avviene in modo “mirato” perché ogni processore conosce il partizionamento dei candidati.

Per i passi k successivi, ogni processore fa una passata sulla propria partizione del database e calcola il supporto globale dei propri candidati.

Ogni processore *j* colleziona L_{ki} degli altri processori *i*. Il problema è che i processori hanno velocitá diverse: si distingue se un itemset non è stato ricevuto dal processore *j* perché *j* è piú lento oppure no.

Candidate (IV)

Vantaggi
Rende indipendenti i processori sfruttando la conoscenza del dominio. Elimina la comunicazione sincrona tra processori.

Svantaggi
E’ difficile bilanciare il carico di lavoro tra processori. Il passo di redistribuzione dei dati si è dimostrato troppo costoso. Non sfrutta la memoria globale sufficientemente.

Tecnologie adottate

Sequenziale

Con partizionamento della base dati

Con campionamento

Parallela

Incrementale

Algoritmi incrementali

D.W.Cheung, J.Han, V.T.Ng, C.Y.Wong, *Maintenance of Discovered Association Rules in Large Databases: An Incremental Updating Technique,* ICDE 1996.

Propongono un algoritmo per l'aggiornamento delle regole di associazione (mantenenedo inalterato il supporto minimo) dopo un consistente aggiornamento del database.

Fast UPdate

L’aggiornamento del database potrebbe:

1. aver modificato il valore del supporto e confidenza delle regole di associazione;
2. aver invalidato alcune regole di associazione;
3. aver introdotto nuove regole di associazione.

FUP (II)

L’algoritmo fa molteplici iterazioni sul database con DHP. Ciascuna di esse è divisa in due fasi.

Nella prima si scandisce solo il Delta di incremento del database per individuare il loro supporto rispetto al Delta. Lo scopo è di individuare l’insieme di candidati il cui supporto deve essere confermato nella seconda fase, con una passata sul database originale.

FUP (III)

Per ciascun itemset che era large rispetto al database originale, si controlla il supporto (countDB+countDelta)/(DB+Delta)

- Se <minsup, l'itemset viene eliminato.
- Se >minsup il nuovo supporto dell'itemset è determinato.

FUP (IV)

L'insieme dei candidati è determinato con Apriori rispetto all'insieme degli itemset che risultano a supporto sufficiente rispetto al database *aggiornato*.

Tuttavia, solo quelli che *non* erano a supporto sufficiente sul database originale vengono poi effettivamente controllati nella successiva passata sul database originale.

Nuovi algoritmi

D-I. Lin, Z.M.Kedem, *Pincer-Search: A New Algorithm for Discovering the Maximum Frequent Set*, EDBT 1998.

R.J.Bayardo Jr., *Efficiently Mining Long Patterns from Databases*, SIGMOD 1998.

Pincer-Search

Il numero di candidati determina sia il tempo di I/O (ripetute letture del database) sia di CPU (aggiornamento del supporto e creazione del nuovo insieme di candidati).

Si cerca di ridurre il numero di candidati mantenuti esplicitamente rappresentandoli implicitamente nel Maximum Frequent Candidate Set (MFCS).

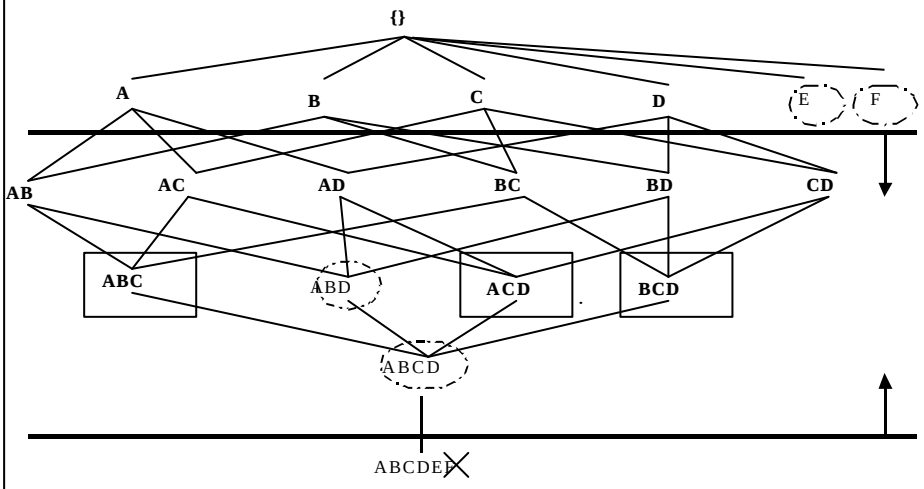
Pincer-Search (II)

Usa entrambi i principi seguenti:

Tutti i subset di un itemset a supporto sufficiente sono itemset a supporto sufficiente.

Tutti i superset di un itemset che non è a supporto sufficiente non sono a supporto sufficiente.

Pincer-Search (III)



Pincer-Search (IV)

Fa restore degli itemset mantenuti implicitamente in MFCS per generare la frontiera dei candidati a cardinalitá inferiore.

Il pruning di un candidato (prima della lettura del database) avviene su MFCS, controllando che il candidato sia un sottoinsieme di un elemento in MFCS.

Pincer-Search (V)

Fa esplicito riferimento all'algoritmo dello spazio delle versioni di Mitchell, facendo le seguenti analogie:

- un itemset che si rivela frequente è un esempio positivo;
- un itemset che si rivela infrequente è un esempio negativo;
- la frontiera dei candidati a cardinalitá inferiore è la linea S;
- la frontiera dei candidati a cardinalitá massimale (MFCS) è la linea G.

Pincer-Search (VI)

Ci sono alcune importanti differenze però:

Pincer-Search riceve osservazioni provenienti sia dall'alto del lattice (osservazioni generali) che dal basso (osservazioni specializzate);

Version-Space invece riceve osservazioni solo dal basso (osservazioni specializzate) e induce dall'alto (sulle generalizzazioni).

Pincer-Search (VII)

Pincer-Search

Per ogni esempio positivo:

1. Combina l'esempio con un altro, specializzandolo: ({AB}+{AC}={ABC}).
2. Se non vale una sua generalizzazione ({BC}), rimuovi la specializzazione fatta (Apriori).

Per ogni esempio negativo:

1. Rimuovi dall'insieme dei candidati tutti quelli che lo coprono.
2. Rimuovi da MFCS tutte le descrizioni che lo coprono con tante nuove descrizioni ottenute eliminando uno alla volta gli attributi.

Version Space

Per ogni esempio positivo:

1. Combina l'esempio con un altro, generalizzando S: (A,B,x,x)+(A,x,x,D)=(A,x,x,x).
2. Rimuovi da G le descrizioni che non coprono l'esempio dato.

Per ogni esempio negativo:

1. Rimuovi da S le descrizioni che lo coprono.
2. Rimuovi le descrizioni in G che lo coprono con tante nuove descrizioni ottenute specializzando uno alla volta gli attributi: (x,B,x,x)=>(A,B,x,x),…,(x,B,x,D).

Pincer-Search (VIII)

Pincer-Search è particolarmente interessante per le seguenti caratteristiche, rispetto alle quali gli algoritmi tradizionali perdono in prestazione.

- Trae beneficio dal diminuire del supporto minimo.
- Trae beneficio dall'aumentare della lunghezza massima dei large itemset.

Seq

R.Meo, *A New Approach for the Discovery of Frequent Itemsets,* DAWAK 1999.

Seq ha caratteristiche simili all'algorimo di Bayardo e trae beneficio da database molto grossi, con una statistica pronunciata, ossia quando gli itemset large si trovano *clusterizzati* e non sono invece "sparsi".

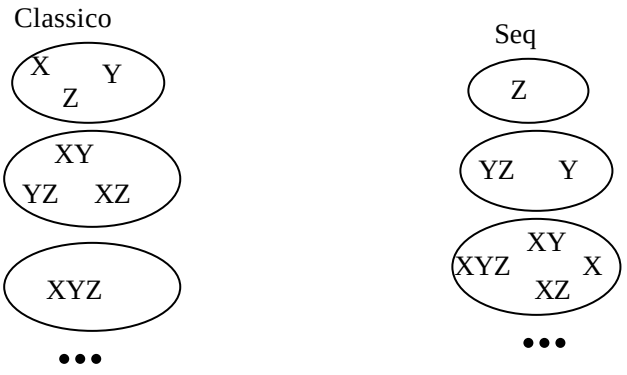
Seq

Esistono diversi approcci per decidere quali itemset considerare in ciascuna iterazione:

- solo gli itemset di data cardinalità;
- solo gli itemset di una data porzione del database;
- solo gli itemset presi da un campione del database.

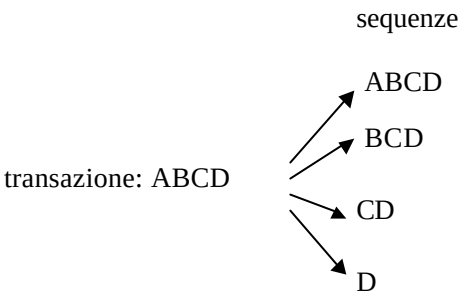
Seq

In ciascuna iterazione sono considerati solo gli itemset che iniziano con un dato item.



L’algoritmo *Seq*

E’ basato sulla creazione di sequenze.



I passi di *Seq*

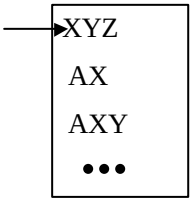
Seq lavora in due passi:

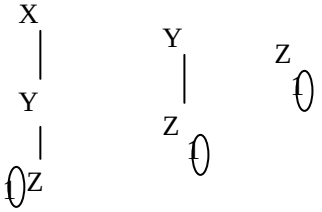
Step 1: Generazione delle sequenze

Input: Il database.

Si trovano le sequenze per ciascuna transazione e si memorizzano in memoria principale.

Output: gli alberi delle sequenze con il loro supporto.





I passi di *Seq*

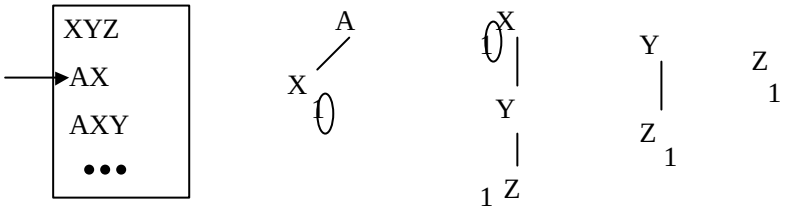
Seq lavora in due passi:

Step 1: Generazione delle sequenze

Input: Il database.

Si trovano le sequenze per ciascuna transazione e si memorizzano in memoria principale.

Output: gli alberi delle sequenze con il loro supporto.



I passi di *Seq*

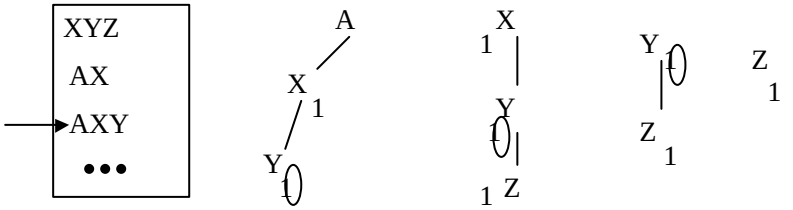
Seq lavora in due passi:

Step 1: Generazione delle sequenze

Input: Il database.

Si trovano le sequenze per ciascuna transazione e si memorizzano in memoria principale.

Output: gli alberi delle sequenze con il loro supporto.



I passi di *Seq* (II)

Step 2: Generazione degli itemset dalle sequenze

Input: le sequenze e il loro supporto.

Si legge un albero alla volta ordinando gli alberi in maniera decrescente per l'item nella radice.

Gli itemset sono generati come sottoinsiemi delle sequenze.

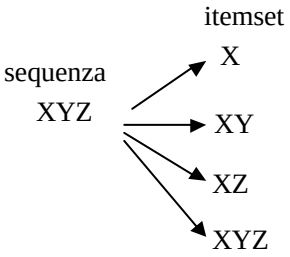
Quando la lettura di un albero è completata, si esegue una fase di *pruning* sul supporto degli itemset.

Risultato: gli itemset e il loro supporto.

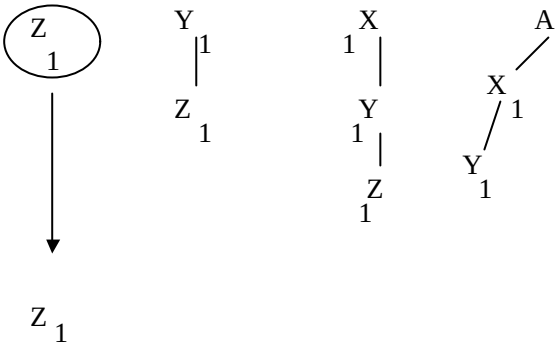
La generazione degli itemset

Da ciascuna sequenza, si generano solo gli itemset *che iniziano con lo stesso item della sequenza*.

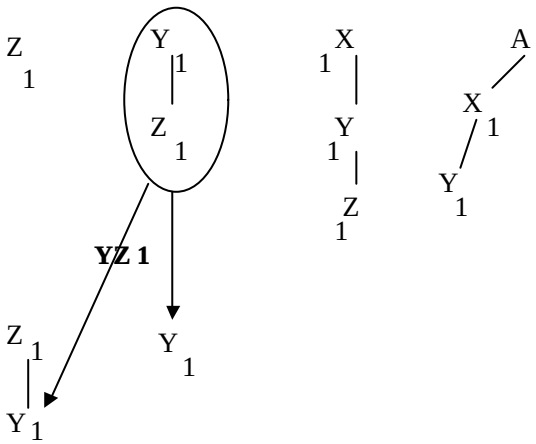
Il loro supporto è aumentato del supporto della sequenza.



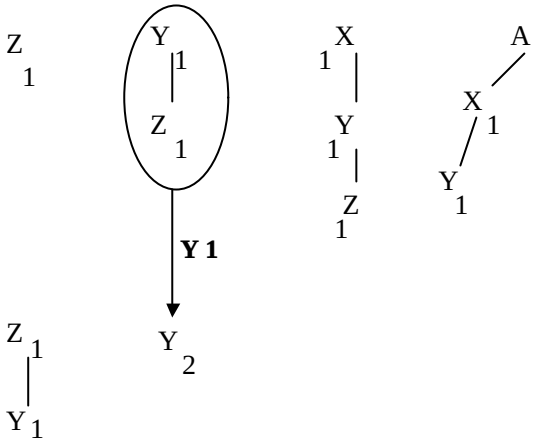
La generazione degli itemset (II)



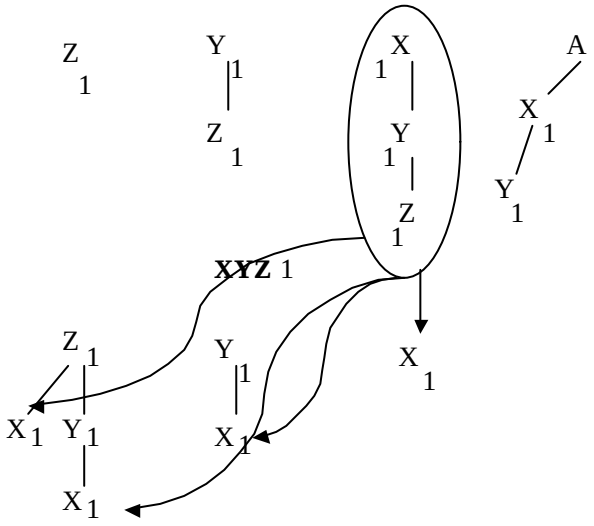
La generazione degli itemset (II)



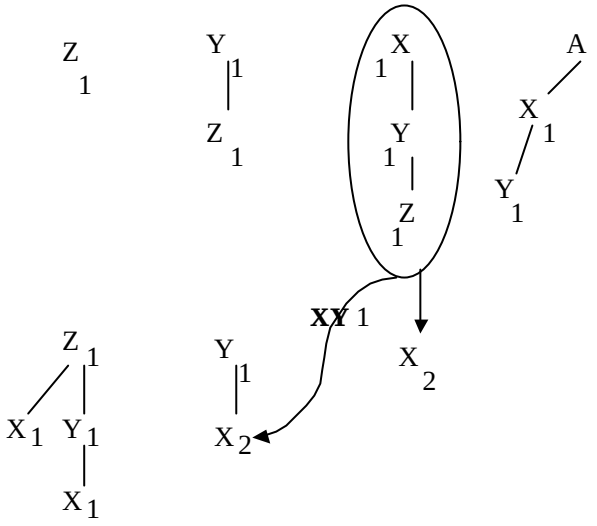
La generazione degli itemset (II)



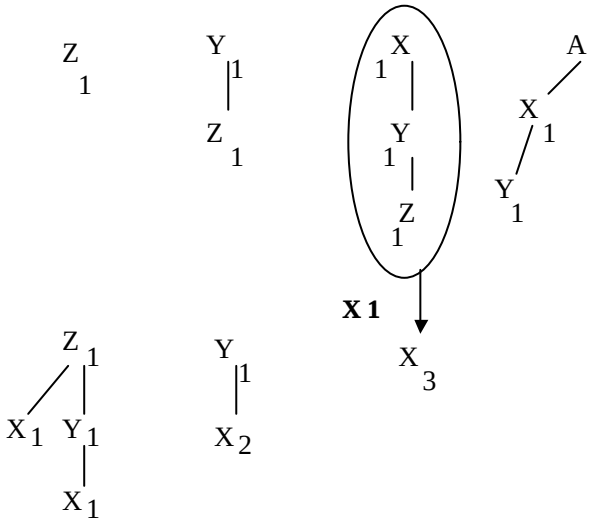
La generazione degli itemset (II)



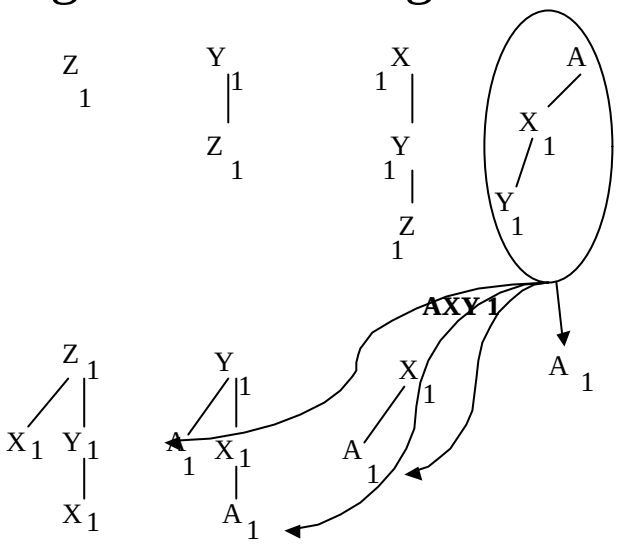
La generazione degli itemset (II)



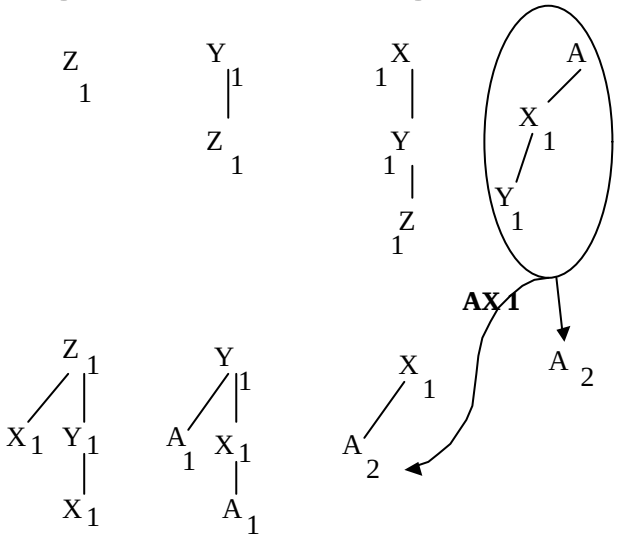
La generazione degli itemset (II)



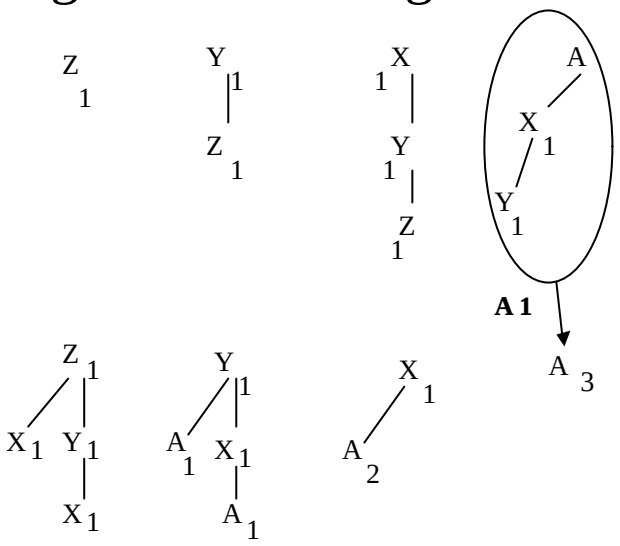
La generazione degli itemset (II)



La generazione degli itemset (II)



La generazione degli itemset (II)



Vantaggi delle sequenze

Le sequenze mantengono una sorta di “riassunto” degli itemset: ciò permette la riduzione del numero di contatori in memoria durante la lettura del database.

La fase di pruning viene eseguita con più frequenza rispetto agli altri algoritmi (una volta per ciascun albero invece che una volta per ciascun livello di profondità degli alberi).

Si risparmia tempo di CPU (la creazione di un itemset composto da k item richiede solo un accesso alla memoria principale invece di k).

Implementazione di *Seq*

Ci si occupa della gestione della memoria durante il passo 1:
quando occorre, uno o più alberi delle sequenze vengono trasferiti in file (separati); i file hanno un formato compresso.

I file vengono letti al passo 2.

Per database molto grandi la dimensione dei file è una frazione della dimensione del database.

Valutazione di *Seq*

Il database viene letto solo una volta.

Esiste un limite superiore al volume totale di dati scambiati con la memoria di massa:
dimensione del database + 2 * dimensione dei file.

Il tempo di esecuzione è una funzione lineare del numero totale di transazioni del database.

Il tempo di esecuzione è quasi costante rispetto a:

- il valore del supporto minimo;
- la lunghezza media degli itemset.