Approfondimento di Data Mining

Autore: Lionello Marco matricola: 810079

15 Aprile 2006

Indice

Elenco delle figure

Capitolo 1

Introduzione

SPADE acronimo di Sequential PAttern Discovery using Equivalence classes è un'algoritmo per la scoperta veloce di *Pattern Sequenziali* (Sequenze Frequenti). Le principali caratteristiche che fanno di SPADE un'algoritmo efficente sono:

- la suddivisione in sottoproblemi del problema originale, rendendo i sottoproblemi gestibili direttamente in memoria centrale.
- utilizzo di efficenti tecniche di ricerca grazie alla teoria *lattice*.
- tutte le possibili sequenze sono scoperte al massimo con tre scansioni del database.
- linearmente scalabile sui customer (clienti) e sui parametri del database.

La scoperta delle sequenze frequenti è un'operazione molto complessa. Si pensi che in un database composto da m attributi è possibile ricavare $\Theta(m^k)$ possibili sequenze frequenti, dove k è la lunghezza della sequenza. Il problema delle sequenze frequenti è trovare in un database delle regole che descrivono eventi in una determinata sequenza temporale. Un esempio molto semplice è quello del negozio di libri. Vogliamo ad esempio estrarre dal database dei libri venduti la regola che dice: il settanta per cento delle persone che compra il libro 'Codice da Vinci' compra anche 'Angeli e Demoni'. Si noti che l'ordine temporale è fondamentale infatti

non è detto che il settanta per cento delle persone che compra 'Angeli e Demoni' compra 'Codice da Vinci'. Prima della pubblicazione dell'articolo di Zaki[1], inventore di SPADE, da cui questo approffondimento trae il maggior spunto, le limitazioni degli algoritmi per il riconoscimento delle Sequenze Frequenti erano molte. Le due limitazioni erano essenzialmente: l'iteratività degli algoritmi che causavano molte scansioni del database e le strutture dati complicate che comportavano un overhead spaziale e computazionale.

Il goalr di SPADE è nell'evitare le limitazioni. Le sue caratteristiche principali sono:

1. usare un *Vertical Id Database*, nel quale si associa ad ogni sequenza una lista di oggetti che occorrono nel database con una marca temporale.

2. usare la teoria *Lattice* per decomporre il problema in sottoproblemi indipendenti e gestibili direttamente in memoria centrale. Grazie a questa teoria riusciamo a individuare le sequenze frequenti in:

   - 3 scansioni del database.
   - 1 scansione del database con alcune informazioni preprocessate.

3. usare il *Disaccoppiamento* del problema della decomosizione da quello della scoperta delle sequenze frequenti.

Capitolo 2

Mining delle sequenze

In questa fase si introducono le nozioni fondamentali per affrontare il problema. Si considerino le seguenti definizioni:

- **Item**: è un'insime di m attributi distinti e lo indichiamo con $I = \{i_1, ..., i_m\}$
- **ItemSet**: è una collezione disordinata e non vuota di item.
- **Sequenza**: è una lista di item ordinata.
- **K Sequenza**: è una sequenza di k item.
- **Sotto Sequenza**: una sequenza $\alpha(\alpha_1 \mapsto \alpha_2 \mapsto ..... \mapsto \alpha_n)$ è una sottosequenza di $\beta(\beta_1 \mapsto \beta_2 \mapsto ..... \mapsto \beta_n)$ e si denota con $\alpha \leq \beta$ se esiste un intero $i_1 \leq i_2 \leq ..... \leq i_n$ per il quale $a_j \subseteq b_{ij}$ per ogni a_j.
- **Sotto Sequenza Propria**: si denota con $\alpha < \beta$ se $\alpha \leq \beta$ ma non $\beta \leq \alpha$.
- **Sequenza massima**: se non è sottosequenza di altre sequenze.
- **Transazione**: record contenente un insieme di item e un unico identificatore.
- **Cliente**: record costituito da un'identificatore unico e ha associato una lista di transazioni $\{t_1, ..., t_n\}$

DATABASE

Customer-Id	Transaction-Time	Items
1	10	C D
1	15	A B C
1	20	A B F
1	25	A C D F
2	15	A B F
2	20	E
3	10	A B F
4	10	D G H
4	20	B F
4	25	A G H

FREQUENT SEQUENCES

Frequent 1-Sequences

A	4
B	4
D	2
F	4

Frequent 2-Sequences

AB	3
AF	3
B->A	2
BF	4
D->A	2
D->B	2
D->F	2
F->A	2

Frequent 3-Sequences

ABF	3
BF->A	2
D->BF	2
D->B->A	2
D->F->A	2

Frequent 4-Sequences

D->BF->A	2

Figura 2.1: Database originale

Prima di cominciare con la trattazione vera e propria è utile fare alcune assunzioni. La prima assunzione è che nessun customer ha più di una transazione con lo stessa marca temporale (time stamp) quindi si può porre l'uguaglianza tra il tempo della transazione e l'identificatore della transazione. La seconda assunzione è che la lista delle *transazioni - clienti* deve essere ordinata sulla base del tempo della transazione. Si noti che la lista delle *transazioni - clienti* è una sequenza di transazioni chiamata *sequenza - clienti* (customer sequence) $t_1 \mapsto t_2 \mapsto ..... \mapsto t_n$. Il database è quindi un contenitore di *sequenze - clienti*. Possiamo indicare con $\alpha \leq C$ il fatto che una sequenza di customer C **contiene** una sequenza α. Il **supporto** $\sigma(\alpha)$ è il numero totale di clienti che contiene la sequenza. Quindi una formula è **frequente** se occorre più volte di una certa soglia che chiameremo min-sup. Quindi dato un database D il problema di trovare le 'sequenze con significato' è di trovare le sequenze che sono presenti almeno min-sup volte.

2.1 Problemi associati

Il problema del 'mining sequential pattern' è stato introdotto in Agraw-al[3] nel quale veniva presentato anche 'l'algoritmo a priori'. Successivamente viene creato l'algoritmo GSP che è risultato 20 volte più performante.

Un eppisodio consiste in un insieme di eventi e una associazione parziale ordinata sull'evento. La nostra definizione di sequenza può essere definita come eppisodio. Il problema degli eppisodi frequenti è trovare gli eppisodi frequenti in un unica lunga sequenza di eventi, mentre il nostro problema è quello di trovare le sequenze frequenti in molte sequenze di molti clienti. L'algoritmo MEDD e MSDD scopre pattern in sequenze di eventi multiple. Purtroppo trova al massimo sequenze di lunghezza due.

La scoperta di sequenze quindi altro non è che un problema di scoperta delle associazioni (association discovery) su un database temporale. Possiamo affermare che mentre la association discovery scopre solo l'intra - transaction pattern e quindi gli itemset, la sequential discovery scopre l'inter - transaction pattern e quindi le sequenze. Insomma si può affer-

mare che l'insieme delle sequenze frequenti è un super insieme dell'insieme dei pattern frequenti.

Capitolo 3

Approccio basato sulla teoria lattice

Sia P un insieme (itemset). Un *ordine parziale* L in P è una relazione lineare $\leq$ che è:

- riflessiva: $X \leq X$
- antisimmetrica: da $X \leq Y$ e $Y \leq X$ si ha $X \equiv Y$
- transitiva: da $X \leq Y$ e da $Y \leq Z$ si ha $X \leq Z$

Un ordine parziale si dice *Lattice* se le due operazioni lineari:

- join: $X \vee Y$
- meet: $X \wedge Y$

esistono per ogni X,Y appartenenti a L.

Si dice inoltre Lattice completo se ogni join e meet esitono su ogni sottoinsieme arbitrario di L. Un noto teorema della teoria del lattice afferma inoltre che tutti i lattice finiti sono anche completi. Quindi questo implica che se M è un sotto-lattice di L, allora se X,Y appartengono a M allora le operazioni di join e di meet appartengono a M ossia: $X \vee Y$ e $X \wedge Y$ appartengono a M.

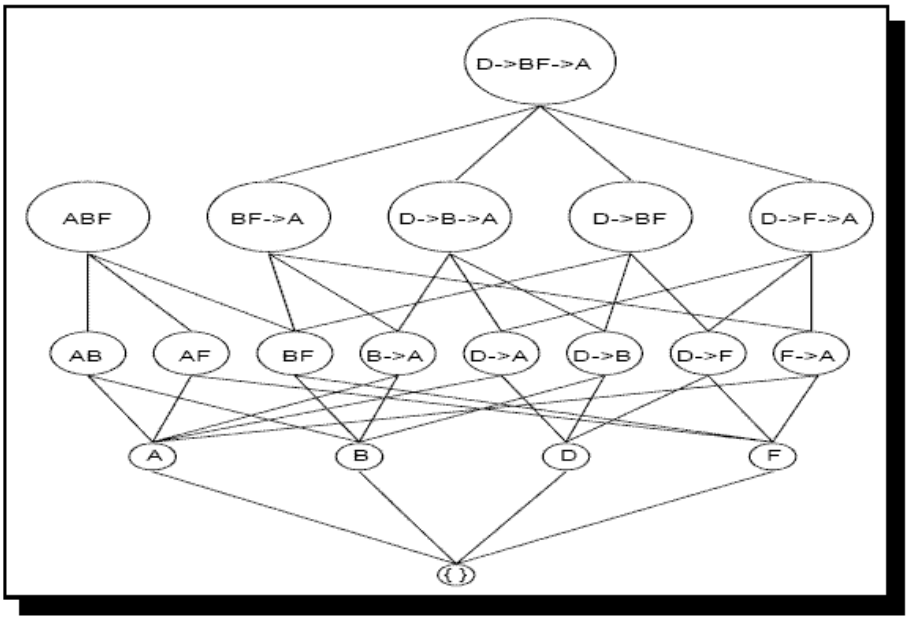

Figura 3.1: Lattice indotto dalla sequenza massimale $D \mapsto BF \mapsto A$

3.1 Teorema 1

Dato un insieme I di item, l'insieme ordinato S di tutte le possibili sequenze di item, è un lattice completo nel quale join e meet sono date dall'unione e intersezione rispettivamente.

$$\vee \{A_i, i \in I\} \equiv \bigcup_i A_i$$

$$\wedge \{A_i, i \in I\} \equiv \bigcap_i A_i$$

Si veda ad esempio la figura 3.1. Alla base dell'albero è presente l'insieme vuoto $\perp \equiv \{\}$. Gli *atomi* sono gli elementi immediatamente sopra il primo livello. Si fa notare inoltre che le sequenze frequenti formano un meet-semilattice ossia se si ipotizza X e Y frequenti allora anche $X \cap Y$ sarà frequente. Questo non vale per il join-semilattice.

3.2 Lemma 1

Tutte le sottosequenze di sequenze frequenti sono frequenti.

3.3 Conteggio del supporto

Definiamo con L(X) la id-list formata da due parametri *cid* (customer id) e *tid* (transaction id). Sia inoltre A(S) l'insieme degli atomi di una sequenza S.

3.3.1 Lemma 2

Ogni sequenza S può essere otenuta come unione dei suoi atomi e il suo supporto è ottenibile come intersezione delle rispettive id-list.

A	
CID	TID
1	15
1	20
1	25
2	15
3	10
4	25

B	
CID	TID
1	15
1	20
2	15
3	10
4	20

D	
CID	TID
1	10
1	25
4	10

F	
CID	TID
1	20
1	25
2	15
3	10
4	20

Figura 3.2: id-list degli atomi

3.3.2 Lemma 3

Se una sequenza viene data come unione di sequenze di atomi e denotiamo con J questo insieme allora il supporto è dato come intersezione delle rispettive id list degli atomi che compongono la sequenza.
Grazie a questo lemma che è conseguenza immediata del lemma precedente si può determinare il supporto di ogni sequenza di lunghezza K semplicemente intersecando l'id-list di alcune delle sue (k-1) sottosequenze. L'algoritmo quindi utilizzerà a pieno questo lemma effettuando alla fine un controllo di cardinalità per stabilire se la sequenza è frequente o meno.

3.3.3 Lemma 4

Se $X \leq Y$ ossia X è sottosequenza di Y allora: $XL(X) \supseteq L(Y)Y$

In pratica questo lemma afferma che se X è sottosequenza di Y allora la cardinalità di Y deve essere minore o uguale a quella di X.

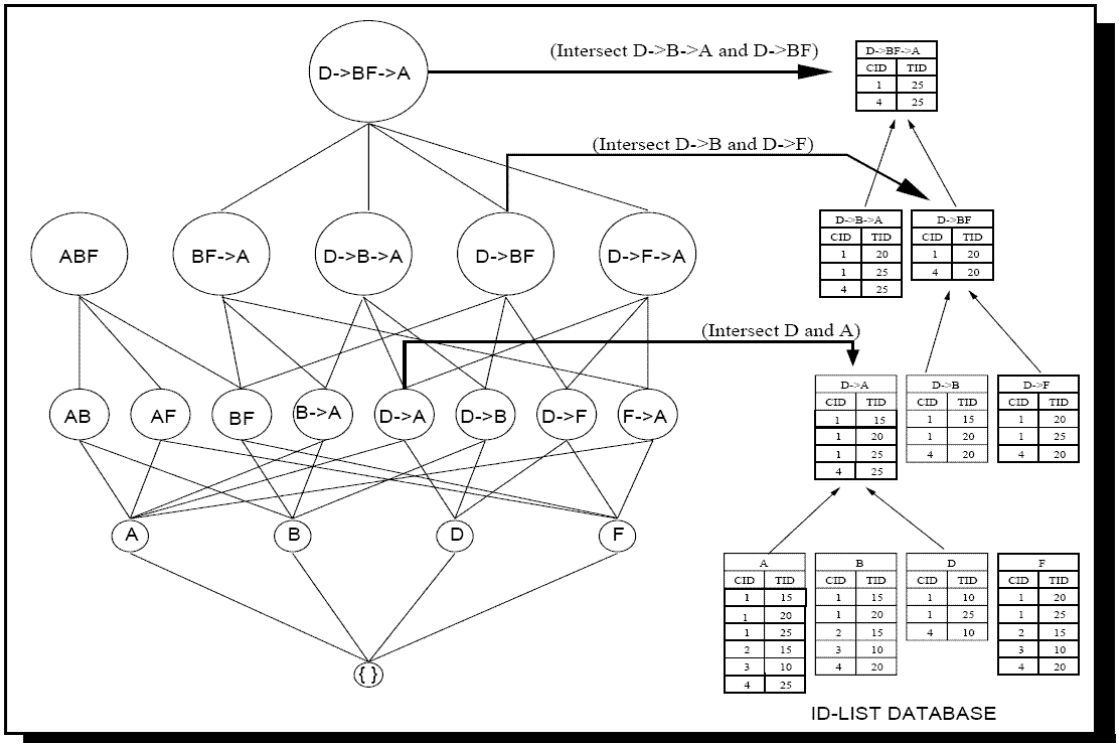

Figura 3.3: computazione del supporto grazie a intersezioni di id-list

3.4 Decomposizione della Griglia. Classi basate su prefissi

In questa fase ci si preoccupa della decomposizione del problema in sottoproblemi al fine di garantire che la parte di database che si deve computare sia presente in memoria principale. Questo abbatte i costi di Input-Output. Una *relazione di equivalenza* è una relazione che è riflessiva, simmetrica e transitiva di tipo binario. Una *classe di equivalenza* è il risultato del partizionamento dell'insieme in sottoinsiemi disgiunti. In pratica le sequenze X e Y sono della stessa classe se e solo se condividono le stessa lunghezza di prefisso K. Questo tipo di equivalenza si chiama 'prefix based equivalence'.

3.4.1 Lemma 5

Ogni classe di equivalenza indotta dalla relazione di equivalenza θ_k è un sub-lattice (sotto griglia) di S.

Grazie al lemma 2 e 3 inoltre si possono generare tutti i supporti dall'intersezione delle id-list degli atomi o come intersezione di due sottosequenze di livello inferiore.

3.5 Ricerche di sequenze frequenti

Per la ricerca di sequenze frequenti possiamo utilizzare due strategie: breadth first e depth first entranbe basate su ricerche ricorsive. La prima è una ricerca in larghezza ed effettua una ricerca di tipo bottom-up, in pratica processa tutti i figli di ogni livello prima di passare al successivo. Il secondo invece risolve le classi di equivalenza lungo un percorso prima di muoversi su di un'alto percorso. I vantaggi di BFS sono essenzialmente la presenza di informazioni per la potatura. I vantaggi di DFS sono l'uso di poca memoria, infatti richiede solo l'id-list del percorso che stà risolvendo. Attualmente in fase di studio ci sono approcci alternativi che eliminano alcuni percorsi conosciendo a priori il fatto che essi non sono frequenti.

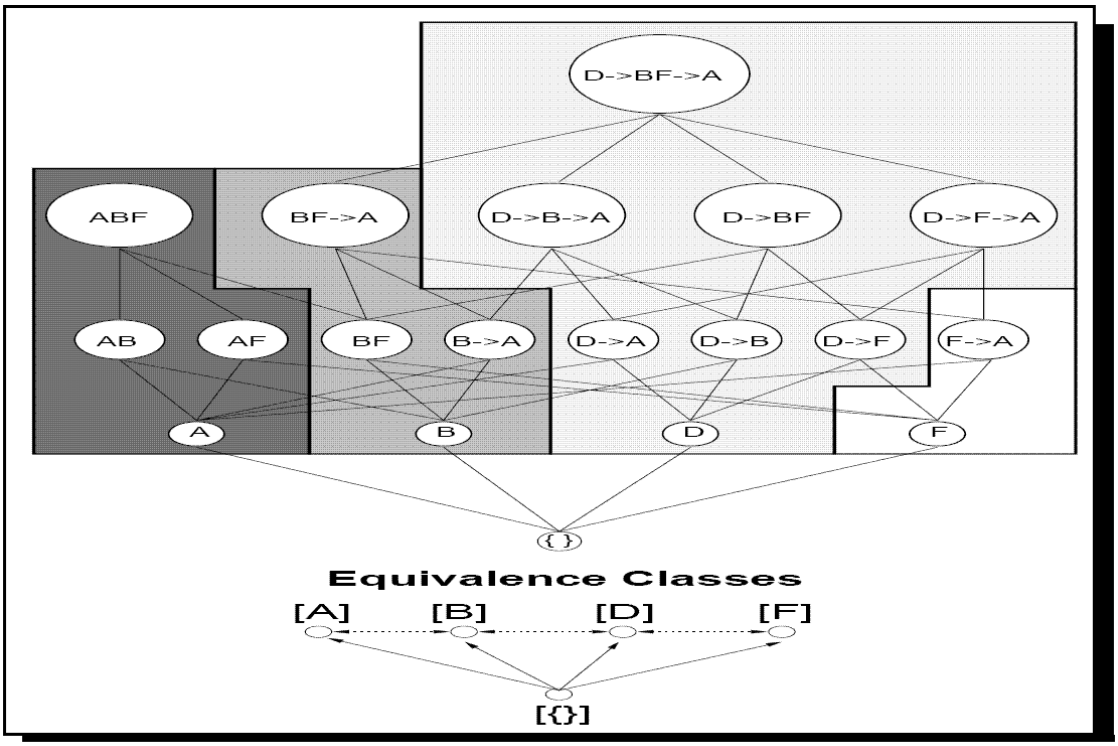

Figura 3.4: Classi di equivalenza indotte

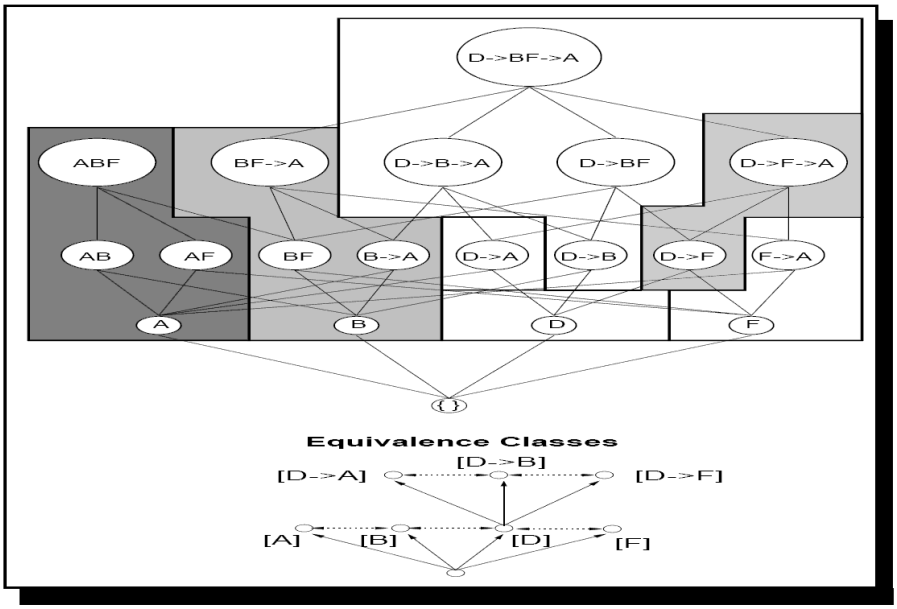

Figura 3.5: Classi di equivalenza indotte

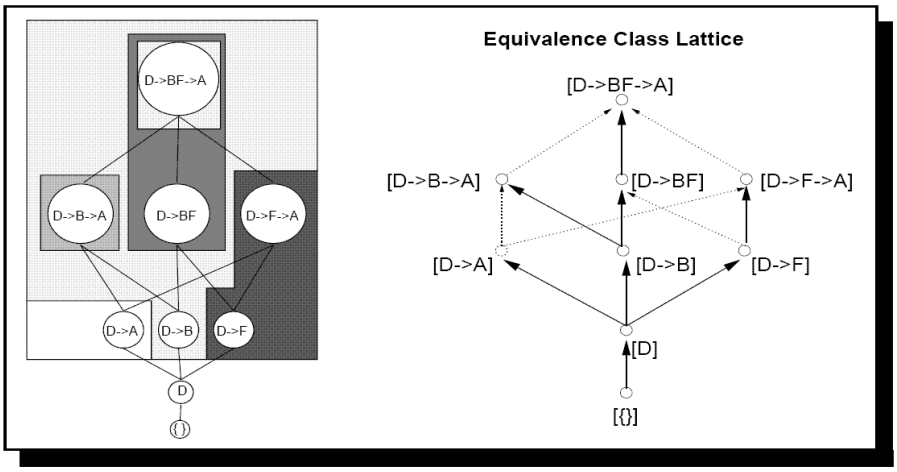

Figura 3.6: Decomposizione ricorsiva della casse D

Capitolo 4

SPADE: design dell'algoritmo e implementazione

4.1 Computazione delle sequenze frequenti di lunghezza 1 e 2

L'uso del database verticale ci permette di controllare il supporto tramite l'intersezione delle id-list.

Per quanto riguarda la computazione del livello 1 essa si esegue facendo una intera scansione del database. Si incrementa il supporto quando si incontra un nuovo *cid*. Questo viene effettuato per ogni item.

Per quanto concerne invece la computazione del livello 2 poniamo $N = I$ ossia N è il numero di item frequenti. Sia inoltre A la media della dimensione in byte delle id-list. Una implementazione stupida è quella di computare tutte le possibili intersezioni di N. I due approcci alternativi sono:

1. Usare alcune informazioni pre processate che diano informazioni sulle 2-sequenze senza appesantire l'algoritmo.

2. Facciamo una trasformazione del database da verticale a orrizzon-

SPADE $(min\_sup, \mathcal{D})$:
$\mathcal{F}_1 = \{$ frequent items or 1-sequences $\}$;
$\mathcal{F}_2 = \{$ frequent 2-sequences $\}$;
$\mathcal{E} = \{$ equivalence classes $[X]_{\theta_1}\}$;
for all $[X] \in \mathcal{E}$ **do** *Enumerate-Frequent-Seq*([X]);

Figura 4.1: L'algoritmo SPADE

cid	$(item, tid)$ pairs
1	(A 15) (A 20) (A 25) (B 15) (B 20) (C 10) (C 15) (C 25) (D 10) (D 25) (F 20) (F 25)
2	(A 15) (B 15) (E 20) (F 15)
3	(A 10) (B 10) (F 10)
4	(A 25) (B 20) (D 10) (F 20) (G 10) (G 25) (H 10) (H 25)

Figura 4.2: Recovery del database da vericale a orizzontale

tale 'al volo'. Questo può essere eseguito con un leggero overhead. A questo punto:

- per ogni item i facciamo uno scan in memoria
- per ogni cliente e transazione (c,t) in L(i) inseriamo (i,t) nella lista del customer c.
- si forma la lista delle sequenze di lunghezza due per ogni cliente c e si incrementa un contatore in un array bidimensionale indicizzato dagli item frequenti.

```
Enumerate-Frequent-Seq(S):
  for all atoms A_i ∈ S do
    T_i = ∅;
    for all atoms A_j ∈ S, with j > i do
      R = A_i ∪ A_j;
      if (Prune(R) == FALSE) then
        L(R) = L(A_i) ∩ L(A_j);
        if σ(R) ≥ min_sup then
          T_i = T_i ∪ {R}; F_|R| = F_|R| ∪ {R};
    end
    if (Depth-First-Search) then Enumerate-Frequent-Seq(T_i);
  end
  if (Breadth-First-Search) then
    for all T_i ≠ ∅ do Enumerate-Frequent-Seq(T_i);
```

Figura 4.3: Pseudo-codice per la procedura Enumerate-Frequent-Seq

4.2 Computazione delle sequenze frequenti di lunghezza $k \geq 3$

Si mostra in figura 4.3 l'algoritmo e si spiegano qui di seguito le varie operazioni. L'input è un sub-lattice S. Le sequenze frequenti vengono generate come intersezione dell'id.list. Prima di effettuare l'intersezione viene effettuato un pruning per evitare che sottosequenze della sottosequenza risultino frequenti, quindi se si effettua l'operazione di pruning di salta l'operazione di intersezione. La fase di lettura del database carica in memoria solo le sequenze frequenti del database. In questo modo si necessita di un solo scan dopo la computazione del livello2.

4.3 Intersezione delle id-list

Si consideri la classe di equivalenza $[B \mapsto A]$ con l'insieme di atomi $\{B \mapsto AB, B \mapsto AD, B \mapsto A \mapsto A, B \mapsto A \mapsto D, B \mapsto A \mapsto F\}$. Se si sostituisce con P il prefisso $A \mapsto B$ si ottengono due categorie:

- $\{P \mapsto A, P \mapsto D, P \mapsto F\}$
- $\{PB, PD\}$

Per ottenere gli atomi usiamo l'intersezione. Ci possono essere quindi tre tipi di intersezione:

1. Itemset atom VS Itemset atom: se stiamo intersecando PB con PD e otteniamo li'itemset PDB
2. Itemset atom VS Sequence Atom: se stiamo intersecando PB con $\{P \mapsto A\}$ e otteniamo $\{PB \mapsto A\}$
3. Sequence atom VS Sequence atom: se stiamo intersecando $\{P \mapsto A\}$ con $\{P \mapsto F\}$. In questo caso ci sono tre possibilità:
   - un nuovo itemset atom $P \mapsto AF$
   - un nuovo sequence atom $P \mapsto A \mapsto F$
   - un nuovo sequence atom $P \mapsto F \mapsto A$

4.4 Sequenze potate

Sia $\alpha 1$ il primo item della sequenza α. Prima di generare l'id-list di una sequenza β di lunghezza k si controlla che tutte le sottosequenze di lunghezza k-1 siano frequenti. Se le k-1 sequenze sono frequenti si crea l'id-list altrimenti non si considera. Quindi nella fase di potatura è necessario avere già processato tutte le sequenze di lunghezza k-1. Risulta per questo motivo la ricerca in larghezza BFS.

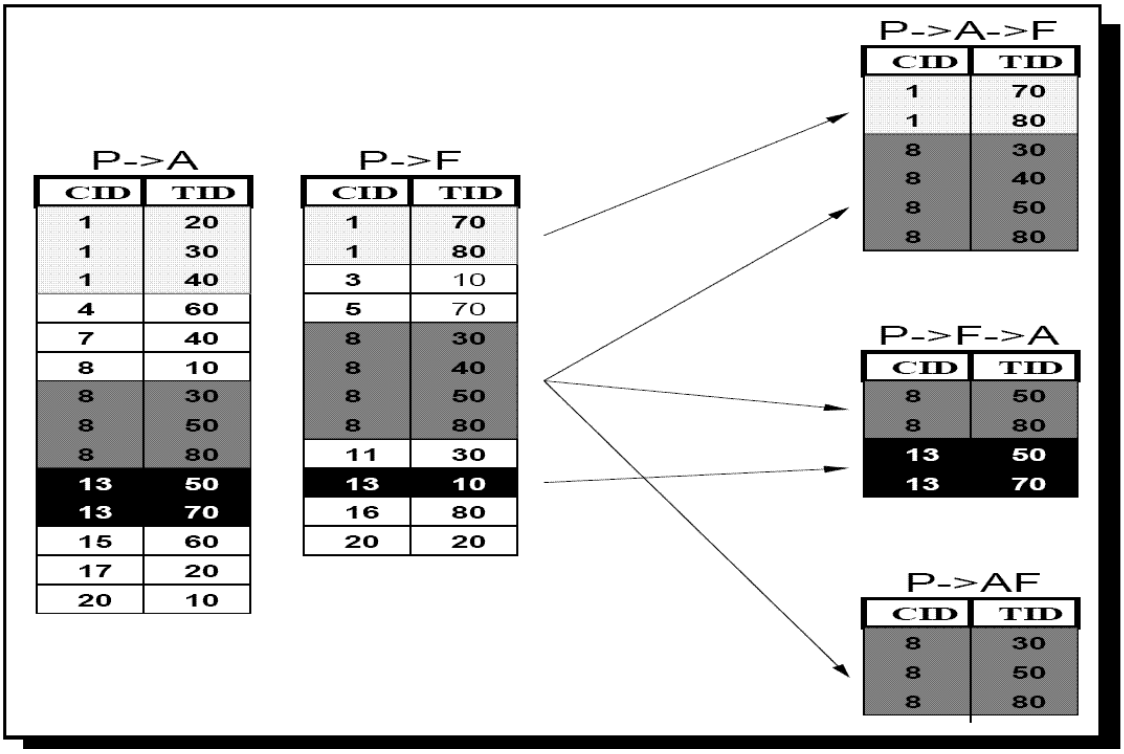

Figura 4.4: Interzezione delle id-list

Prune (β):
  for all $(k-1)$-subsequences, $\alpha \prec \beta$ **do**
    if ($[\alpha_1]$ has been processed, and $\alpha \notin \mathcal{F}_{k-1}$) **then**
      return TRUE;
  return FALSE;

Figura 4.5: Pseudo codice di potatura

Capitolo 5

Risultati Sperimentali

In questa fase si mostreranno i risultati sperimentali dell'esecuzione di SPADE realizzato come BFS e di GSP. La macchina in cui si eseguiranno i test è la stessa e le sue caratteristiche vengono riportate qui di seguito:

- 100 Mhz MIPS
- 256 MB di RAM
- O.S. IRIX 6.2
- 2GB di disco non locale.

5.1 Dataset sintetico

Per eseguire le prove ci si serve di due dataset. Il primo è di tipo *sintetico* ed è stato fornito dall'IBM 'Quest Data Mining Project'. Il dataset simula transazioni reali dove il cliente compra delle sequenze di oggetti. Alcuni clienti possono comprare oggetti da sequenze o possono comprare oggetti da sequenze multiple. La dimensione delle sequenze non sono casuali in modo da garantire che alla fine ci siano poche sequenze frequenti di grande lunghezza. Il dataset viene creato in maniera seguente:

1. N_I Itemset massimale di dimensione media I generato da N item

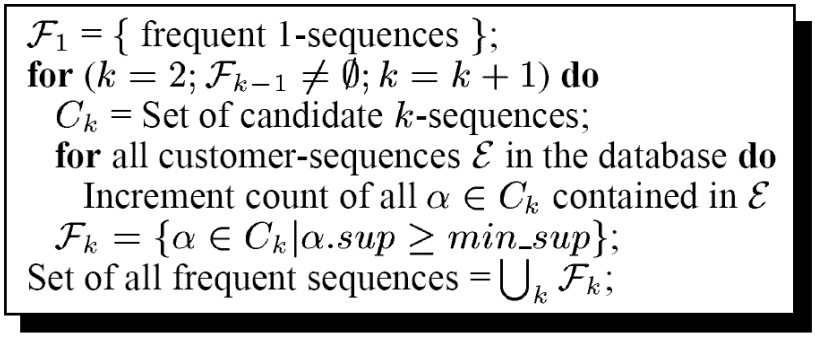

Figura 5.1: L'algoritmo GSP

2. N_S Sequenza massimale di dimensione media S generato assegnando N_I a ogni sequenza

3. si crea un cliente di media C transizioni e le sequenze in N_S vengono assegnate a diversi elementi, rispettando la dimensione media delle transazioni

4. il processo termina quando si arriva a D clienti

In questo test si useranno i valori: NS=5.000 NI=25.000 D=200.000

5.2 Dataset di pianificazione

Il secondo dataset è ottenuto dallo scopo di pianificare dei percorsi. In pratica il pianificatore genera i piani di indirizzamento per andare da una città ad un'altra. In questo caso il cliente è l'identificatore del piano. La transazione è un'evento (un evento è ad esempio 'arrivato con successo','non arrivato','ritardo',ecc. ecc.), un tipo di azione (es. 'muoviti') e dei parametri addizionali (ad esempio: partenza destinazione, mezzo di trasporto ecc. ecc.).

Il mining dei dati in questo caso consiste nel trovare i piani fallimentari. I numeri che contraddistinguono questo dataset sono: 77item , 202071

Dataset	Size (MB)
C10-T2.5-S4-I1.25-D200K	36.8
C10-T2.5-S4-I1.25-D500K	92.0
C10-T2.5-S4-I1.25-D1000K	184.0
C10-T5-S4-I1.25-D200K	56.5
C10-T5-S4-I2.5-D200K	54.3
C20-T2.5-S4-I1.25-D200K	76.7
C20-T2.5-S4-I2.5-D200K	66.5
C20-T2.5-S8-I1.25-D200K	76.4

Table 1: Synthetic Datasets

Figura 5.2: Dataset sintetico

piani , 829236 eventi. La media della lunghezza di un piano è 4,1 mentre la media della lunghezza degli eventi è 7,6.

5.3 Comparazione di SPADE con GSP

I grafici mostrano l'andamento di SPADE e GSP in funzione del supporto. La barra etichettata con SPADE-F2 e GSP-F2 sono i casi in cui il le sequenze di lunghezza due sono già state processate. Il caso SPADE è invece il caso di SPADE con trasformazione al volo del database. Come si può notare il grafico evidenzia il distacco di SPADE su GSP al diminuire del supporto. Si nota infatti che SPADE è circa due volte più veloce per supporti bassi. Anche nel caso degli algoritmi con F2 SPADE risulta migliore di un fattore pari a tre. Le tre motivazioni principali sono:

1. SPADE usa semplici operazioni di join su tid-list. All'aumentare delle sequenze frequenti la dimensione delle tid-list diminuisce. Comporta veloci join

2. Non vengono usati complicati Hash-tree di conseguenza si diminuisce l'overhead.

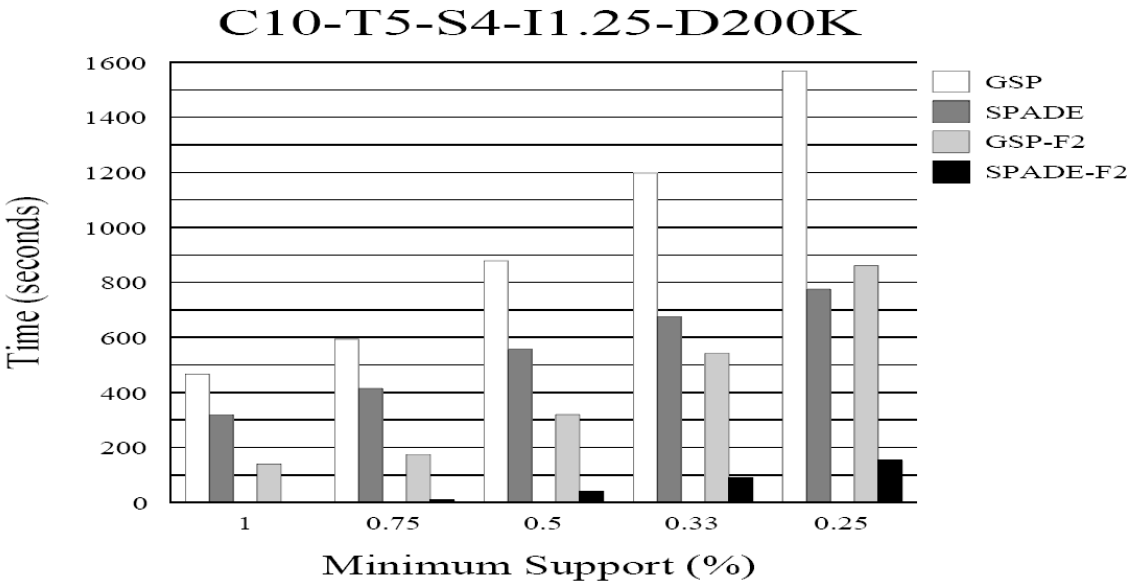

Figura 5.3: Comparazione di performance in dataset sintetico

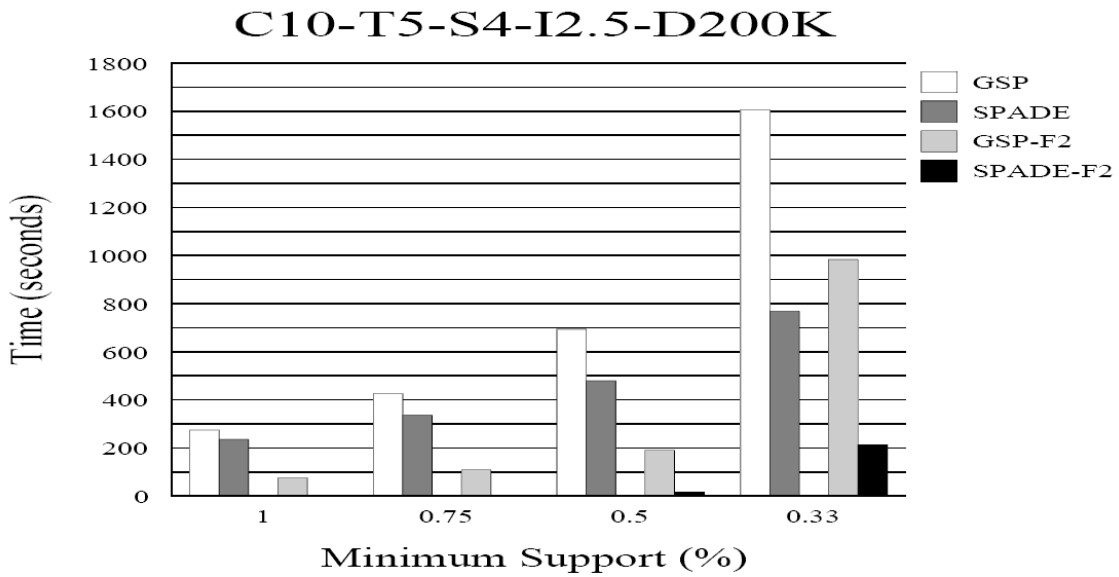

Figura 5.4: Comparazione di performance in dataset sintetico

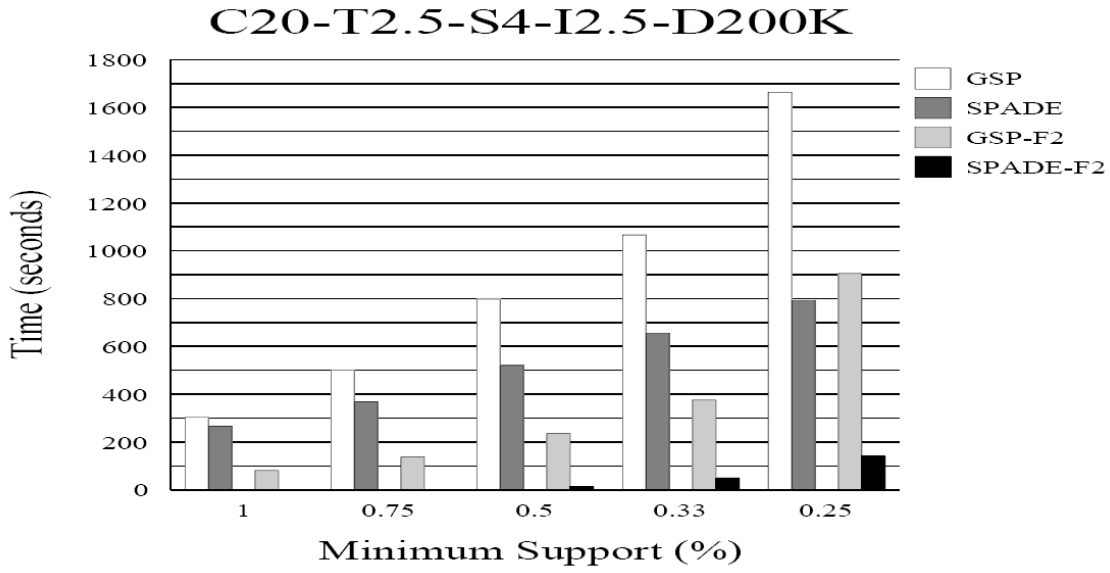

Figura 5.5: Comparazione di performance in dataset sintetico

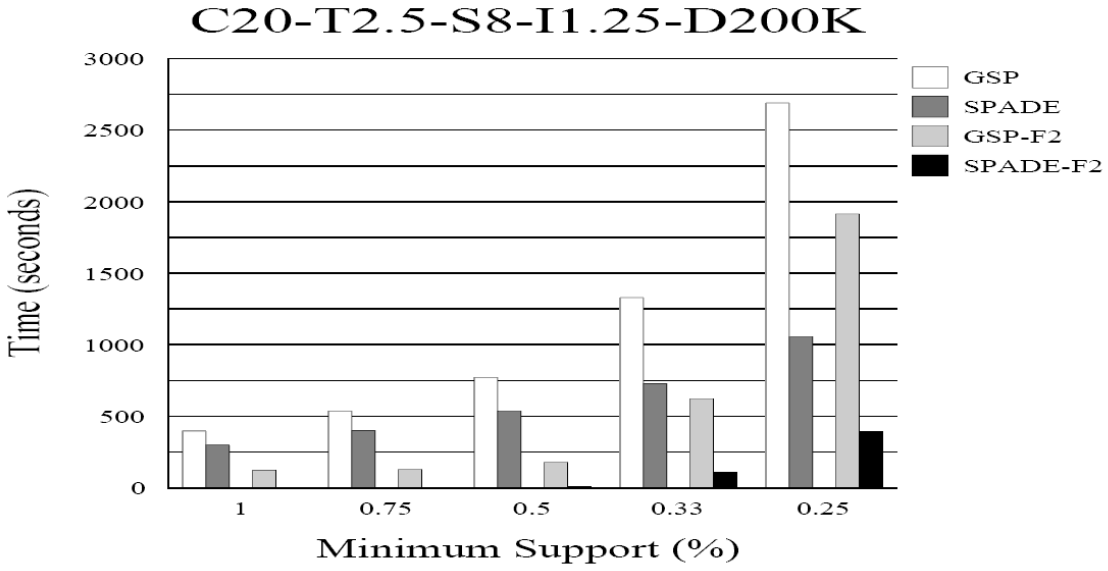

Figura 5.6: Comparazione di performance in dataset sintetico

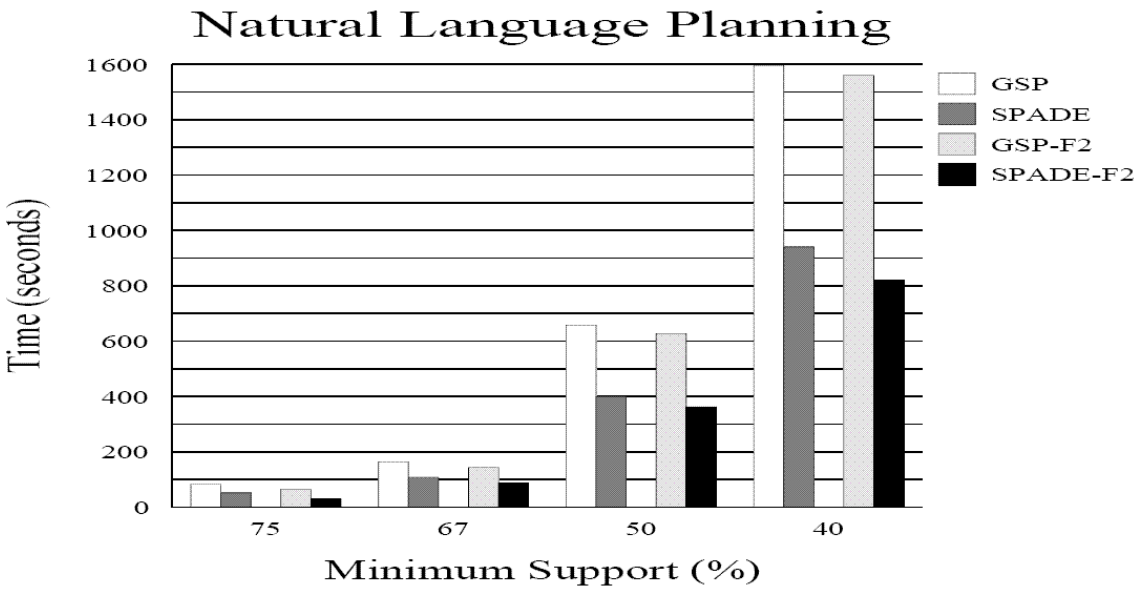

Figura 5.7: Comparazione di performance in dataset di pianificazione

3. Quando il supporto è basso molte sequenze frequenti vengono trovate. GSP fa uno scan completo ad ogni iterazione

5.4 Scalabilità

Nei grafici viene riportata la scalabilità dell'algoritmo. E' facile notare che l'algoritmo scala linearmente sui paremetri.

In figura 5.8 si confonta la scalabilità di SPADE e GSP in un dataset sintetico nel quale il numero di clienti viene fatto variare da 0.1 milioni a 1 milione. Inoltre il numero di transazioni per cliente viene fatto variare da 1 a 10 milioni rispettivamente. Nella legenda inoltre viene riportato il valore di supporto percentuale applicato. Si nota chiaramente che SPADE oltre a scalare linearmente è anche nettamente migliore a GSP.

In figura 5.9 e 5.10 viene studiata inoltra la scalabilità in maniera differente. Nel caso di figura 5.9 si tiene costante il numero di item per

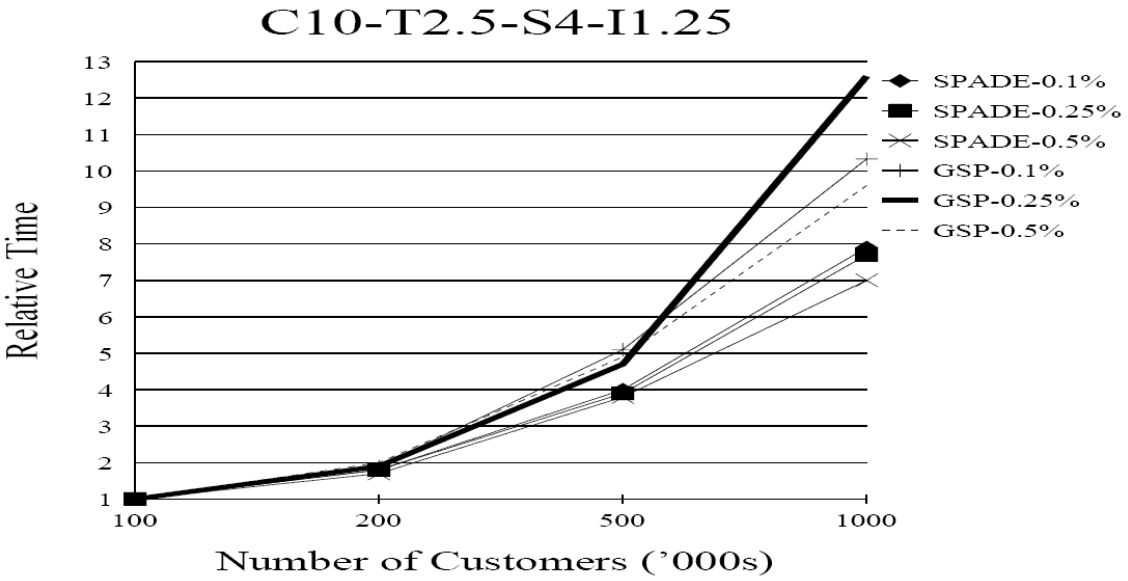

Figura 5.8: Comparazione di scalabilità in dataset Sintetico

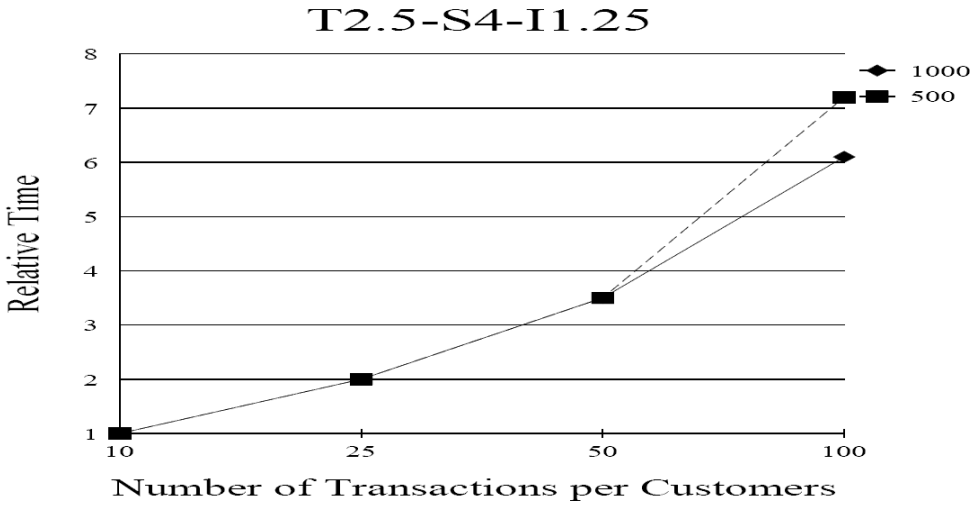

Figura 5.9: Scalabilità sul numero di transazioni per clienti

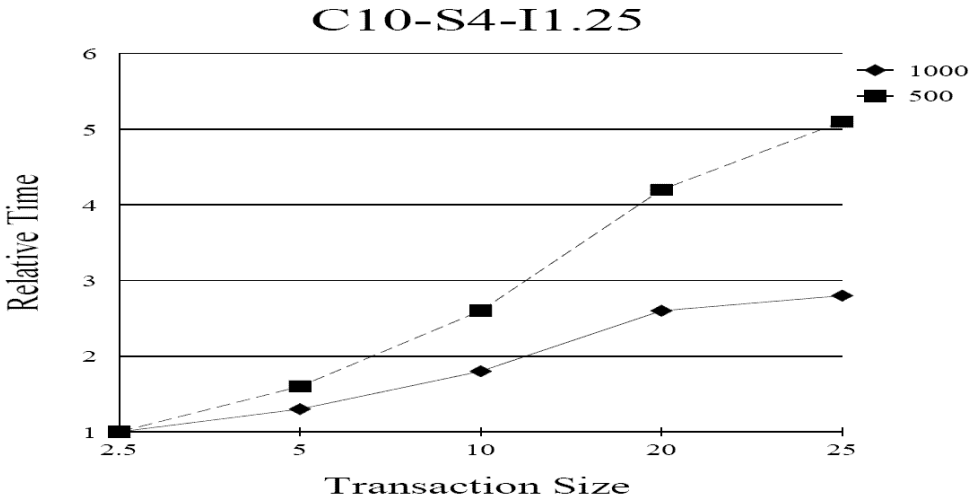

Figura 5.10: Scalabilità sulla dimensione delle transazioni

transazione e si aumenta la media delle transazioni per cliente. Nella figura 5.10 invece si tiene costante la media delle transazioni per cliente e si aumenta la media del numero di item per transazione. La dimensione del dataset rimane costante in quanto si mantiene costante la produttoria (T C D) dove T è la media della dimesione della transazione, C il numero di clienti e D la media delle transazioni per cliente. Si fa notare inoltre che il valore del supporto questa volta viene riportato in valore assoluto per evitare che il numero di sequenze frequenti aumenti di molto.

Si studia inoltre la scalabilità cambiando il numero di elementi massimali in due modi:

- tenendo tutti i parametri costanti si aumenta la lunghezza media delle sequenze frequenti potenzialmente massimali figura 5.11
- tenendo tutti i rarametri costanti si aumenta la lunghezza media degli itemset frequenti potenzialmente massimali figura 5.12

Come si può notare per alti valori di supporto il tempo comincia a decrescere con l'aumentare della dimensione dell'elemento massimale. Questo è spiegato dal fatto che ad ogni passo sempre meno elementi

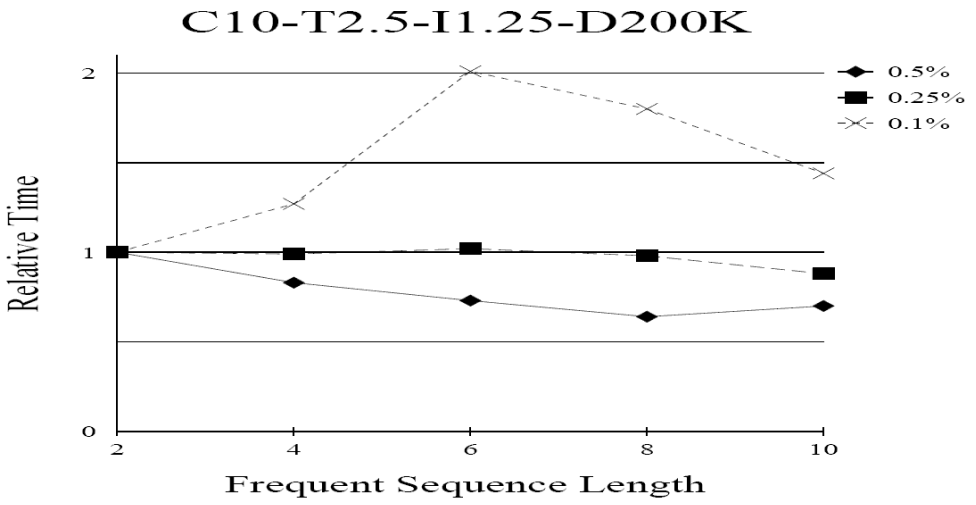

Figura 5.11: Scalabilità sulla lunghezza delle sequenze frequenti

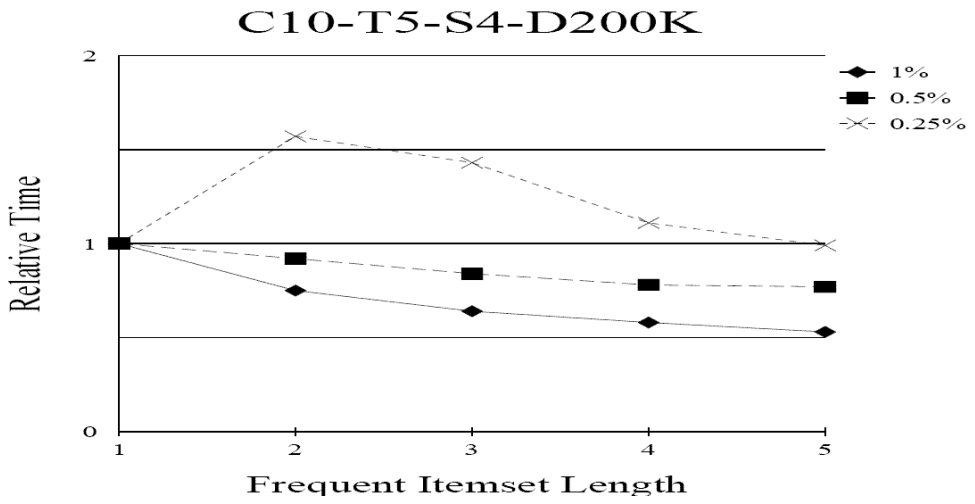

Figura 5.12: Scalabilità sulla lunghezza degli itemset frequenti

entrano nelle sequenze massimali e quindi la computazione avviene su pochi elementi.

Capitolo 6

Conclusioni

In questo approfondimento si è inizialmente introdotto il problema della scoperta delle sequenze frequenti, si è affrontata la teoria lattice, è stato introdotto l'algoritmo SPADE e lo si è comparato con l'algoritmo GSP. In conclusione si può affermare che l'algoritmo non utilizza scansioni multiple del database e non usa stutture dati complesse come ad esempio l'hash-tree. I punti forti di SPADE sono molti. Il primo ad esempio è la possibilità dell'algoritmo di decomprimere il problema in sottoproblemi usando le classi di equivalenza. Nei test si è notato inoltre che anche se non tutte le classi possono essere eseguite indipendentemente, esse nella maggioranza dei casi possono risiedere contemporaneamente in memoria. SPADE inoltre riesce a trovare tutte le sequenze in al massimo tre scansioni del database. La prima serve essenzialmente per scovare le sequenze di lunghezza 1 (gli atomi), la seconda per trovare le sequenze di lunghezza 2 e la terza per le sequenze di lunghezza maggiore di 2. L'ultimo punto di forza dell'algoritmo inoltre è l'uso di semplici operazioni di intersezione che sono facili da integrare nel database. Nelle fase di test inoltre si è potuto notare che SPADE è quasi due volte più veloce e performante di GSP. Inoltre si è notato che l'algoritmo scala bene sul numero di parametri, sul numero di clienti, sul numero di transazioni e sulla loro dimensione.

Bibliografia

[1] Zaki M. J., *Efficient Enumeration of Frequent Sequences.*,
7th International Conference on Information and Knowledge Management, Washington DC, November 1998.

[2] Davey, B. A. and Priestley, H. A. (1990)., *Introduction to Lattices and Order*,
Cambridge University Press.

[3] Agrawal, R.(1996)., *Fast discovery of association rules.*,
Advances in KDD, AAAI Press.

[4] Han J. and Kamber M.(2001)., *Data Mining: Concepts and Techniques.*,
Morgan Kaufmann.

[5] Dunham M. H., *Data Mining: Introductory and Advanced Topics.*,
Prentice Hall.