



UNIVERSITÀ DI PISA

Università degli Studi di Pisa

Facoltà di Scienze Matematiche, Fisiche e Naturali

Corso di Laurea Specialistica in Tecnologie Informatiche

Anno Accademico 2002/03

Tesi di Laurea

*"Studio, Progettazione e Implementazione di un Algoritmo per
il calcolo di Sequential Pattern"*

Candidata: Daniela Iozzia

Relatori: Prof. F. Turini

Dr. S. Ruggieri

Controrelatore: Prof. P. Ferragina

ai miei genitori

Indice

1	Introduzione	8
2	KDD e Web Mining	13
2.1	Cos'è il KDD	13
2.1.1	Il processo KDD	15
2.1.2	Data Mining	16
2.1.3	Modelli di Data Mining	17
2.2	Web Mining	19
2.3	Web Usage Mining	21
2.3.1	Sorgente dati	22
2.3.2	Preprocessing	26
2.3.3	Pattern Discovery	29
2.3.4	Pattern Analysis	32
2.4	Sistemi per il Web Usage Mining	33
2.4.1	Personalization	33
2.4.2	System Improvement	34
2.4.3	Site Modification	35
2.4.4	Business Intelligence	35
2.4.5	Usage Characterization	36
3	Sequential Pattern	40
3.1	Sequential Pattern: Stato dell'Arte	40
3.1.1	Argomenti correlati	42
3.2	Sequential Pattern Mining Problem	43
3.3	Algoritmi per inferire SP	46
3.3.1	Algoritmi generali	46
3.3.2	Classificazione degli algoritmi esaminati	56
3.3.3	Esempi	58

3.4	GSP	59
3.4.1	Struttura di base	59
3.4.2	GSP e le restrizioni	63
3.4.3	GSP rispetto al tempo e alla memoria	71
3.5	PrefixSpan	74
3.5.1	PrefixSpan rispetto al tempo e alla memoria	84
3.6	Estensioni degli algoritmi di mining	87
4	Algoritmo per inferire SP	90
4.1	Introduzione	90
4.2	Scelta dell'algoritmo	91
4.2.1	Caratteristiche	91
4.2.2	PrefixSpan con pseudo proiezione	92
4.3	Scelte implementative	94
4.3.1	Dati di input	94
4.3.2	Scelte di Progettazione e Implementazione	98
4.3.3	Test ed esperimenti	106
5	Conclusioni e Sviluppi Futuri	120
5.1	Conclusioni	120
5.2	Sviluppi Futuri	121

Elenco delle tabelle

2.1	Descrizione dei campi di un web log	24
2.2	Definizioni e termini del Web Usage	26
3.1	Database di sequenze	43
3.2	Database originale	53
3.3	Formattazione dei dati orizzontale	54
3.4	Formattazione dei dati verticale	55
3.5	Formato dei dati ottenuto usando il metodo di proiezione . . .	55
3.6	Tabella classificazioni algoritmi	59
3.7	Generazione delle sequenze candidate a partire dal seed set (L_{k-1} =insieme delle sequenze frequenti di lunghezza $k - 1$). .	60
3.8	GSP	61
3.9	Esecuzione dell'algoritmo GSP (minimo supporto = 2).	62
3.10	Generazione dei Candidati: Esempio	66
3.11	Strutture dati interne di GSP dopo che sono state generate le sequenze candidate di lunghezza 3	68
3.12	Esempio di sequenza di input	70
3.13	Esempio di gerarchia	70
3.14	Passi di GSP	73
3.15	Database di sequenze	77
3.16	Esempio utilizzando PrefixSpan (minimo supporto = 2). . . .	79
3.17	Algoritmo PrefixSpan	80
3.18	Proiezione Bi-level: matrice S	82
3.19	S-matrix in $S _4 \ 2$	82
3.20	Definizione del database nel mining incrementale	88
3.21	Un database di sequenze multidimensionale	89
4.1	Dati di input utilizzando un file di testo.	97
4.2	Dati analizzati	106

4.3	Pattern sequenziale e relativa corrispondenza	111
4.4	Pattern sequenziale e relativa corrispondenza	111
4.5	Pattern sequenziale e relativa corrispondenza	112

Elenco delle figure

2.1	Il processo di <i>Knowledge Discovery in Database</i>	37
2.2	Il processo di Web Usage Mining	38
2.3	Aree applicative del Web Usage Mining	39
3.1	Tipica restrizione di supporto decrescente con la lunghezza . . .	50
3.2	Smallest Valid Extensione (SVE)	52
3.3	PrefixSpan: Pseudo Proiezione con finestra di memoria virtuale	85
4.1	Tempo di esecuzione di PrefixSpan al variare del supporto e dei file di input.	107
4.2	Tempo di esecuzione di PrefixSpan al variare del supporto e dei file di input.	108
4.3	Tempo di esecuzione di PrefixSpan al variare del supporto e dei file di input.	113
4.4	Tempo di esecuzione di PrefixSpan al variare del supporto e dei file di input.	114
4.5	Tempo di esecuzione di PrefixSpan al variare del supporto e dei file di input.	115
4.6	Tempo di esecuzione di PrefixSpan al variare del supporto e dei file di input.	116
4.7	Tempo di esecuzione di PrefixSpan al variare del supporto e dei file di input.	117
4.8	PrefixSpan: occupazione di memoria al variare del supporto e dei file di input.	118
4.9	Esecuzione PrefixSpan su <i>Tera</i>	119

Capitolo 1

Introduzione

Al giorno d'oggi, l'uso di sistemi automatici per l'elaborazione dell'informazione è diventato un aspetto fondamentale per l'organizzazione delle aziende e per la crescita ed il mantenimento della loro competitività.

Con le moderne tecnologie di raccolta e memorizzazione delle informazioni è possibile acquisire un gran numero di dati per unità di tempo, ovviamente questo si traduce in database di dimensioni sempre maggiori. Ma raccogliere e organizzare i dati non è più sufficiente.

Il metodo tradizionale di trasformare i dati in conoscenza si basa su analisi manuali e interpretazioni. Per esempio, nell'industria farmaceutica, è comune per gli specialisti periodicamente analizzare le tendenze correnti e i cambiamenti nei dati per poi fornire un reportage dettagliato dell'analisi fatta; questi studi diventano le basi per le decisioni future e per ogni pianificazione che riguardi il management dell'azienda. Sia esso scientifico, di marketing, finanza, o qualsiasi altro campo, l'approccio classico di analizzare i dati si basa fondamentalmente su uno o più analisti, che prendono molta familiarità con i dati e fungono da intermediari tra dati e utenti e prodotti. Per molte applicazioni questa forma di sondaggio manuale dei dati è lenta, costosa e soggettiva. Visto che i dati che vengono memorizzati crescono continuamente, grazie anche alle nuove tecnologie di memorizzazione, è chiaro che questo lavoro di analisi ha bisogno di essere automatizzato, almeno parzialmente.

La necessità di accrescere le capacità dell'analisi umana per trattare enormi mole di dati che possono essere memorizzati è sia economico che scientifico. Il business usa i dati per ottenere dei vantaggi a livello di competizione, aumentare l'efficienza e fornire ai clienti servizi più validi.

Proprio questa esigenza di estrapolare dai dati ogni conoscenza utile ha determinato una spinta da parte della ricerca verso un settore dell'informatica noto come **Knowledge Discovery in Database (KDD)**.

Il *KDD* è il processo di scoperta automatica di pattern, regole, e altri contenuti regolari, prima non noti, che sono implicitamente presenti in grossi volumi di dati; il suo compito è quello di permettere di estrarre dalle basi di dati della conoscenza realmente utile e valida in modo tale che possa essere completamente compresa dall'uomo e utilizzata come **supporto alle decisioni (DSS-Decision Support System)**. Il KDD viene suddiviso in fasi e particolare risalto viene dato alla fase di **data mining**: è il passo principale del processo ed è l'applicazione di algoritmi specifici per estrarre pattern dai dati.

Gli obiettivi primari del **data mining** sono predizione e descrizione dei dati, obiettivi che vengono acquisiti usando una varietà di particolari metodi e algoritmi (Regole di associazione, pattern sequenziali, classificazione e clustering).

E' stato tale il vantaggio ottenuto dall'utilizzo del processo di *KDD* nell'analisi dei dati da pensare di poter sfruttare questo strumento in un altro importantissimo campo: **World Wide Web (WWW)**.

Negli ultimi anni, abbiamo infatti assistito ad una vera e propria esplosione dell'utilizzo di internet; esso rappresenta un grande centro di servizi per l'informazione ed è usato nei più svariati ambiti. Di conseguenza, mentre fino a pochi anni fa il web era considerato solo un grande deposito di informazioni, oggi, grazie al suo inatteso incremento di popolarità, molte compagnie ne hanno scoperto il potenziale economico e soprattutto hanno compreso quanto possa essere utile applicare il processo di estrazione della conoscenza alle risorse web.

Questa necessità ha indirizzato la ricerca verso un nuovo campo di studi: il **Web Mining**.

Il Web Mining è una nuova area di ricerca che studia l'uso delle tecniche di data mining per l'estrazione di informazione dai documenti e servizi Web. La complessità e la vastità del dominio Web ha portato alla suddivisione di questa area in tre parti: *Web Content Mining* (mira ad estrarre informazione dal contenuto delle risorse web), *Web Structure Mining* (mira ad estrarre informazione dalla topologia di interconnessione fra le pagine web) e *Web Usage Mining*.

In questa tesi siamo in particolare interessati al **Web Usage Mining**.

Questa area di ricerca si occupa di scoprire in maniera automatica i modelli di accesso degli utenti dai server Web; cioè mira ad estrarre informazioni relative all'uso delle risorse presenti in rete a partire dai dati che vengono registrati nei *file di log* di web e proxy server.

Questa esigenza è legata al conseguente sviluppo di internet; infatti, una delle varie conseguenze dello sviluppo del web è stato quello di creare sempre una maggiore competitività fra i vari soggetti economici e quindi una sempre maggiore adattività alle esigenze dei visitatori per ottenere dei miglioramenti nei siti.

Grazie allo sviluppo di tale tecnologia, è possibile quindi tenere memoria delle azioni compiute dagli utenti che navigano sui siti web; questi dati forniscono informazioni comportamentali del visitatore, vale a dire che dall'analisi di questi dati è possibile capire in linea di principio il comportamento dell'utente durante la sessione di navigazione, i suoi interessi, il tempo trascorso a visitare una pagina, le sue scelte,... Questi tipi di analisi possono produrre enormi benefici nelle scelte, nell'organizzazione, nell'efficacia ed efficienza di un sito.

Questa, ovviamente è una prerogativa irrinunciabile per molte aziende.

Un punto importante in questa tesi è stato quello di approfondire lo studio di un particolare modello di data mining che per le sue caratteristiche meglio si adatta al dominio del Web Usage Mining: i **pattern sequenziali**.

Questo modello di mining nasce dall'esigenza di estrapolare conoscenza utile e nascosta da dati che hanno un'intrinseca natura sequenziale. Quando tali dati, astrattamente, possono essere visti come una sequenza di valori o meglio di eventi, dove ogni evento ha associato un tempo di occorrenza, i pattern sequenziali rappresentano la tecnica di data mining più adatta per ottenere l'informazione più significativa considerando il fattore tempo: essi cercano di trovare le relazioni esistenti tra le occorrenze di eventi sequenziali. La scoperta di relazioni sequenziali o pattern presenti in tali dati è utile per la descrizione dei dati, predizione di eventi, identificazione di regole che caratterizzano differenti classi di dati.

Questo modello di data mining è stato intensivamente studiato e risulta ancora essere un'area di ricerca aperta a nuove scoperte. Numerosi sono gli algoritmi proposti, dagli algoritmi più semplici che si limitano ad una sem-

plice estrazione di pattern ad altri che inseriscono nel mining altre elementi come restrizioni, multidimensioni. Essi, ovviamente vista la loro natura, sono particolarmente adatti a quei domini dove l'ordine temporale dei dati è fondamentale (medico, scientifico, marketing).

Vista la particolare natura del dominio Web, e in particolare, visto che la principale sorgente dati sono file di accessi web effettuati ad un dato tempo (la sequenzialità degli eventi risulta fondamentale) risulta naturale pensare di poter sfruttare le potenzialità dei pattern sequenziali nel particolare dominio del *Web Usage Mining*.

Veniamo così all'*obbiettivo* di questa tesi: *studiare le proposte esistenti in letteratura, scegliere e implementare un algoritmo per inferire pattern sequenziali da database di accessi a web e proxy server*.

Questo obiettivo è stato raggiunto attraverso un'ampia fase di ricerca in letteratura del materiale riguardante l'argomento, studio dei *sequential pattern* in generale (cioè senza tener conto di nessuna loro campo di applicazione in particolare) e relative problematiche; un'attenta analisi dei vari algoritmi per il loro calcolo proposti in letteratura dai più recenti a quelli meno e, da questo studio, è stata realizzata una loro classificazione in base alle diverse caratteristiche riscontrate; studio delle caratteristiche del KDD e in particolare del dominio web e del Web Usage Mining; scelta di un algoritmo adeguato alle nostre esigenze; progettazione e implementazione dell'algoritmo; sperimentazione.

Il nome dell'algoritmo scelto è **PrefixSpan**([11]). Questo algoritmo si basa su un approccio *dividi et impera* in quanto partiziona ricorsivamente il database dei dati in porzioni più piccoli che vengono analizzati separatamente. Il partizionamento avviene per mezzo di una tecnica di proiezione basata su prefisso: la proiezione del database diminuisce i dati da analizzare al passo successivo rendendo così l'algoritmo più veloce. Gli autori presentano tre diversi metodi di proiezione dell'algoritmo: *level-by-level*, *bi-level* e *pseudo* proiezione. Quella da noi prescelta è la pseudo-proiezione che è una versione *main-memory* dell'algoritmo.

Per l'implementazione dell'algoritmo è stato scelto come linguaggio *Java*,

e in particolare, vista la nuova spinta dell'informatica verso l'*hyperThreading* è stata realizzata una sua versione multiThreading (nell'algoritmo implementato il numero di thread attivabili è un parametro).

Struttura della tesi

La tesi si articola nel seguente modo:

Capitolo 1. Introduzione. Descrizione generale del contesto di ricerca e introduzione al lavoro svolto.

Capitolo 2. Viene presentata una panoramica sullo stato dell'arte. In particolare viene trattato il processo KDD, i suoi obiettivi, le aree applicative soffermandoci sulla fase di Data Mining. Passando al dominio Web viene descritto il Web Mining e in particolare il Web Usage Mining, anche in questo caso vengono descritti gli obiettivi e le aree applicative.

Capitolo 3. Viene presentata una panoramica dello stato dell'arte dei pattern sequenziali: caratteristiche, domini di applicazioni, algoritmi proposti e problematiche affrontate. Individuazione di caratteristiche importanti nei diversi algoritmi esaminati e una loro classificazione. Analisi approfondita di due algoritmi in particolare.

Capitolo 4. Scelta di un particolare algoritmo, *PrefixSpan con pseudo proiezione*, e motivazioni. Discussioni sull'applicazione dei pattern sequenziali al dominio Web (natura dei dati da analizzare). Scelte implementative. Descrizione dello strumento realizzato. Test ed esperimenti fatti.

Capitolo 5. Conclusioni e futuri sviluppi. PrefixSpan verrà adattato all'interno di un recente sistema di data mining, il **KDDML**. Quest'ultimo è un ambiente di sviluppo per problemi di estrazione di conoscenza scritto interamente in Java in grado di supportare l'intera fase di Data Mining ([24]).

Capitolo 2

KDD e Web Mining

2.1 Cos'è il KDD

Nell'era di internet e dei grandi database, i paradigmi fondamentali dell'analisi classica dei dati hanno subito e continuano a subire enormi trasformazioni. La massa dei dati memorizzati nei sistemi informativi esistenti sono arrivati ad avere delle dimensioni impensabili negli anni passati. Questa crescita vertiginosa di informazioni digitali è dovuta alla progressiva ed inesorabile diffusione dei computer, al drastico abbassamento del rapporto prezzo/capacità dei supporti di memorizzazione, ad un aumento della potenza di calcolo dei moderni processori.

La tecnologia per la memorizzazione e la consultazione dei database è arrivata ad un punto tale che, quelli che fino ad adesso sono stati i principali obiettivi da perseguire, registrazione istantanea di nuovi dati e successivamente una loro ricerca estremamente veloce al momento delle interrogazioni, sono stati praticamente raggiunti.

In qualunque settore coinvolto, sia esso finanziario, delle telecomunicazioni, o ancora quello medico, industrie e associazioni analizzano i dati raccolti per migliorare il proprio lavoro. Per analizzare tali dati, una volta erano sufficienti delle tecniche statistiche ad hoc. Oggi, la loro enorme mole, la velocità con cui i database vengono modificati e inoltre, spesso proprio la loro complessità, hanno reso necessario lo sviluppo di nuovi approcci; così, negli ultimi anni, grazie al contenuto di diverse discipline, sono apparse tecnologie che hanno messo a disposizione strumenti per correggere, ripulire e soprattutto analizzare questi dati.

Knowledge Discovery in Databases, in sigla **KDD**, si presenta come una disciplina emergente, in via di sviluppo, nata per fronteggiare e gestire la quantità enorme di informazione contenuta nelle basi di dati.

La ragione per cui il KDD ha attratto gran parte dell'attenzione nell'industria dell'informazione nei recenti anni è dovuta alla disponibilità di grosse quantità di dati e all'imminente bisogno di trasformare questi dati in informazione utile.

Una definizione formale del KDD data in [6] è la seguente: *“Per KDD si intende il processo non banale di identificazione di modelli validi, nuovi, potenzialmente utili e comprensibili sui dati”*.

Questa definizione è solitamente adottata dagli studiosi in quanto consente di mettere in luce l'aspetto inferenziale del processo nonché le caratteristiche dei modelli o pattern in termini di validità, novità, utilità e comprensibilità. Il compito di questa disciplina è quindi, quello di permettere di estrarre dalle basi di dati della conoscenza realmente utile, valida in modo tale che essa possa essere interamente compresa dall'uomo e utilizzata come **supporto alle decisioni** (DSS).

Come ci mostra la figura 2.1, il KDD è un processo costituito da una serie di stadi interattivi che manipolano e trasformano i dati .

All'interno dell'intero processo di estrazione della conoscenza, in questi ultimi anni gli studi si sono focalizzati su una fase del processo KDD, nota con il nome di *data mining*.

*Il **Data Mining** (DM) è il passo principale del processo e si occupa della scoperta automatica di pattern e dello stesso sviluppo di modelli predittivi.*

Proprio per l'importanza rivestita dal DM nel KDD, nella pratica spesso si tende a far coincidere i due processi; in realtà non è così, infatti il KDD non consiste solamente in una semplice applicazione di algoritmi sofisticati di data mining ma comprende diverse fasi fondamentali. In questa tesi ci riferiremo ad un modello di KDD che identifica gli elementi necessari e sufficienti per supportare le operazioni richieste per un processo di acquisizione delle conoscenze, e si considererà il Data Mining come l'insieme delle tecniche, dei tools e degli algoritmi utilizzati per l'analisi dei dati.

Andiamo adesso ad analizzare quanto detto in dettaglio.

2.1.1 Il processo KDD

Il processo di acquisizione dei dati può essere suddiviso in 4 passi (Figura 2.1).

Il processo è interattivo ed iterativo. E' *interattivo* in quanto necessita di un intervento umano su alcune decisioni, soprattutto per quanto riguarda la scelta del modello dei dati da utilizzare e dell'algoritmo specifico; è *iterativo* perchè richiede l'applicazione di una o più fasi precedenti se i risultati ottenuti in una determinata fase non sono soddisfacenti.

Il processo KDD consiste nell'integrare i dati provenienti da varie sorgenti informative in un unico deposito di dati (*data warehouse*), analizzare tali dati attraverso varie operazioni di elaborazione analitica (OLAP *On-line Analytical Processing*), quali aggregazioni e frazionamento, per ottenere informazioni sintetiche, e infine applicare tecniche di *Data Mining* per ricavare regolarità nei dati e pervenire ad eventuali modelli interpretativi. La conoscenza ottenuta viene impiegata in un Sistema per il Supporto alle Decisioni (DSS) e viene acquisita come il risultato dei seguenti passi:

1. **Consolidamento dei dati.** L'obiettivo è di prelevare i dati da sorgenti eterogenee (DB relazionali, DB deduttive, files di testo) per costruirne una visione uniforme a prescindere da quale tipo di sorgente siano stati estratti. Il risultato di questo passo è la creazione di un database, detto **Data Warehouse**, utilizzato per il supporto alle decisioni e mantenuto separato rispetto al database operativo dell'organizzazione, che è invece utilizzato per le normali operazioni di transaction processing.
2. **Selezione e preprocessing.** Gli algoritmi di data Mining necessitano di lavorare su dati che hanno un formato ben preciso. Le informazioni, tuttavia, provengono da sorgenti eterogenee, e quindi l'integrazione prodotta nella fase precedente può aver portato a possibili inconsistenze, ridondanze o errori. Per questo motivo, è necessario un'intera fase che si occupa di ripulire i dati e di prepararli per la fase di DM (**DataCleaning**), di rimuovere le inconsistenze e le ridondanze (**Data Integration**), di eliminare attributi non significativi ai fini del processo o di campionare un insieme di tuple (**Data Reduction**).
3. **Data Mining.** I dati trasformati sono adesso esplorati utilizzando una o più tecniche di Data Mining in modo da poter estrarre l'informazione desiderata. Questo stadio include la scelta del tipo di meto-

do/algoritmo da utilizzare per la ricerca dei pattern e delle regolarità tra i dati. Mentre si effettua questo passo, può essere necessario accedere a dati addizionali e/o effettuare ulteriori trasformazioni sui dati originali, ritornando quindi alle fasi precedenti. Vista l'importanza svolta dal DM nel processo di estrazione della conoscenza, riprenderemo in seguito questo argomento per trattarlo in dettaglio.

4. **Interpretazione e valutazione.** I pattern e i modelli estratti dal sistema vengono interpretati per ottenere della conoscenza utile che successivamente può essere utilizzata come supporto alle decisioni dell'analista o del manager. Questa fase ha sostanzialmente due scopi:

- filtrare la conoscenza ottenuta;
- presentare l'informazione filtrata mediante tecniche di visualizzazione.

2.1.2 Data Mining

Come detto in precedenza il data mining rappresenta la fase fondamentale del processo KDD; essi spesso vengono usati come sinonimi, a testimonianza di quanto questa fase sia importante.

L'obiettivo principale del Data Mining è di individuare modelli descrittivi i dati che siano facilmente interpretabili dall'uomo in modo da fornire un valido supporto alle decisioni. Tale estrazione deve avvenire in modo completamente *automatico*, in quanto gli strumenti e le tecniche *manuali* di analisi non sono in grado di supportare e di gestire l'enorme quantità di dati utilizzati per l'estrazione.

Abbiamo affermato che lo scopo del Data Mining è di trasformare i dati in informazione utile. Per scoprire patterns e relazioni ci si avvale di tecniche di analisi. Tali tecniche sono essenzialmente di due tipi:

- Tecniche descrittive: descrivono i pattern nei dati esistenti al fine di aiutare chi deve prendere una decisione;
- Tecniche previsionali: usano dati di cui già sono disponibili i risultati per poter prevedere il valore che in futuro assumerà la variabile dipendente.

In conclusione possiamo dire che il compito fondamentale del Data Mining è quello di applicare algoritmi specializzati per inferire conoscenza. Sarà poi

compito del KDD provvedere a interpretare e presentare i modelli estratti dalla fase del Data Mining.

Nel DM tali modelli possono essere classificati in base al tipo di risultato che riescono a ricavare; vediamo quali sono i principali modelli sui dati disponibili nella ricerca.

2.1.3 Modelli di Data Mining

Le tecniche utilizzabili sono varie e, di conseguenza, anche gli algoritmi che li implementano. La scelta dipende principalmente dall'obiettivo che si vuole raggiungere e dal tipo di dati da analizzare.

Vediamo quali sono i modelli di Data Mining più utilizzati.

Regole di associazione

Le **Regole di Associazione(RdA)**, costituiscono un modello per rilevare regolarità fra i dati. L'esempio di applicazione più noto riguarda l'analisi sui prodotti venduti all'interno di un supermercato, detta **Market Basket Analysis** utilizzata per individuare le correlazioni tra i prodotti (o le categorie di prodotti) che un cliente acquista mediante l'analisi dei dati contenuti negli scontrini di cassa. Le informazioni trovate possono aiutare il settore marketing a progettare sconti o promozioni particolari, oppure a disporre i prodotti nelle posizioni più strategiche, in accordo alle preferenze dei clienti. Informalmente, una regola di associazione è un legame di causalità valido tra gli attributi dei record di un database, ossia una espressione del tipo $X \implies Y$, dove X e Y sono due insiemi di attributi. Considerando, per esempio, le transazioni nel contesto del supermercato, una possibile regola di associazione è la seguente:

“il 70% dei clienti che comprano pane e burro comprano anche il latte”.

Ogni regola è caratterizzata da due parametri:

- il supporto della regola; denota la frequenza degli item contenuti nella regola all'interno delle transazioni.
- la confidenza della regola; denota la percentuale delle transazioni contenenti X che contengono anche Y , e dà quindi una stima della probabilità condizionata di Y data X .

Classificazione

La classificazione consiste nell'esaminare il comportamento di un oggetto di interesse e di assegnarlo in una classe predefinita. Applicazioni tipiche si possono riscontrare nel marketing, per classificare i clienti in base alle preferenze, nelle diagnosi mediche per la classificazione delle malattie in base ai sintomi riscontrati dai pazienti, nei settori bancari ed assicurativi per classificare i clienti in base al loro grado di fedeltà.

Il processo di classificazione si articola sempre attraverso due fasi:

1. **Costruzione del modello:** una volta scelto l'attributo dei dati che deve essere stimato, detto *attributo di classificazione*, occorre selezionare un database campione (*training set*) costituito da esempi pre-classificati, nel quale ogni tupla contiene un preciso valore per l'attributo di classificazione. Attraverso il training set e un algoritmo di classificazione si può costruire il modello desiderato che può assumere varie forme, ad esempio regole di classificazione, alberi di decisione, reti neurali, oppure formule matematiche.
2. **Utilizzo del modello:** una volta costruito il modello, esso può essere utilizzato per classificare nuove tuple di cui ignoriamo l'attributo di classificazione.

Clustering

Il clustering è una tecnica di analisi dei dati che raggruppa gli oggetti in base a misure di similitudini o in base a qualche criterio prestabilito.

Il clustering differisce dalla classificazione in quanto non si basa sul calcolo di un singolo attributo, ma lavora su elementi definiti da un certo numero di attributi e assegna gli oggetti ad un gruppo, detto *cluster*, in base alle loro somiglianze.

La qualità di un cluster si misura sempre in funzione di due parametri:

- alta similarità di oggetti;
- bassa similarità tra oggetti appartenenti a cluster diversi.

Pattern Sequenziali

Il tempo è un concetto importante nelle basi di dati; anche all'interno del DM è quindi importante utilizzare modelli che prevedono una dimensione

temporale. Per questo motivo, sono stati introdotti in letteratura delle varianti dei modelli attuali che permettono di esprimere il fattore tempo.

Uno studio condotto in questa direzione è quello svolto sui **Pattern Sequenziali**, che sono un modello di Data Mining che permette di estrarre tutte le sottosequenze interessanti da una sequenza di transazioni, il cui supporto supera una certa soglia specificata dall'utente.

Un possibile esempio di applicazione, con riferimento al marketing, può essere lo studio condotto sul comportamento di acquisto nel tempo dei clienti; l'informazione estratta in questo contesto può assumere la seguente forma: *“il 95% dei clienti ha acquistato prima il prodotto X e successivamente il prodotto Y”*.

2.2 Web Mining

Negli ultimi anni abbiamo assistito e stiamo assistendo ad una continua esplosione dell'utilizzo di internet in Italia e nel mondo: oggi sempre più persone si rendono conto delle grandi potenzialità della rete.

Il web è essenzialmente un centro di servizi per l'informazione, esso è enorme, distribuito e globale, utilizzato nei più diversi ambiti, quali l'informazione giornalistica, la pubblicità e l'informazione ai consumatori, la finanza, la formazione, il commercio elettronico, i servizi ai cittadini; la quantità, la qualità e la dinamicità dell'informazione su web rendono pertanto indispensabili meccanismi efficaci di indicizzazione e ricerca e contemporaneamente rendono tali meccanismi difficili da realizzare e tenere aggiornati. Analizzando in breve le caratteristiche di internet al momento si noti che:

- cresce rapidamente e l'informazione è continuamente aggiornata;
- è utilizzato da una comunità di utenti ampiamente variegata con background, interessi e usi completamente diversi;
- solo una piccola porzione dell'informazione è veramente rilevante o utile.

Di conseguenza mentre fino a pochi anni fa il Web era considerato solo un enorme deposito di informazioni, oggi, grazie al suo inatteso incremento di popolarità, molte compagnie ne hanno scoperto il potenziale economico e soprattutto hanno compreso quanto possa essere determinante applicare ricerche di Data Mining a questa fertile area.

Da qui nasce il bisogno di estrarre informazione utile dall'immensa mole di dati contenuti nella rete, in pratica da qui nasce l'esigenza di applicare il processo di Knowledge Discovery a transazioni Web.

Con il termine **Web Mining** si definisce l'utilizzo delle tecniche di Data Mining per la scoperta e l'estrazione automatica delle informazioni della rete. Il Web Mining studia l'uso delle tecniche di data mining per l'estrazione di informazione (Knowledge Discovery) dai documenti e servizi Web.

Se in generale il DM affronta il problema di scoprire regolarità nascoste nei dati, il Web Mining affronta il problema di individuare regolarità nella struttura e nel contenuto delle risorse Web, e nei pattern di accesso alle risorse Web.

Il Web Mining può essere suddiviso in tre principali categorie:

1. *Structure Mining*: mira ad estrarre informazioni dalla topologia di interconnessione fra le pagine Web. Tali informazioni sono utilizzati per diversi scopi: categorizzare i siti, scoprire relazioni di similarità fra i siti, valutare la rilevanza delle pagine. I modelli sono costruiti analizzando i riferimenti ipertestuali, il grafo di connettività del Web che essi formano e le altre informazioni culturali contenute nei documenti HTML o XML.
2. *Content Mining*: mira ad estrarre informazioni dal contenuto delle risorse Web. Le tecniche di mining possono essere applicate a dati di natura diversa: testuali, semistrutturati (HTML, XML), strutturati (tabelle relazionali, biblioteche digitali), dinamici (risultati di query a basi di dati). I modelli estratti sono utilizzati ad esempio per classificare o categorizzare le pagine Web, per estrarre keyword o sequenze frequenti di keyword, per inferire lo schema concettuale di una collezione di dati semistrutturati, etc.
3. *Usage Mining*: mira ad estrarre informazioni relative all'uso delle risorse Web, a partire dai dati di *log* generati dalle interazioni degli utenti con il Web. Tali dati generalmente derivano dai log di Web server e di proxy server, cookies¹, profili utenti, dati di registrazione a servizi Web,

¹Meccanismo di gestione dello stato di una sessione; sono mantenuti sul computer client per tracciare l'utente che accede al sito. Questo tipo di file sono legati alla macchina che accede alla rete pertanto sorgono problemi se utenti multipli accedono al sito dallo stesso terminale.

sessioni utente, interrogazione utente, bookmarks. L'analisi dell'uso del Web può essere applicata alla definizione di strategie intelligenti di caching e prefetching di risorse Web presso proxy o web server, all'identificazione di utenti e sessioni utente, alla ristrutturazione automatica di sito Web (*adptive web sites*), ai sistemi di raccomandazione e di gestione della clientela nell'*e-commerce*². L'analisi del traffico del Web viene invece utilizzata per migliorare i flussi e determinare la migliore dislocazione dei server di replicazione.

In questa tesi ci restringeremo in particolare al campo di studi del *Web Usage Mining*.

2.3 Web Usage Mining

Il *Web Usage Mining* è l'applicazione delle tecniche di Data Mining per scoprire i pattern d'uso dei dati Web allo scopo di meglio capire e servire i bisogni delle applicazioni basate su rete; dal momento che esso cerca di estrarre pattern interessanti dai log di accesso Web nel traffico del WWW e proprio per la natura dei dati analizzati, spesso, il *Web Usage Mining* viene anche chiamato *Web Log Mining*.

Visto il potenziale di applicazione, il *Web Usag Mining* ha visto una rapida crescita di interesse sia da parte della ricerca che delle comunità di natura più pragmatica.

In letteratura, sono state individuate tre fasi distinte che compongono il processo di estrazione della conoscenza da risorse Web:

- *Preprocessing*
- *Pattern Discovery*
- *Pattern Analysis*

Adesso analizzeremo per prima quali sono le sorgenti dati del Web Usage Mining e successivamente ciascuna delle sue fasi.

²Electronic-commerce: insieme delle iniziative adottate per supportare l'attività commerciale di un'azienda svolte tramite Internet; Solitamente ci si riferisce all'attività di vendita o di acquisto di un prodotto o di un servizio in cui i soggetti interagiscono elettronicamente.

2.3.1 Sorgente dati

La principale sorgente dati per il *Web Usage Mining* sono i log di accesso web; essi sono mantenuti dagli agenti web per monitorare il traffico web, gestire le cache, memorizzare il comportamento dei visitatori mentre navigano in rete.

Questi dati vengono raccolti da diversi fonti, in particolare, possiamo individuare tre livelli in cui i dati sono memorizzati: livello client, livello server e livello proxy. Ogni sorgente dati differisce per formato, accuratezza, scopo e metodo di implementazione.

I log del *livello cliente* descrivono l'interazione di un singolo utente con un insieme di siti, e quindi essi mantengono le informazioni più accurate relative al comportamento degli utenti. Essi possono essere implementati usando agenti remoti o modificando il codice sorgente di un browser esistente. L'implementazione dei metodi per la raccolta dei dati dal lato client richiede la cooperazione dell'utente sia per quanto riguarda il rendere attivo le funzionalità di un agente remoto, sia per l'uso volontario di un browser modificato. Questa collezione di dati dal lato client ha dei vantaggi rispetto a quella del lato server, essa, infatti, migliora i problemi di identificazione delle sessioni e del caching. La maggiore difficoltà nell'usare questo metodo è convincere gli utenti sull'uso del browser per le loro attività quotidiane.

Un proxy server agisce come un livello intermedio di caching tra i browser del cliente e i server Web. La sua primaria funzione è quella di servire sia come misura di sicurezza per bloccare utenti indesiderati e sia come risorsa cache per ridurre il traffico della rete riusando i file acceduti più recentemente. I log a *livello proxy* sono simili a dei server web e descrivono l'interazione di molteplici utenti con un insieme di diversi siti. Il processo di logging è automatico e richiede meno intervento rispetto al logging del livello cliente. Il suo formato è dipendente dal logging software. La sua accuratezza è comunque diminuita rispetto alla cache del livello client poichè alcune richieste non sono ricevute dal proxy ma sono servite dai file più recentemente acceduti memorizzati nel computer del cliente. Comunque analizzare questo tipo di log può essere utile per caratterizzare il comportamento di un gruppo di utenti che condividono un comune proxy server o per studiare metodi che hanno lo scopo di migliorare la strategia di caching.

I log a *livello server* rappresentano probabilmente la principale sorgente dati per il web usage mining poichè essi esplicitamente memorizzano il comportamento dei visitatori del sito. Questo tipo di log descrivono l'accesso concorrente di un insieme di utenti ad un singolo sito web. Comunque, esistono degli inconvenienti quando si usano i web log a livello server:

- dovuti alle cache del cliente e del livello proxy, non tutte le richieste vengono catturate nei log di questo livello;
- la durata in termini di tempo di visita di una pagina può essere non precisa: se una richiesta è servita da una cache cliente o dalla cache del proxy, allora il tempo di visita della sua pagina precedentemente verrà interpretato come essere più lungo di quanto effettivamente è.
- quando l'identificatore di un utente non è disponibile e i clienti diciamo sono nascosti dietro il proxy, molte richieste saranno memorizzate con lo stesso nome host (nome host del proxy). Come risultato, le visite possono sembrare errate con un tempo di visita brevissimo.

La maggior parte dei server web dà la possibilità di memorizzare i file di log in “Common Log Format” (CLF), “Extended Common Log Format” (ECLF) o formati proprietari.

La tabella 2.1 descrive i campi in entrambi i formati.

Ogni CLF log entry ha il seguente formato:

```
HostID rfc931 authUser [date offset] “method URI protocol” status bytes
```

ECLF log inoltre contiene anche i campi della pagina referrer e agent.

Quello che segue è un esempio di ECLF log entry. Si noti che i campi *rfc931* e *authUser* sono vuoti (indicati dal valore “-”).

```
142.107.30.180 - - [25/Mar/1999:23:01:41 -800] “GET my.html HTTP/1.1”  
200 4219 -Index.html Mozilla(IE4.2, Win95)
```

L'informazione ottenuta dalle sorgenti dati descritti prima possono essere usati per costruire e identificare diverse astrazioni sui dati, come *utente*, *sessione server*, *episodi*, *clickstream* e *page view*. Per evitare possibili problemi di inconsistenza riguardo a questi termini, adottiamo le definizioni di termini

Termine	Descrizione
host remoto	nome host remoto o indirizzo IP
Rfc931	login name remota del cliente
identificazione utente	nome del cliente identificato dal server
Data	data e tempo di richiesta
Offset	offset del tempo locale rispetto al tempo di Greenwich
Metodo	Metodo di richiesta (Get, Post, etc)
Protocollo	protocollo di comunicazione http usato dal cliente
Status	status del server http inviato al cliente
Bytes	numero di bytes trasferiti
Referrer	URI da cui è stato originato
Agent	Sistema Operativo e browser software del cliente

Tabella 2.1: Descrizione dei campi di un web log

Web pubblicata dal *World Wide Web Committee Web Usage Characterization Activity (W3C WCA)* ([27]) che è rilevante in ambito del web usage mining. Questi termini sono elencati nella tabella 2.2.

Un *utente* (user) è identificato come un singolo individuo che accede a dei file da differenti web server attraverso un browser. Nonostante possa sembrare un compito semplice, in pratica è molto difficile identificare in modo unico e in maniera ripetuta un utente, come detto prima, infatti, a causa dei proxy server molti utenti possono avere lo stesso host name e spesso i campi *rfc931* *authUser* nei web log possono essere vuoti. In tali casi, solamente host name con la combinazione di informazioni di un agente, se a disposizione, sono usati per identificare l'utente.

Una *page view* consiste di un insieme di risorse (uno o più file html, grafici, etc) usati per soddisfare una URI³ richiesta dal browser di un utente. Ogni risorsa sarà memorizzata separatamente in un web log. Una page view è generalmente il risultato di un singolo click di mouse di un utente su un hyperlink. Alcune page view contengono molteplici strutture, questo rende

³*Uniform Resource Identifier* (URI) è una definizione più generale che include il noto *Uniform Resource Locator* (URL).

il lavoro di identificazione ancora più difficile.

Un *click-stream* è definita come una lista ordinata in base al tempo (sequenziale) di richieste di page view. I dati disponibili dal lato server non sempre riescono a dare abbastanza informazione per ricostruire l'intero flusso di click per un sito. Il *click-stream* di un utente sull'intero web è chiamato *sessione utente*, quest'ultima è definita come il sottoinsieme di click su un particolare server eseguiti appunto da un utente, approfondiremo meglio l'argomento. L'insieme delle page view in una sessione utente per un particolare sito web è definita come *server session* o più comunemente *visita*.

All'interno del click-stream di una visita o di una sessione utente, un *episodio* è definito come un insieme di click che sono correlati sequenzialmente o semanticamente. Il tipo di relazione dipende dagli obiettivi degli studi.

ClickStream Analysis

Particolare rilievo va dato al *ClickStream Analysis*. La sequenza dei click eseguiti dagli utenti sulle pagine che compongono il sito rappresenta uno dei migliori punti di partenza di ogni valutazione. L'utente, ogni volta che visita una pagina Web o che clicca su un link, produce, quindi, dei file nei quali vengono memorizzati i dati da lui visitati, permettendo così di ricostruire il percorso dettagliato seguito lungo la navigazione. Da questa analisi si può capire quali sono le pagine visitate, il tempo che vi viene trascorso, la provenienza e le vie di uscita. Questo tipo di analisi, inoltre, aiuta ad individuare i contenuti del sito che maggiormente attirano gli utenti. Grazie a queste informazioni, diventa possibile, per esempio, migliorare il sito Web adeguandolo ai gusti e alle reali esigenze degli utenti, migliorare la struttura stessa del sito, etc.

In conclusione, il *ClickStream Analysis* rappresenta sicuramente una delle fonti più determinanti al servizio di tutte le applicazioni del Web Usage Mining.

Termine	Descrizione
<i>Server</i>	ruolo svolto da un'applicazione che fornisce servizi e risorse
<i>Proxy</i>	applicazione intermediaria che agisce sia come server sia come client
<i>Client</i>	ruolo svolto da un'applicazione quando richiede risorse da un server
<i>Utente</i>	persona che usa un'applicazione cliente per interagire e richiedere risorse da un server
<i>sessione utente</i>	insieme delle scelte dell'utente ("click") lungo uno o più server
<i>sessioni server/visita</i>	insieme delle scelte dell'utente ("click") dal punto di vista di un singolo server
<i>Episodio</i>	sottoinsieme di click (semanticamente correlati) dall'utente memorizzati in una sessione utente
<i>pagina Web</i>	collezione di risorse identificate da un singolo URI
<i>"Page view"</i>	pagina web visitata in una specifica applicazione
<i>"Click-stream"</i>	serie di pagine visitate sequenzialmente da un utente

Tabella 2.2: Definizioni e termini del Web Usage

2.3.2 Preprocessing

La fase di preprocessing è importante in ogni progetto di data mining, ma diventa fondamentale nel dominio di web mining. Il principale obiettivo di questo passo è quello di creare degli oggetti che possono essere oggetto di estrazione di conoscenza per il Knowledge Discovery nonostante la presenza di ambiguità e incompletezze nei dati. Questo passo dipende fortemente dalla sorgente dati. In questa sezione ci focalizzeremo in particolare sulle tecniche usate per processare i web log file a livello server. Il risultato di questa fase è un insieme di *sessioni utente* che definiscono la sequenza delle pagine riferite da ogni singolo visitatore durante una sessione di lavoro. Tali sessioni diventeranno successivamente l'input per la fase di estrazione della conoscenza.

La fase di preprocessing include i compiti di:

- *Data Cleaning.* Le tecniche di web mining sono utili quando i dati contenuti nei log forniscono un'accurata visione degli accessi al sito

web; per questo motivo, sono importanti delle operazioni di cleaning (pulitura) per eliminare dai log in ingresso le informazioni ridondanti, irrilevanti o inconsistenti. Il problema del data cleaning è legato al fatto che ogni singolo oggetto in una pagina richiesta corrisponde ad un entry log separato, questo implica che, quando un utente interagisce con una pagina contenente testo, immagini, audio, etc, vengono prodotti vari entry log separati e al fine di ottenere un disegno del comportamento dell'utente non ha senso tener conto di richiesta a file che l'utente non ha esplicitamente richiesto. Per questa motivazione si cerca di ripulire i log da quei file che non hanno relazione con il sito analizzato o con gli obiettivi dell'analisi.

- *User Identification.* Una volta che i dati sono stati ripuliti, occorre innanzitutto raggruppare tutte le pagine riferite da un singolo visitatore (*identificazione degli utenti*) e, successivamente, suddividere queste pagine in singole unità di lavoro (*identificazione di sessioni*). Nei migliori casi si possono utilizzare i campi *rfc931* e *authUser* per identificare accuratamente un utente. Questi due compiti sono enormemente complicati dalla presenza di cache locali e proxy server, vediamo perchè. Per migliorare la performance e minimizzare il traffico Web, molto Web Browser inseriscono in cache le pagine che sono state richieste; questo significa che, quando un utente effettua un backtracking, cioè richiede una pagina visitata precedentemente, la pagina presente in cache viene visualizzata immediatamente e il server non deve ripetere l'accesso nuovamente; i proxy server, a loro volta, forniscono un livello di caching aggiuntivo e creano quindi ulteriori complicazioni dovuti al fatto che nei server log, tutte le richieste provenienti da un proxy server hanno lo stesso identificatore IP anche se, spesso, tali richieste coinvolgono utenti diversi che condividono lo stesso indirizzo IP.

Sono stati proposti vari approcci per cercare di superare queste difficoltà. Alcune ricerche suggeriscono di utilizzare i *cookies* per ottenere in modo automatico l'identità del visitatore o altri ancora di utilizzare le registrazioni degli utenti per generare un contesto dinamico in cui predire le richieste HTTP. In tutti questi approcci la collaborazione dell'utente è fondamentale; comunque, in generale, vengono sempre preferiti soluzioni non invasive.

Altri tipi di euristiche combinano l'utilizzo dell'indirizzo IP, gli agenti esterni e informazioni temporali per identificare gli utenti.

- *Session Identification.* Una volta che l'utente è stato individuato (utilizzando uno dei vari approcci elencati prima), lo stream di click per ogni singolo visitatore deve essere sezionato in *sessioni*, anche chiamate *transazioni*, che rappresentano l'insieme delle pagine visitate da un utente in una singola unità di lavoro.

Secondo la definizione proposta dal W3C, *una sessione è un gruppo di attività effettuate dall'utente dal momento in cui entra nel sito al momento in cui egli lascia il sito.*

Il problema dell'identificazione delle sessioni è causato dal fatto che, per log che riaprono un lungo periodo di tempo, è molto probabile che uno stesso utente visiti il sito Web più di una volta, e che quindi siano presenti molte sessioni di lavoro che lo riferiscono. Data la particolare struttura dei log, non è possibile conoscere quando un utente lascia il sito o si disconnette da un proxy server, e questa peculiarità rende quindi difficile l'identificazione di una precisa sessione.

Le euristiche utilizzate per l'identificazione delle transazioni sono di due tipi: *euristiche time-oriented* e *euristiche navigation-oriented*.

L'*euristiche time-oriented* utilizzano un time-out per risolvere il problema delle sessioni: se il tempo che intercorre tra la richiesta di due pagine consecutive eccede un certo limite, viene assunto che l'utente stia iniziando una nuova sessione. Altri ricercatori si sono concentrati sull'utilizzo di euristiche in cui il calcolo del timeout avviene in funzione del contesto della pagina e della natura dell'applicazione; un esempio proposto in questa direzione è quello di definire una sessione utente come una sequenza di pagine di navigazione seguita da una pagina di contesto, basandosi, quindi, sulla distinzione tra pagine di navigazione e di contesto dipendente dalla durata della visita della pagina stessa. L'*euristiche navigation-oriented* prendono in considerazione i link fra le varie pagine. L'idea si basa sulla considerazione che i visitatori, solitamente, piuttosto che inserire manualmente URL, preferiscono seguire gli hyperlink proposti dalla struttura del sito per raggiungere una certa pagina. Ogni sessione viene allora individuata con l'insieme dei link visitati dall'utente prima di un riferimento in backtracking, in sintesi, si inizia una nuova sessione appena viene riferita una pagina già visitata. Il vantaggio di questa soluzione è che non richiede nessun parametro in ingresso e non si basa su un particolare tipo di dato.

- *Path Completion.* Altro problema da risolvere è la completezza dei

camminini; infatti, data la presenza di cache e proxy server intermedi, gli accessi web potrebbero non essere registrati nei log locali e le sessioni potrebbero essere incomplete. Una soluzione, purtroppo impraticabile, per scoprire ogni accesso sarebbe quella di controllare la collezione dati al livello cliente; sono state proposte altre euristiche, alcune basate su pagine riferite altre utilizzano tecniche di apprendimento automatico.

- *Formatting.* Quando la fase di preprocessing è terminata, un'ulteriore fase di preparazione dei dati può essere utilizzata per formattare le sessioni utente per il particolare modello di Data Mining da estrarre o per lo specifico algoritmo da utilizzare.

Ad esempio, se si desidera applicare degli algoritmi di mining per l'estrazione di regole di associazione che non richiedono l'informazione temporale, si può allora filtrare l'attributo "tempo" da ogni singola sessione.

2.3.3 Pattern Discovery

Pattern Discovery utilizza metodi e algoritmi già sviluppati per diversi campi come data mining, statistica, apprendimento automatico e pattern recognition. Questi metodi precedentemente utilizzati devono però tener conto dei differenti tipi di astrazione di dati e della diversa conoscenza a priori disponibile per il Web Mining. Per esempio, nell'inferire regole di associazione, la nozione di transazione per il market basket analysis non prende in considerazione l'ordine con cui gli item sono selezionati; nel caso del Web Usage Mining, una sessione server è una sequenza ordinata di pagine richieste da un utente. Pertanto, dovuto alla difficoltà di identificare sessioni uniche, è richiesta una conoscenza a priori addizionale.

Proseguendo, una volta che utenti e sessioni sono stati identificati, esistono diversi tipi di analisi di mining che possono essere applicate in funzione delle necessità degli esperti.

La fase di pattern discovery include i seguenti task:

1. *Analisi dei cammini:* mette a disposizione una serie di grafi che rappresentano qualche relazione sulle pagine Web del sito. Il caso più semplice è quello in cui, ogni nodo del grafo rappresenta una pagina e gli archi rappresentano i link ipertestuali tra le pagine stesse; un'altra soluzione è quella in cui i nodi rappresentano delle similarità tra le pagine mentre gli archi danno il numero di utenti che transitano da una pagina ad

un'altra.

Questo tipo di analisi può aiutare a determinare le visite più frequenti in un sito web. Tipico esempio che descrive l'estrazione estratta con questo tipo di analisi è: il 70% dei clienti che accede a */company/product2* prosegue attraverso le pagine */company/new*, */company/products* e */company/product1*;

2. *Regole di associazione*: nel contesto del web mining esse possono essere utilizzate per correlare pagine, prodotti o contenuti che spesso sono riferiti congiuntamente e che non sono connessi direttamente attraverso degli hyperlink.

Vediamo un esempio: il 45% dei visitatori che accedono alla pagina VolleyBall, accedono anche alla pagina HandBall.

La presenza o l'assenza di una regola può ad esempio aiutare il designer nella ristrutturazione del sito, fornendogli indicazioni su come meglio organizzare lo spazio Web; altri ricercatori sull'utilizzo delle regole di associazione per offrire servizi di personalizzazione, esse possono anche servire come euristiche per il prefetching dei documenti allo scopo di ridurre l'attesa percepita da un utente a seguito di una richiesta etc.

3. *Pattern Sequenziali*: sono un modello di Data Mining che permette di estrarre tutte le sottosequenze da una sequenza di transazioni il cui supporto (frequenza nel database) è maggiore di una soglia minima specificata dall'utente. Ogni transazione viene correlata con un'informazione temporale (*time stamp*) che specifica l'ordine della transazione nella particolare sequenza associata ad un utente.

Nell'ambito del Web Mining, ogni visita di un utente (sessione) viene registrata su un determinato periodo di tempo. Questa tecnica di Pattern Discovery cerca di scoprire i pattern *inter session*, pattern tali che la presenza di un insieme di item è seguita da un altro item in un insieme ordinato rispetto al tempo di sessioni o episodi.

Vediamo un esempio: analizzando tale informazioni, il progettista del sito può determinare delle relazioni temporali tra i dati, come il 12% dei visitatori accede alla pagina sportiva e successivamente alla pagina questionario.

Tante sono le applicazioni dove poter utilizzare questo tipo di relazioni scoperte nei dati.

L'utilizzo di pattern sequenziali per lo sviluppo di modelli predittivi è

stato estensivamente studiato; il principale obiettivo di questi studi è stato il prefetching delle pagine Web (cioè la predizione del prossimo accesso ad una pagina dell'utente) per migliorare la performance del server e la latenza della rete. Altri tipi di analisi temporali che possono essere realizzati sui pattern sequenziali includono *trend analysis*⁴, *change point detection* o *similarity analysis*.

Approfondiremo meglio questo modello in seguito.

4. *Classificazione*: cerca di identificare schemi o insiemi di caratteristiche che definiscono il gruppo cui appartiene un dato elemento; essa parte dall'utilizzo di insiemi esistenti e già classificati a priori e porta alla definizione di alcune regolarità. Tali profili possono essere utilizzati successivamente per classificare nuovi dati che vengono aggiunti al database.

Nel contesto del Web Mining le tecniche di classificazione permettono di sviluppare un profilo degli utenti che accedono al sito che appartengono ad una particolare classe o categoria. Il risultato potrebbe essere la scoperta di relazioni del tipo: il 50% dei clienti che inserisce un ordine in */company/product2* ha un'età compresa tra i 20 e i 25 anni e vive in una località turistica.

5. *Clustering*: è una tecnica di analisi che raggruppa gli oggetti in base a misure di similitudini (aventi, cioè, caratteristiche simili); l'obiettivo è quello di massimizzare le similitudini tra oggetti di uno stesso cluster e minimizzare quelle tra elementi appartenenti a cluster differenti.

Nel dominio Web Usage Mining ci sono tre tipi interessanti di cluster da analizzare: *clustering di pagine*, *clustering di utenti* e *clustering di sessioni*. Essi, rispettivamente, cercano di inferire: gruppi di pagine con un contesto correlato o che sono accedute insieme da molti utenti; gruppi di utenti con abitudini di visita simili in termini di pagine accedute o con gli stessi accessi nel tempo; sessioni di visita in base alle similitudini individuate nei cammini percorsi dagli utenti o in base al tempo speso in ogni pagina.

Molti ricercatori si sono indirizzati sull'uso di tecniche di clustering nel contesto del Web Usage Mining.

⁴Metodologia di previsione secondo cui le stime future derivano da analisi statistiche su serie storiche. Attraverso tale metodologia, ad esempio, si può stimare il livello delle vendite future sulla base degli andamenti passati delle stesse.

Queste tecniche possono essere utilizzate a se stante oppure combinate in un approccio unico.

Il risultato di questa fase è comunque una serie di modelli, un insieme di regole di associazione, un albero di classificazione, un insieme di cluster, che a questo punto possono essere utilizzati per diversi aspetti del Web Usage Mining.

2.3.4 Pattern Analysis

Il Pattern Analysis rappresenta l'ultimo passo del processo del Web Usage Mining.

Una volta scoperti i pattern è necessario utilizzare degli strumenti e meccanismi in grado di aiutare l'analista a meglio comprenderli. Quindi in aggiunta allo sviluppo di tecniche di mining su Web, c'è la necessità dell'utilizzo di metodologie e strumenti che mettono in pratica un'analisi dei pattern individuati.

In particolare possiamo individuare due obiettivi in questa fase:

1. filtrare le regole o i pattern ritenuti non interessanti dall'insieme individuato nella fase precedente,
2. interpretare la conoscenza estratta.

Per quanto riguarda il primo task, si noti che esso dipende ovviamente dal tipo di applicazione di web mining che è stata fatta. Ad esempio, può essere necessario filtrare alcune regole di associazione o selezionare solo i gruppi di interesse dopo il processo di clustering.

Per quanto riguarda il secondo, numerosi strumenti di visualizzazione sono stati implementati con successo per aiutare le persone a comprendere i vari tipi di fenomeni, sia reali che astratti. Diventa quindi naturale l'applicazione di queste tecnologie al contesto web.

La più comune forma di pattern analysis consiste in meccanismi di query come SQL; un altro metodo consiste nel caricare i dati in un *data cube* allo scopo di applicarvi delle operazioni OLAP⁵ oppure esistono varie tecniche di visualizzazione.

⁵On-Line Analytical Preprocessing; permette di eseguire complesse query usando simultaneamente molteplici attributi. Tale capacità è molto utile per gestire la grande quantità di dati provenienti dal Web.

2.4 Sistemi per il Web Usage Mining

Abbiamo visto in cosa consiste questo particolare tipo di Mining, cerchiamo adesso di capire quali sono gli scopi che portano a elaborare i dati di Web Usage.

Fino ad ora, i pattern d'uso estratti dai dati Web sono stati applicati ad una ampia gamma di applicazioni. Alcuni ricercatori sono riusciti ad includere i sistemi di Web Usage Mining in cinque categorie: *personalizzazione*, *system improvement*, *site modification*, *business intelligence* e *usage characterization* (figura 2.3).

Analizziamo ciascuno di queste categorie.

2.4.1 Personalization

Abbiamo accennato prima all'enorme sviluppo di internet e di come industrie e aziende ne abbiano compreso le enormi potenzialità tanto da rendere più che massiccia la loro presenza in rete; questa presenza da parte di più soggetti economici ha portato e continua a portare ad una loro competizione per acquisire sempre nuovi clienti e per mantenere quelli già esistenti. Da questa necessità, il fornire un servizio standard indipendentemente dalla diversità dei bisogni dei clienti è stato soppiantato da una sempre maggiore attenzione alle preferenze ed ai comportamenti dei visitatori: prestare loro attenzione può dare molti vantaggi sia in termini di efficienza che di efficacia di un sito. Uno dei principali obiettivi del Web Usage Mining è quello di personalizzare il contesto di un sito per ogni singolo visitatore.

La personalizzazione viene definita da Mulvenna, Anand e Buchner come *la fornitura di informazioni, prodotti e servizi su misura*; essa è una vasta area che comprende recommender system⁶, customizzazione⁷ e siti web adattivi⁸. Personalizzare un'esperienza web per un utente è una parte molto importante si pensi ad esempio al *individualized marketing* per e-commerce. Poter fare delle "raccomandazioni" dinamiche ad un utente web, basate sul suo profilo

⁶ "Sistemi raccomandanti" sono i sistemi in grado di supportare chi debba valutare alternative o prendere decisioni, fornendogli informazioni su comportamenti analoghi tenuti da altri operatori.

⁷ Combinazione di informazioni individuali e design flessibile di prodotti in relazione alle specifiche del cliente.

⁸ Siti web che migliorano automaticamente la loro organizzazione e presentazione imparando dai pattern di accesso dei visitatori.

in aggiunta al comportamento d'uso è di notevole attrazione per molte applicazioni, per esempio *cross-sale* e *up-sales* in e-commerce. Web usage mining è un eccellente approccio per raggiungere questo obiettivo.

Fino ad oggi la maggior parte dei sistemi per la personalizzazione su web ha usato tre categorie di metodi: collaborative filtering, content-based filtering, usage-based filtering. I sistemi per *collaborative filtering* usano informazione esplicita sulle preferenze degli utenti ed un correlation engine per predire quali informazioni o prodotti corrispondono alle preferenze degli utenti in base al comportamento avuto in passato da clienti simili. L'approccio dei sistemi per *content based filtering* si basa sulla similitudine tra il contenuto dei documenti ed un profilo personale ottenuto esplicitamente o implicitamente dagli utenti. Gli input utilizzati per questa tecnica, oltre ai server log, riguardano informazioni di contesto sul sito (es. le descrizioni dei prodotti venduti), ma ignorano i dati relativi ad altri utenti. Infine, l'approccio dei sistemi *usage-based filtering* non utilizza sorgenti dati di tipo collaborativo o informazione di contesto sul contenuto delle pagine, ma basa la personalizzazione esclusivamente sulla storia della visita degli utenti, eventualmente supportata da informazioni opzionali ottenute ad esempio da un processo di registrazione. La nuova generazione di tool per la personalizzazione è cerca di incorporare tecniche di web mining.

2.4.2 System Improvement

L'enorme successo del World Wide Web è stato pagato in termini di un incremento del tempo di attesa durante il caricamento di un documento.

Per superare questo tipo di problemi, da qualche anno, sono state studiate strategie in grado di ridurre la latenza delle richieste dell'utente. Lo scopo di queste metodologie è di predire le richieste future e caricare le pagine prima che l'utente ne faccia reale richiesta. Fanno parte di questa categoria anche le politiche adottate per la sicurezza nel commercio elettronico riguardanti lo sviluppo di pattern utili per individuare intrusioni o frodi informatiche.

Le metodologie adottate in questo campo sono basate sull'introduzione di proxy cache e su tecniche di pre-fetching. Alcune ricerche si sono concentrate sulla costruzione di modelli per la predizione della località, spaziale e/o temporale, attraverso l'analisi delle pagine Web richieste da un particolare gruppo di utenti sullo stesso proxy server; altri studi hanno prodotto algoritmi

per la creazione profili dei cammini attraverso l'analisi dei server log; altri ancora hanno presentato una metodologia di pre-fetching basata su tecniche per l'estrazione di regole di associazione dalle pagine visitate dall'utente.

2.4.3 Site Modification

La capacità attrattiva di un sito è fondamentale per numerose applicazioni nel campo del commercio elettronico; per capacità attrattiva si intende non solo l'apparenza di una pagina web ma anche la qualità della struttura di interconnessione tra le pagine per assistere al meglio i visitatori nel soddisfare i loro bisogni siano essi informazioni o prodotti.

Nella ristrutturazione di un sito web è molto utile ricorrere a soluzioni basate su Web Usage Mining; questo tipo di studi fornisce al designer informazioni di base sul comportamento dei visitatori. Per esempio, alcuni ricercatori hanno indirizzato i loro studi sulla modifica della struttura di un sito basata sulla scoperta automatica di pattern dai server log oppure altri hanno implementato una tecnica per migliorare l'efficienza di un sito web basandosi su un'analisi comparata del comportamento di navigazione dei clienti.

2.4.4 Business Intelligence

Una volta sviluppati i vari siti per i vari soggetti economici, l'informazione su come i clienti utilizzano questi siti è critica.

Il *Business Intelligence* rappresenta un insieme di metodi e tecnologie che permette alle imprese di raccogliere, immagazzinare, analizzare, garantire accesso ai dati al fine di aiutare gli utenti aziendali a prendere decisioni di business. Generalmente le applicazioni sono alimentate da basi di dati esistenti e si presentano a chi voglia servirsene sotto forma grafica per essere di facile interpretazione. Possono fornire ai responsabili istogrammi e grafici aggiornati in tempo reale riguardanti andamento degli ordini, delle vendite e delle consegne in relazione ai dati di budget.

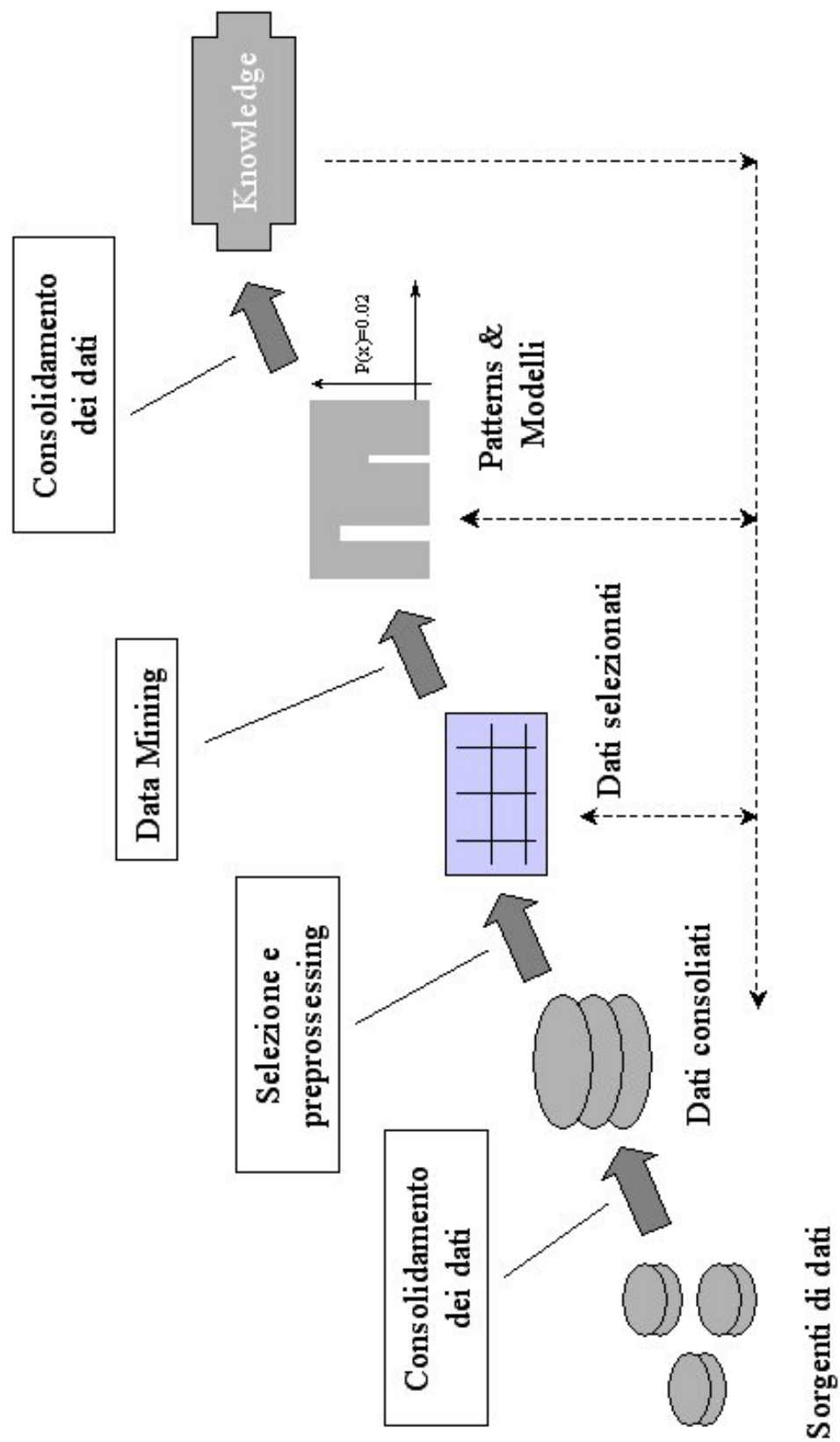
Il Business Intelligence analizza l'efficacia di un sito da due prospettive diverse: marketing e merchandising. Il *marketing* si riferisce allo studio delle tecniche per acquisire e mantenere utenti on-line, ad esempio attraverso compagnie di e-mail o con l'utilizzo di banner pubblicitari ad hoc. Il *merchandising* consiste nello sviluppo di strategie adeguate per l'acquisto dei prodotti e per renderli disponibili nel posto e nel tempo giusti, con il prezzo e la quantità opportuni.

I *clickstream*, cioè le sequenze degli accessi degli utenti, relative alle attività on-line contengono informazioni utili per capire l'efficacia delle strategie di marketing e del merchandising in quanto ci indicano i clienti che accedono al magazzino, i prodotti che essi esaminano maggiormente, e infine i prodotti che acquistano.

2.4.5 Usage Characterization

Questo settore cerca di caratterizzare il comportamento di navigazione dei visitatori di un sito web.

Il risultato di alcuni studi condotti in questo campo sono ad esempio informazioni relative all'interazione degli utenti con l'interfaccia di un particolare browser oppure vengono proposti modelli per predire la distribuzione probabilistica delle pagine che un utente potrebbe voler visitare in un sito.

Figura 2.1: Il processo di *Knowledge Discovery in Database*

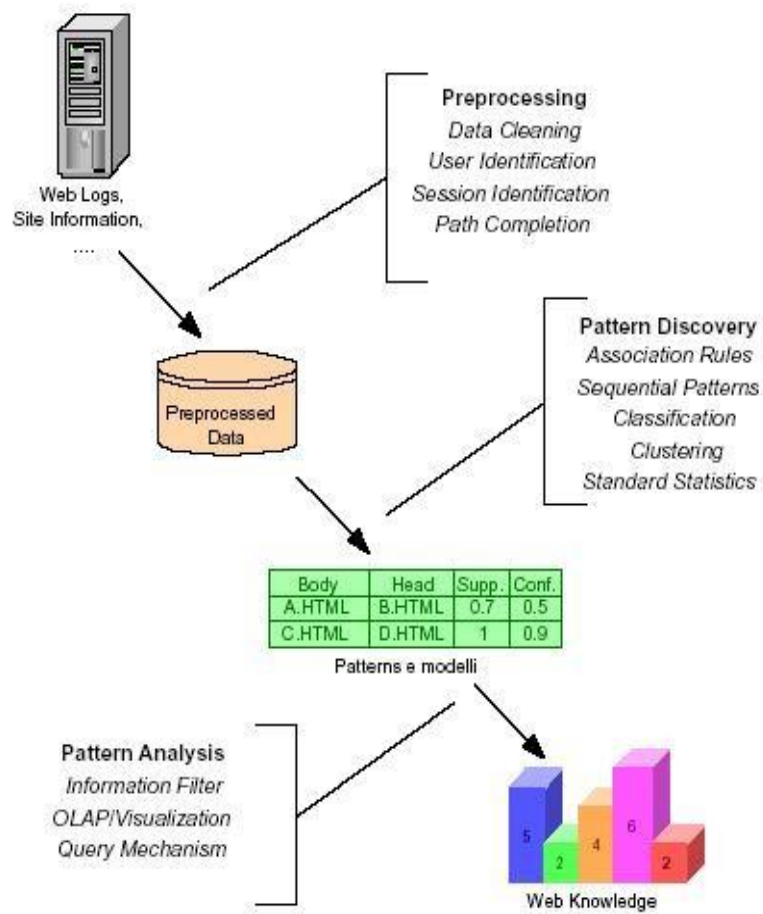


Figura 2.2: Il processo di Web Usage Mining

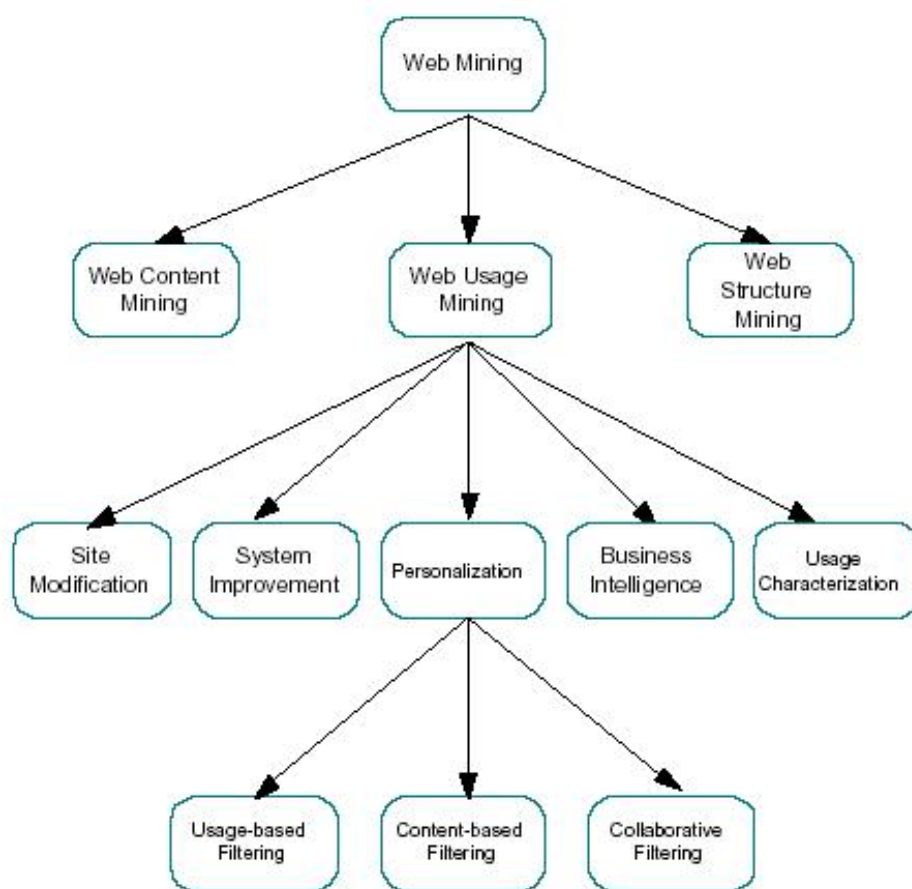


Figura 2.3: Aree applicative del Web Usage Mining

Capitolo 3

Sequential Pattern

3.1 Sequential Pattern: Stato dell'Arte

Molti domini scientifici e commerciali hanno avuto una enorme crescita dei dati negli ultimi tempi. E' diventato sia utile che necessario processare questi dati per riuscire ad estrarne della conoscenza nascosta e interessante. I dati raccolti da esperimenti scientifici, o monitoraggi di sistemi fisici come reti di telecomunicazioni, o da transazioni di un supermercato, o da studi in campo medico, o da Web log (Web surfing) etc hanno una intrinseca natura sequenziale. Quindi sono tante e importanti le aree di applicazione dove i dati che devono essere analizzati sono in qualche modo legati al fattore tempo. Tali dati, astrattamente, possono essere visti come una sequenza di valori o meglio di eventi, dove ogni evento ha associato un tempo di occorrenza. Tra le differenti tecniche di *data mining*, **sequential pattern mining** è quella che può ottenere l'informazione più significativa considerando il fattore tempo. *Sequential pattern mining* cerca di trovare le relazioni esistenti tra le occorrenze di eventi sequenziali. La scoperta di relazioni sequenziali o pattern presenti in tali dati è utile per vari scopi come descrizione dei dati, predizione di eventi, identificazione di regole sequenziali che caratterizzano differenti classi di dati.

Passiamo adesso ad esaminare in dettaglio qual è il contesto in cui si inseriscono i sequential pattern.

In generale, nella letteratura trovare pattern da dati sequenziali può essere classificato in tre categorie [31]. La prima categoria consiste nel trovare pattern simili, dove un pattern è simile ad un altro se la distanza o cor-

relazione tra loro è all'interno di una soglia definita dall'utente. La seconda, consiste nel trovare pattern periodici, cioè si è alla ricerca di pattern ciclici in database "time-stamped". Infine la terza, consiste nel trovare pattern frequenti, cioè pattern la cui frequenza nel database supera una data soglia. Tra queste tre categorie, trovare pattern frequenti è possibilmente la più popolare ed è stata intensivamente studiata grazie alle sue numerose applicazioni. Trovare pattern frequenti può essere suddiviso in:

- trovare pattern sequenziali frequenti (*sequential frequent pattern*)
- trovare pattern non sequenziali frequenti (*non-sequential frequent pattern*)

I primi sono sottosequenze lineari che occorrono frequentemente nel database, mentre i secondi sono sottosequenze non lineari come grafi di ordine parziale, strutture ad albero o strutture a grafo. Noi siamo interessati ai primi.

Si noti che quando parliamo di database ci riferiamo a database che consistono di sequenze di eventi ordinati con o senza nozioni concrete di tempo.

Fondamentalmente, per trovare pattern sequenziali i primi ricercatori usavano metodi simili a quelli dell'estrazione di regole di associazione. *Sequence discovery* può essenzialmente essere pensata come *association discovery* su un database "temporale". Facciamo un esempio per capire meglio. Nel caso di analisi di mercato (*basket analysis*), supponendo di voler studiare il contenuto di un database contenente le transazioni di acquisto dei clienti di un supermercato, il problema di trovare quali prodotti (che chiameremo *items* e che in seguito vedremo meglio) vengono comprati insieme è legato al trovare *intra-transaction patterns*, mentre il problema di trovare sequential pattern è legato a *inter-transaction patterns*: nel primo problema il modello consiste di un insieme non ordinato di items, mentre nel secondo caso un modello è una lista ordinata di insiemi di items. I pattern vengono scoperti identificando nel database quelle sottosequenze con una frequenza non inferiore ad una data soglia. Così, i modelli trovati indicano quali sono le sottosequenze che occorrono frequentemente nei nostri dati su cui indirizzare la nostra attenzione. Per esempio, dal database di transazioni di un negozio di libri possiamo trovare dei pattern sequenziali frequenti di acquisto del tipo: 80% dei clienti che comprano il libro "Database Management" tipicamente

comprano anche il libro “Data Warehouse” e dopo comprano il libro “Web Information System”. Un altro esempio, un utente può visitare in ordine il sito A , dopo il sito B e infine il sito C ; se molti utenti hanno lo stesso comportamento, allora la sottosequenza ABC forma un pattern significativo.

I pattern sequenziali trovati dalle ricerche e dagli studi precedenti possono essere classificati in due principali categorie ([31]):

- pattern sequenziali continui. Tutti gli elementi del modello appaiono necessariamente in ordine consecutivo anche nel database di sequenze.
- pattern sequenziali discontinui. Gli elementi del modello appaiono in ordine non consecutivo nel database (anche se non necessariamente).

Con pattern ibridi ci riferiremo a quei pattern formati da sottopattern sia continui che discontinui dove sia possibile risalire a ciascuno di essi. Vedremo meglio in seguito la necessità di queste distinzioni.

3.1.1 Argomenti correlati

Il sequential pattern mining problem è stato per prima introdotto da Agrawal e Srikant nel '95 [1], in questo articolo furono presentati tre algoritmi; successivamente nel '96, gli stessi autori proposero in un lavoro successivo ([2]) un altro algoritmo che migliorava in performance quelli proposti precedentemente.

Quasi contemporaneamente, Mannila e al. in [19] presentavano il problema di trovare *episodi frequenti* (che verificavano una determinata soglia di frequenza) in una lunga sequenza di eventi (solamente). Un episodio è definito come un insieme di eventi che occorrono con un ordine definito parzialmente e all'interno di un determinato limite di tempo (*time-window*). Precisamente ci sono tre forme di episodi: episodi seriali (è definito un ordine totale sugli eventi all'interno del *time-window* specificato), episodi paralleli (è definito un ordine parziale sugli eventi all'interno del *time-window* specificato), episodi compositi-misti (formati dall'uno e dagli altro tipo). In un lavoro successivo, essi generalizzarono il loro lavoro per permettere di esprimere condizioni arbitrarie unarie su coppie di attributi di eventi. Confrontando questo tipo di studi con le caratteristiche dei pattern sequenziali di nostro interesse, possiamo notare, innanzitutto, che i pattern sequenziali sono in relazione solo

con gli episodi seriali, c'è però una differenza, noi infatti siamo interessati ad episodi seriali frequenti che superino una certa soglia all'interno di molte sequenze e non in una lunga sequenza di eventi (come quelli studiati da Mannila e al), inoltre noi vorremmo dei pattern frequenti senza nessuna restrizione imposta. Pertanto la via più seguita successivamente anche dagli altri ricercatori nel campo del sequential pattern mining è stata quella di Agrawal e Srikant; infatti gli algoritmi da essi proposti sono stati la base di confronto con gli algoritmi successivamente proposti.

Una volta concettualmente introdotti i pattern sequenziali e il problema della loro estrazione (*sequential pattern mining problem*), passiamo adesso a descriverli in modo dettagliato e in maniera più formale.

3.2 Sequential Pattern Mining Problem

Per poter definire in modo preciso i sequential pattern è necessario prima definire altri elementi.

Utilizzando come riferimento il database di sequenze della tabella 3.1, supponiamo che il database riguardi le transazioni di acquisto in una libreria e che quindi le lettere in tale database siano i nomi dei libri acquistati da un generico cliente.

id_sequenza	sequenze
10	$\langle a(bc)e \rangle$
20	$\langle e(ab)(bc)dd \rangle$
30	$\langle c(aef)(abc)dd \rangle$
40	$\langle addcb \rangle$

Tabella 3.1: Database di sequenze

Definizione 1 Sia $I = \{i_1, i_2, \dots, i_n\}$ l'insieme di n distinti elementi chiamati **items**, con $n \in \mathbb{N}$.

Nel nostro esempio, l'insieme delle lettere dell'alfabeto, cioè l'insieme di tutti i libri della libreria, sono gli item del dominio che stiamo analizzando.

Definizione 2 Un *itemset* è un sottoinsieme non vuoto di *items*.

La prima riga e seconda colonna del database individuano una cella contenente tre itemset, chiamati anche transazioni, e sono rispettivamente: $(a), (bc)$ e (e) . Per semplicità le parentesi sono omesse quando un itemset è formato da un unico elemento. Nell'esempio, gli elementi di una transazione rappresentano i libri comprati nella stessa occasione.

Definizione 3 Una *sequenza* è una lista ordinata di itemset:

una sequenza s è denotata da $\langle s_1, s_2, s_3, \dots, s_l \rangle$ dove s_j è un itemset, $s_j \subseteq I$ per $1 \leq j \leq l$, $l \in N$, $j \in N$, s_j è anche chiamata un elemento della sequenza e denotata come $s_j = (x_1, x_2, x_3, \dots, x_m)$, dove $x_k \in I$ per $1 \leq k \leq m$, $k \in N$, $m \in N$.

Gli elementi della seconda colonna del database rappresentano delle sequenze; ogni sequenza è costituita infatti da transazioni che sono ordinate rispetto al tempo. Per esempio la prima sequenza descrive le transazioni d'acquisto di un cliente che in tre occasioni diverse ha comprato in ordine di tempo la prima volta il libro a , la seconda due libri, b e c , la terza il libro e .

Definizione 4 La *lunghezza* di una sequenza è il numero di items in essa, una sequenza con lunghezza 1 è chiamata 1-sequence. Data la sequenza s la sua lunghezza è indicata anche con $|s|$.

Dall'esempio, la prima sequenza è lunga 4, la seconda è lunga 7, la terza è lunga 9 e infine l'ultima è lunga 5; rispettivamente esse hanno 4, 7, 9 e 5 item. Si noti che la lunghezza di una sequenza è indipendente dal numero di itemset che la compongono.

Definizione 5 Una sequenza $\alpha = \langle \alpha_1, \alpha_2, \alpha_3, \dots, \alpha_n \rangle$ è chiamata una *sottosequenza* di un'altra sequenza $\beta = \langle \beta_1, \beta_2, \beta_3, \dots, \beta_m \rangle$ e β una *supersequenza* di α , denotato con $\alpha \sqsubseteq \beta$, se esistono degli interi $1 \leq j_1 < j_2 < \dots < j_n \leq m$ tale che $\alpha_1 \subseteq \beta_{j_1}, \alpha_2 \subseteq \beta_{j_2}, \dots, \alpha_n \subseteq \beta_{j_n}$, con $n \in N$, $m \in N$.

La sequenza $\langle (ab) \rangle$ è una sottosequenza di $\langle e(ab)(bc)dd \rangle$ e di $\langle c(aef)(abc)dd \rangle$, a loro volta entrambe queste sequenze sono supersequenze della prima.

Definizione 6 Un **database di sequenze** S è un insieme di tuple $\langle sid, s \rangle$, dove sid è un identificatore di sequenza (noto anche con *sequence_id*) ed s è una sequenza.

Un esempio di database di sequenze è proprio il database della tabella 3.1.

Definizione 7 Si dice che una tupla $\langle sid, s \rangle$ **contiene** una sequenza α , se α è una sottosequenza di s , $\alpha \sqsubseteq s$.

Per esempio la tupla $\langle 10, \langle a(bc)e \rangle \rangle$ contiene la sequenza $\langle bc \rangle$, infatti $\langle bc \rangle$ è una sottosequenza di $\langle a(bc)e \rangle$.

Definizione 8 Una sequenza α è chiamata **sequential pattern** in un sequence database S se la sequenza è una sottosequenza di almeno una tupla nel database:

$$\exists \langle sid, s \rangle \mid (\langle sid, s \rangle \in S) \wedge (\alpha \sqsubseteq s). \quad (3.1)$$

Definizione 9 Il **supporto** di una sequenza α in un database di sequenze S è il numero di tuple nel database contenenti α ,

$$support_s(\alpha) = |\{ \langle sid, s \rangle \mid (\langle sid, s \rangle \in S) \wedge (\alpha \sqsubseteq s) \}| \quad (3.2)$$

Il supporto della sequenza $\langle bc \rangle$ è 3, sono tre infatti le sequenze del database che la contengono

Definizione 10 Si definisce anche il **supporto con percentuale** di una sequenza il rapporto fra il suo supporto e il numero di sequenze nel database:

$$\sigma_S(\alpha) = \frac{support_S(\alpha)}{|S|} \quad (3.3)$$

Riferendosi all'esempio precedente, il supporto percentuale di $\langle bc \rangle$ è 75%.

Definizione 11 Dato un intero positivo ξ come soglia di supporto, una sequenza α è chiamata un **frequent sequential pattern** in un sequence database S se la sequenza è contenuta da almeno ξ tuple nel database:

$$support_s(\alpha) \geq \xi. \quad (3.4)$$

Fissando una soglia di supporto pari al 75%, alcuni pattern sequenziali del database della tabella 3.1 sono: $a(bc)$, e , dd

Definizione 12 *Un pattern sequenziale con lunghezza l è chiamato l -pattern.*

Definizione 13 ***Definizione del problema:** dato un database di sequenze e una soglia di min_support , il problema di Sequential pattern mining è trovare l'insieme completo dei modelli sequenziali frequenti nel database.*

Data input = database S formato da sequenze del tipo $\langle \text{sid}, s \rangle$, supporto minimo ξ

Data output = tutti i sequential pattern il cui supporto è superiore o uguale a ξ , chiamati ξ -pattern: $\{\alpha | \text{support}_s(\alpha) \geq \xi\}$.

3.3 Algoritmi per inferire SP

Il *Sequential Pattern Mining* è stato intensamente studiato durante gli anni recenti, così esistono una grande varietà di algoritmi per la loro estrazione. Quando parliamo di tali algoritmi possiamo parlare di algoritmi generali e di base per il *sequential pattern mining* o estensioni di questi algoritmi per speciali scopi. Approfondiremo in particolare i primi e discuteremo in generale dei secondi di cui molti sono ancora in fase di sviluppo.

3.3.1 Algoritmi generali

La gran parte dei primi algoritmi e quelli di base sono basati sulla famosa proprietà di *Apriori* proposta per l'estrazione delle regole di associazione che afferma:

Definizione 14 *Qualsiasi sotto pattern di un modello frequente deve essere frequente.*

La proprietà è basata sull'osservazione che se un pattern I non soddisfa la soglia di minimo supporto, allora I non è frequente e qualsiasi *super-pattern* di I sarà anch'esso non frequente.

Basate su queste euristiche una serie di algoritmi *Apriori-like* sono stati proposti: *AprioriAll*, *AprioriSome*, *DinamicSome*, *GSP*. Più tardi un'altra serie

di algoritmi sono stati proposti come: *FreeSpan*, *PrefixSpan*, *PrefixGrowth* basati su proiezione dei dati; *Spade*, *cSpade* algoritmi che lavorano su un particolare formato dei dati; *SPIRIT*, integra delle restrizioni usando espressioni regolari. Fra gli algoritmi più recentemente proposti abbiamo *SLPminer*, utilizza un supporto non costante, *GFP1* e *GFP2* calcolano i modelli ibridi. In generale, dato il database di sequenze contenente i dati in esame e il supporto che deve essere verificato, l'algoritmo prende in pasto i dati e restituisce i pattern sequenziali trovati lavorando in maniera iterativa o ricorsiva. Dall'analisi di questi algoritmi abbiamo realizzato una loro classificazione sulla base delle caratteristiche per noi importanti.

Algoritmi con o senza restrizioni

Innanzitutto è necessario sottolineare che il compito di estrarre tutte le sequenze frequenti in grandi database è un compito abbastanza difficoltoso: lo spazio di ricerca è estremamente grande. Per esempio, con m item ci sono potenzialmente $O(m^k)$ sequenze frequenti di lunghezza k . Con milioni di oggetti nel database il problema di minimizzazione di I/O diventa importante, inoltre la gran parte di algoritmi sono iterativi in natura e richiedono tante intere scansioni del database quanto più lunghe sono le sequenze frequenti; chiaramente un processo molto dispendioso in quanto il costo computazionale è molto elevato. A questo bisogna aggiungere un altro problema, una volta trovati tutti i pattern sequenziali magari solo una parte di essi sono di vero interesse. Infatti, nel caso di *selective users*, cioè utenti alla ricerca di un pattern specifico piuttosto di pattern frequenti, come nel caso di studi di sequenze di natura genetica (*gene sequence studies*) il volume di risultati potenzialmente inutili è opprimente. In pratica stiamo parlando di efficienza e efficacia, e sotto questa chiave è di notevole importanza poter integrare negli algoritmi di mining delle restrizioni, in modo da poter ulteriormente restringere lo spazio di ricerca di nostro interesse. Il mining basato su restrizione può superare entrambe le difficoltà dal momento che le restrizioni generalmente rappresentano l'interesse, il focus dell'utente, e che esse confinano i pattern che devono essere trovati ad un particolare insieme di condizioni. In particolare, le restrizioni possono essere inserite (messe dentro) nel processo di mining, e questo permette possibilmente di acquisire maggiore efficienza. In alcuni campi di applicazione di data mining le restrizioni risultano essere

di notevole aiuto. Questo motiva questo tipo di studio. Passiamo ad una definizione un pò più formale.

Definizione 15 *Sia una restrizione C per un sequential pattern α una funzione booleana $C(\alpha)$. Il sequential pattern mining problem basato su restrizioni consiste nel trovare l'insieme completo dei pattern sequenziali denotati come $SAT(C)$ soddisfacenti una data restrizione C .*

Vediamo un esempio. Per caratterizzare una nuova malattia dei ricercatori possono voler trovare dei pattern sequenziali su dati costituiti da sintomi, tali come:

“trovare patter con restrizioni del tipo tosse da 2 a 7 giorni seguita da febbre in media da $37.5C$ a $39C$ con una temperatura media di $38 \pm 0.2C$, e tutti i sintomi appaiono in un lasso di tempo di 2 settimane.

Un pattern potrebbe essere “5 giorni di tosse più 4 giorni di febbre con forte mal di testa”. Questa query di mining contiene un pò di restrizioni che coinvolgono sequenze contenenti alcune costanti, funzioni di media etc. Comunque non è ancora chiaro come incorporare tutte le restrizioni nel processo di mining e nonostante sono essenziali per molte applicazioni non ci sono ancora studi sistematici su sequential pattern mining basati su restrizioni.

Le restrizioni possono essere di vario tipo, vediamo alcune:

- *restrizione di item.* Specifica quali sono il particolare item o gruppi di item che dovrebbero essere o non essere presenti nei pattern.
- *restrizione di lunghezza.* Specifica la richiesta di lunghezza del pattern, dove la lunghezza può essere sia il numero di occorrenze di item o il numero di transazioni. Questo tipo di restrizione può anche essere specificato come il numero di item distinti o anche il numero massimale di item per transazione.
- *restrizione aggregata.* E' la restrizione su un aggregato di item in un pattern, dove la funzione aggregata può essere *sum*, *avg*, *max*, *min*...
- *restrizione di durata (time-windows).* Questo tipo di restrizione è definita solamente nei database dove ogni transazione in ogni sequenza ha associato un *time-stamp*. Essa richiede che il pattern appaia frequentemente nel database di sequenze tale che la differenza di *time-stamp* tra la prima e l'ultima transazione nel pattern debba essere più lunga o più corta di un dato periodo.

- *intervallo max/min fra gli elementi di una sequenza.* Anche questo tipo di restrizione è definita solamente nei database dove ogni transazione in ogni sequenza ha associato un *time-stamp*. Essa richiede che il pattern appaia frequentemente nel database di sequenze tale che la differenza di *time-stamp* tra ogni due transazioni adiacenti debba essere più lunga o più corta di un dato intervallo.
- *restrizione di espressione regolare C_{RE} .* E' una restrizione specificata come un'espressione regolare su un insieme di item che usa un insieme stabilito di operatori di espressioni regolari, come la disgiunzione e la chiusura di Kleene. Un sequential pattern soddisfa C_{RE} se e solo se il pattern è accettato dal suo equivalente automa a stati finiti deterministico (*equivalent deterministic finite automata*).
-

Si noti che esiste anche la restrizione di supporto, ma essendo essa così importante viene trattata a se stante.

Algoritmi con supporto fisso o non

Negli ultimi anni sono stati sviluppati una varietà di algoritmi per trovare pattern sequenziali frequenti. La caratteristica comune di questi algoritmi è che essi usano una restrizione di supporto costante per controllare la complessità esponenziale del problema; in sintesi una potenziale sottosequenza candidata α diventa frequente se verifica la restrizione di supporto ϵ , cioè se $\text{supp}_S(\alpha) \geq \epsilon$.

In generale, i pattern che contengono solo pochi item tenderanno ad essere interessanti se essi hanno un alto supporto, a differenza invece dei pattern lunghi che possono ancora essere interessanti anche se il loro supporto è relativamente piccolo. Pertanto usando questi algoritmi con supporto costante, in un certo senso, perdiamo dell'informazione che potrebbe essere importante. Sfortunatamente, se usiamo gli algoritmi per la scoperta di sequential pattern basati su un supporto costante per trovare alcuni dei più lunghi ma non frequenti pattern, essi finiranno per generare un numero esponenzialmente grande di pattern diciamo "breve", cioè di pattern formati da pochi item. Idealmente noi desidereremmo avere un algoritmo che trovi tutti i pattern frequenti il cui supporto diminuisce come funzione della loro lunghezza.

Dovuto a questa motivazione, recentemente molti studi si sono e continuano a spostarsi su questa strada.

Proprio da poco è stato introdotto un nuovo algoritmo, il cui nome è *SLP-Miner*, in grado di trovare tutti i sequential pattern che soddisfano una restrizione di supporto decrescente, vale a dire un algoritmo che sfrutti l'idea di trovare tutti i sequential pattern il cui supporto diminuisce come una funzione della lunghezza dei pattern.

L'idea alla base di questo algoritmo è quella di sfruttare un'altra idea, anch'essa introdotta da poco. Vediamo in cosa consiste.

Recentemente è stato introdotto il problema di trovare itemset frequenti il cui supporto soddisfa una funzione non crescente della loro lunghezza, che consiste quindi nel trovare lunghi itemset con basso supporto e corti itemset con alto supporto. Una restrizione di supporto decrescente con la lunghezza è data come una funzione della lunghezza dell'itemset $f(l_a) \geq f(l_b)$ per qualsiasi l_a, l_b che soddisfano $l_a < l_b$ (vedi figura 3.1). Un itemset è frequente solo se il suo supporto è più grande o uguale al valore di minimo supporto determinato dalla lunghezza dell'itemset.

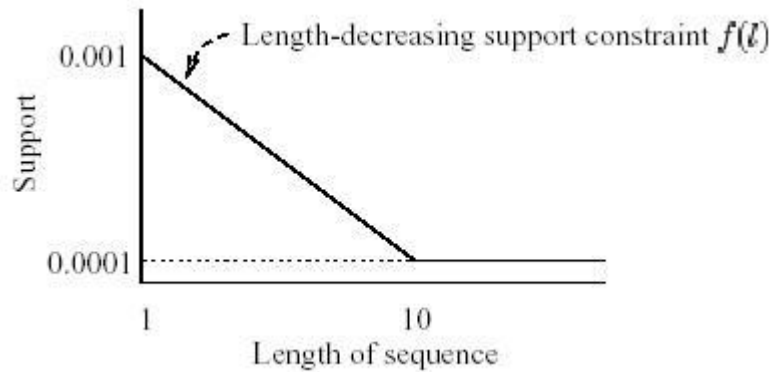


Figura 3.1: Tipica restrizione di supporto decrescente con la lunghezza

L'idea è naturalmente quella di estendere questa restrizione al modello di sequenza usando la lunghezza di una sequenza anziché la lunghezza di un itemset come segue:

Definizione 16 (restrizione di supporto decrescente rispetto alla lunghezza) *Dato un database di sequenze S , una funzione $f(l)$ della lunghezza l di una sequenza che soddisfa $1 \geq f(l) \geq f(l+1) \geq 0$ per qualsiasi intero l , una*

sequenza α è frequente se e solo se $\sigma_S(\alpha) \geq f(|\alpha|)$ (dove σ è il supporto della sequenza).

Data $f(l)$ possiamo definire anche la sua inversa.

Definizione 17 (Funzione inversa della restrizione di supporto decrescente rispetto alla lunghezza) *Data una funzione di supporto decrescente di lunghezza $f(l)$, la sua funzione inversa è definita come $f^{-1}(\sigma) = \min(l | f(l) \leq \sigma)$ per $0 \leq \sigma \leq 1$.*

Formalmente il problema di trovare questi tipi di pattern è definito come segue:

Definizione 18 (Sequential pattern mining con supporto decrescente rispetto alla lunghezza) *Dato un database sequenziale S e una restrizione di supporto decrescente rispetto alla lunghezza $f(l)$, trova tutti i sequential pattern tali che $\sigma_S(\alpha) \geq f(|\alpha|)$.*

E' importante notare che in questo caso la famosa proprietà di *Apriori*, alla base della gran parte degli algoritmi proposti fino ad ora, in questo caso non è valida: sotto l'ipotesi di un supporto variabile, una sequenza può essere frequente anche se le sue sottosequenze non sono frequenti dal momento che il valore di supporto minimo decresce con l'aumentare della lunghezza della sequenza.

Viene così identificata una proprietà chiave, già utilizzata per trovare itemset frequenti rispetto ad un supporto decrescente con la lunghezza, riguardante le sequenze il cui supporto diminuisce come una funzione della loro lunghezza, proprietà chiamata "*smallest valid extension*" o "*SVE*" in breve; essa rende possibile la potatura (*pruning*) di grandi porzioni di database di sequenze che sono irrilevanti per trovare pattern frequenti e permette di acquisire una maggiore performance sviluppando dei metodi di pruning basati su essa molto efficaci. Ecco in cosa consiste.

Dato un sequential database S e una particolare sequenza $\alpha \in S$, se la sequenza α è momentaneamente non frequente, cioè $\sigma_S(\alpha) < f(|\alpha|)$, la funzione $f^{-1}(\sigma_S(\alpha))$ è la lunghezza minima che una sequenza $\alpha' \supset \alpha$ deve avere prima che essa possa diventare potenzialmente frequente.

La figura 3.2 illustra questa relazione graficamente. La lunghezza di α' non è nient'altro di più che il punto in cui una linea parallela all'asse delle x

interseca la curva di supporto nel punto $y = \sigma_S(\alpha)$; qui, noi essenzialmente assumiamo il miglior caso in cui α' esiste ed è supportata dallo stesso insieme di sequenze come la sua sottosequenza α . Anche questa proprietà è stata inizialmente introdotta per il problema di trovare gli itemset che soddisfano un supporto decrescente con la lunghezza.

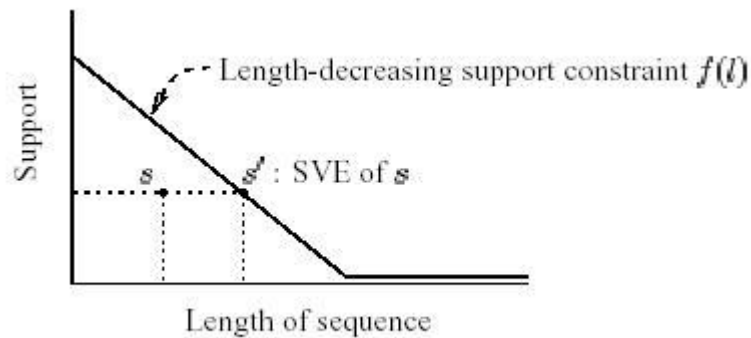


Figura 3.2: Smallest Valid Extension (SVE)

Algoritmi main memory e disk-based

Alcuni algoritmi presentano un'altra utile caratteristica: in casi particolari in cui il database da processare può essere contenuto in memoria principale e grazie anche alle particolari caratteristiche di questi algoritmi il processo di mining può essere interamente svolto in memoria principale senza alcuno accesso al disco, è il caso di Algoritmi come *PrefixSpan* e *SPADE*. Solitamente questa è un'opzione in più, nel senso che questi algoritmi hanno sia una versione main memory che disk-based.

Metodo di formattazione dei dati

Un'altra suddivisione può essere data esaminando il metodo di formattazione dei dati utilizzato, individuiamo così tre classi di algoritmi:

1. *Apriori-based*, metodo di formattazione orizzontale
2. *Apriori-based*, metodo di formattazione verticale

3. *projection-based*, metodo di crescita di pattern basata su proiezione

Supponiamo di essere nel dominio di *Web Usage Mining*, di avere come input delle visite di utenti (tabella 3.2) e di voler conoscere i pattern sequenziali in questi dati. La maggiorparte degli algoritmi di sequence mining usano un

Userid	Time	Visit
1	10	3,4
1	15	1,2,3
1	20	1,2,6
1	25	1,3,4,6
2	15	1,2,6
2	20	5
3	10	1,2,6
4	10	4,7,6
4	20	2,6
4	25	1,7,8

Tabella 3.2: Database originale

layout di database orizzontale; nel formato orizzontale il database consiste di un insieme di sequenze di input; ognuna di essa ha un insieme di eventi (*itemset*), a loro volta formati da item. Dopo la prima scansione si ottiene come risultato gli item frequenti con il relativo supporto; via via i sequential pattern vengono trovati in modo incrementale, ad ogni generico passo comunque, i dati in esame sono sempre le possibili sottosequenze candidate e relativo supporto. Vedi la tabella 3.3.

Spade e *cSPADE* usano un formato di database verticale; per ogni item viene mantenuta una lista detta *id-list* e ogni entrata della *id-list* è una coppia (*sid, eid*) dove *sid* e *eid* sono rispettivamente la sequenza di input in cui l'item occorre e il time-stamp. Nei passi successivi la struttura è sempre quella verticale, cioè al passo *k* troviamo sempre le *id-list* delle sottosequenze candidate. Vedi la tabella 3.4.

Nel terzo caso, gli items frequenti vengono utilizzati per proiettare ricorsivamente il database di sequenze originale in un insieme di database proiettati più piccoli e crescere i frammenti delle sottosequenze in ciascuno dei database proiettati. Questo processo partiziona sia i dati che l'insieme dei pattern frequenti che devono essere testati, e restringe ogni test al corrispondente

Sequence Database		
Userid	Time	Sequence
1	25	$\langle (3, 4)(1, 2, 3)(1, 2, 6)(1, 3, 4, 6) \rangle$
2	20	$\langle (1, 2, 6)(5) \rangle$
3	10	$\langle (1, 2, 6) \rangle$
4	25	$\langle (4, 7, 8)(2, 6)(1, 7, 8) \rangle$

Frequent 1-sequence	
Sequence	Support
(1)	4
(2)	4
(4)	2
(6)	4

Frequent 2-sequence	
Sequence	Support
(1,2)	3
(1,6)	3
(2)(1)	2
(2,6)	4
(4)(1)	2
(4)(2)	2
(4)(6)	2
(6)(1)	2

Tabella 3.3: Formattazione dei dati orizzontale

più piccolo database proiettato. Le dimensioni di ogni database proiettato generalmente, ma non necessariamente, diminuiscono rapidamente con la ricorsione. Per esempio, utilizzando come dati di input sempre il database della tabella 3.2, la figura sotto mostra i database proiettati dalle sequenze frequenti di lunghezza 1 usando l'algoritmo *PrefixSpan*; si noti che gli item non frequenti sono stati filtrati. Vedi tabella 3.5.

Modelli continui, discontinui e ibridi

Una delle classificazioni precedentemente date sui pattern sequenziali è quella che li vede classificati in:

1. pattern continui. Tutti gli elementi del modello appaiono in ordine consecutivo anche nel database di sequenze.
2. pattern discontinui. Gli elementi del modello appaiono in ordine non consecutivo (anche se non strettamente).

Formato dei dati verticale																											
<p><i>id-list</i> item (1)</p> <table> <tr><th><i>sid</i></th><th><i>eid</i></th></tr> <tr><td>1</td><td>15</td></tr> <tr><td>1</td><td>20</td></tr> <tr><td>1</td><td>25</td></tr> <tr><td>2</td><td>15</td></tr> <tr><td>3</td><td>10</td></tr> <tr><td>4</td><td>25</td></tr> </table>	<i>sid</i>	<i>eid</i>	1	15	1	20	1	25	2	15	3	10	4	25	<p><i>id-list</i> item (2)</p> <table> <tr><th><i>sid</i></th><th><i>eid</i></th></tr> <tr><td>1</td><td>15</td></tr> <tr><td>1</td><td>20</td></tr> <tr><td>2</td><td>15</td></tr> <tr><td>3</td><td>10</td></tr> <tr><td>4</td><td>20</td></tr> </table>	<i>sid</i>	<i>eid</i>	1	15	1	20	2	15	3	10	4	20
<i>sid</i>	<i>eid</i>																										
1	15																										
1	20																										
1	25																										
2	15																										
3	10																										
4	25																										
<i>sid</i>	<i>eid</i>																										
1	15																										
1	20																										
2	15																										
3	10																										
4	20																										
	<p><i>id-list</i> item (4)</p> <table> <tr><th><i>sid</i></th><th><i>eid</i></th></tr> <tr><td>1</td><td>10</td></tr> <tr><td>1</td><td>25</td></tr> <tr><td>4</td><td>10</td></tr> </table>	<i>sid</i>	<i>eid</i>	1	10	1	25	4	10																		
<i>sid</i>	<i>eid</i>																										
1	10																										
1	25																										
4	10																										
<p><i>id-list</i> item (6)</p> <table> <tr><th><i>sid</i></th><th><i>eid</i></th></tr> <tr><td>1</td><td>20</td></tr> <tr><td>1</td><td>25</td></tr> <tr><td>2</td><td>15</td></tr> <tr><td>3</td><td>10</td></tr> <tr><td>4</td><td>20</td></tr> </table>	<i>sid</i>	<i>eid</i>	1	20	1	25	2	15	3	10	4	20	<p><i>id-list</i> item (6)(1)</p> <table> <tr><th><i>sid</i></th><th><i>eid</i></th></tr> <tr><td>1</td><td>20</td></tr> <tr><td>4</td><td>20</td></tr> </table> <p>.....</p>	<i>sid</i>	<i>eid</i>	1	20	4	20								
<i>sid</i>	<i>eid</i>																										
1	20																										
1	25																										
2	15																										
3	10																										
4	20																										
<i>sid</i>	<i>eid</i>																										
1	20																										
4	20																										

Tabella 3.4: Formattazione dei dati verticale

Database proiettati	
Prefisso	Database proiettato
$\langle(1)\rangle$	$\langle(\underline{2})(1, \underline{2}, 6)(1, \underline{4}, 6)\rangle, \langle(\underline{2}, 6)\rangle, \langle(\underline{2}, 6)\rangle$
$\langle(2)\rangle$	$\langle(1, \underline{26})(1, 4, 6)\rangle, \langle(\underline{6})\rangle, \langle(\underline{6})\rangle, \langle(\underline{6}, 1)\rangle$
$\langle(4)\rangle$	$\langle(6)\rangle, \langle(2, 6)(1)\rangle$
$\langle(6)\rangle$	$\langle(1, 4, 6)\rangle \langle(1)\rangle$

Tabella 3.5: Formato dei dati ottenuto usando il metodo di proiezione

Più precisamente nel primo caso diciamo che un pattern $A = \langle a_1, a_2, \dots, a_n \rangle$ occorre in una sequenza $A = \langle b_1, b_2, \dots, b_m \rangle$ di un sequence database S se

esistono degli interi i tali che $a_1 = b_i, a_2 = b_{i+1}, \dots, a_n = b_{i+n-1}$. Per esempio, il pattern continuo $\langle AB \rangle$ occorre nella sequenza (A,B,K,F,P) ma non in (K,Q,A,D,B). Nel secondo caso, un pattern discontinuo $A = \langle a_1, a_2, \dots, a_n \rangle$ occorre in una sequenza $A = \langle b_1, b_2, \dots, b_m \rangle$ di un sequence database S se esistono degli interi $i_1 < i_2 < \dots < i_n$, tali che $a_1 = b_{i_1}, a_2 = b_{i_2}, \dots, a_n = b_{i_n}$. Per esempio, i pattern discontinui $\langle ABKF \rangle, \langle BKF \rangle, \langle BF \rangle, \langle K \rangle$, occorrono tutti nella sequenza (A,B,K,F,P).

La gran parte degli algoritmi di mining proposti fino ad ora possono essere classificati in algoritmi che estraggono modelli continui e algoritmi che estraggono modelli discontinui. In particolari i secondi riescono a catturare in un certo senso i pattern continui (come si può notare nella definizione gli elementi del modello possono anche essere continui), hanno però un problema: essi restituiscono come output tutti i pattern trovati senza riuscire a distinguere quali sono continui e quali no. Proprio per queste esigenze, negli ultimi tempi sono stati proposti dei nuovi algoritmi per il sequential pattern mining che riescono non solo a catturare tutti i pattern ma riescono proprio ad individuare la natura di questi modelli (a risalire al tipo di pattern e sottopattern). Questi tipi di modelli costituiti sia da pattern continui che da pattern discontinui dove si è in grado di risalire a ciascuno di loro prendono il nome di modelli ibridi.

Nuovi studi si stanno oggi muovendo verso algoritmi che siano in grado di estrarre anche i modelli ibridi, i primi esempi sono *GFP1* e *GFP2*.

Infatti, in alcuni casi, può essere necessario estrarre dei pattern di questo tipo.

Visto che il tipo di applicazione che seguirà la mia analisi sulle varie problematiche affrontate dagli algoritmi di sequential pattern mining riguarda l'implementazione di un algoritmo nel particolare dominio di Web Mining, non risulta essere di fondamentale importanza l'estrazione di pattern continui, pertanto mi sono orientata allo studio di algoritmi che calcolano l'insieme completo di pattern discontinui.

3.3.2 Classificazione degli algoritmi esaminati

Una volta fatta questa classificazione vediamo le caratteristiche generali di alcuni degli algoritmi più significativi.

Gli algoritmi *Apriori-like*, **AprioriAll**, **AprioriSome**, **DinamicSome**, **GSP**,..., sono molto simili fra di loro; fra questi *AprioriAll* e *GSP* rappresentano i primi studi validi sul *sequential pattern mining*; essi sono stati spesso utilizzati come dei buoni termini di paragone per tanti altri successivi algoritmi di mining.

Fra i primi proposti (*AprioriAll*, *AprioriSome*, *DinamicSome*), *AprioriAll* è l'unico ad estrarre l'insieme completo dei pattern sequenziali frequenti. Gli autori presentano due famiglie di algoritmi, chiamate *count-all* e *count-some*. Gli algoritmi che appartengono alla prima famiglia, a cui *AprioriAll* appartiene, calcolano tutte le sequenze frequenti comprese quelle non massimali (le sequenze non massimali verranno poi scartate in una fase successiva), essi sono basati sull'algoritmo *Apriori* utilizzato per le regole di associazione. Gli algoritmi che appartengono alla seconda famiglia si basano sull'intuizione che, dal momento che sono di nostro interesse le sequenze massimali, si può evitare di calcolare sequenze che sono contenute in sequenze più lunghe se si calcolano queste per prima.

Questi algoritmi sono di natura iterativa, usano una restrizione di supporto costante, si basano sul formato dei dati orizzontale, hanno solo una versione *disk-based* e non main memory e calcolano l'insieme di tutti i modelli discontinui. Un discorso a parte va fatto per l'argomento "restrizioni"; in generale essi non hanno alcun tipo di restrizione se non quella di supporto; l'unico che rappresenta un passo avanti in questa strada è *GSP* (*Generalized Sequential Pattern*) da cui il nome. Questo algoritmo estende gli algoritmi *AprioriAll* e *AprioriSome* aggiungendo alcune restrizioni di tempo come intervallo minimo e massimo tra le occorrenze di elementi (*min-gap*, *max-gap*), finestre di tempo (*sliding time windows*) e gerarchie.

Spirit(**S**equential **P**attern mining **R**egular express**I**on cons**T**raints) ricalca per le caratteristiche di base gli algoritmi *Apriori-like*, la sua particolarità è il fatto che calcola l'insieme dei pattern sequenziali usando come restrizioni delle espressioni regolari. In questo approccio dove le richieste specifiche dell'utente sono tradotte in una espressione regolare sono state introdotte quattro versioni di Spirit: esse propongono un'espressione regolare e una serie di operatori che possono leggere e interpretare l'espressione regolare di restrizione. La differenza fra questi quattro algoritmi è legata a quali restrizioni sono inserite nella generazione dei candidati e nei processi di *pruning* dell'intero mining.

SPADE, cSPADE. La grande novità di questi algoritmi è quella di usare un formato dei dati orizzontale (che risulta minimizzare notevolmente gli accessi I/O) e di poter essere, essendo i dati processati separatamente, un primo passo verso algoritmi paralleli. Essi utilizzano una restrizione di supporto costante, in particolari condizioni possono anche essere *main-memory*, e per quanto riguarda il fattore restrizioni, *SPADE* non ne contiene (ne è privo), *cSPADE* è stato proposto proprio allo scopo di utilizzare l'idea alla base di *SPADE* e arricchirne il processo di mining con esse. Essi calcolano l'insieme di tutti i modelli discontinui.

Anche gli algoritmi **FP-growth, FreeSpan, PrefixSpan, Prefix-Growth, SLPMiner** hanno delle caratteristiche comuni. Essi usano una strategia *divide et impera* partizionando il problema originale in problemi più piccoli che vengono risolti separatamente. Dal momento che essi utilizzano un formato di database basato su proiezione, il mining non è più legato alla fase di “generazione e test” ma solamente alla crescita di pattern (frammenti) frequenti. Di questi algoritmi solamente *PrefixGrowth* permette di inserire nel processo di mining vari tipi di restrizioni tra cui anche quelli più complessi. Tutti questi algoritmi hanno solo una versione *disk-based* escluso solamente *PrefixSpan* che ha anche una versione *main-memory* e inoltre tutti utilizzano una restrizione di supporto costante escluso *SLPMiner* che rappresenta il primo studio su questo nuovo campo di ricerca. Essi calcolano l'insieme di tutti i modelli discontinui.

GFP1 e GFP2. Sono stati proposti da studi molto recenti e presentano una grande novità: calcolano l'insieme completo dei pattern ibridi. Questi algoritmi utilizzano il formato di database orizzontale, sono algoritmi *disk-based* e l'unico tipo di restrizione usata è quella di supporto fisso.

3.3.3 Esempi

Tra i vari algoritmi prima discussi sono stati scelti due algoritmi che verranno esaminati in maniera più approfondita: *GSP, PrefixSpan*.

Algoritmo	Restrizioni	Supporto	Main-Memory	FormatoDati	Pattern
<i>AprioriAll</i>	no	fisso	no	O	D.
<i>GSP</i>	no	fisso	no	O	D
<i>SPIRIT</i>	sì	fisso	no	O	D
<i>SPADE</i>	no	fisso	sì	V	D
<i>cSPADE</i>	sì	fisso	no	V	D
<i>FreeSpan</i>	no	fisso	no	P	D
<i>PrefixSpan</i>	no	fisso	sì	P	D
<i>PrefixGrowth</i>	sì	fisso	no	P	D
<i>SLPminer</i>	no	variabile	no	P	D
<i>GFP1</i>	no	fisso	no	O	I
<i>GFP2</i>	no	fisso	no	O	I

Pattern: D=Discontinui, C=Continui, I=Ibridi
Formato Dati: O=Orizzontale, V=Verticale, P=Proiezione

Tabella 3.6: La Tabella riassume le caratteristiche degli algoritmi esaminati.

3.4 GSP

GSP è un algoritmo *Apriori-like*. Esso è iterativo, consiste di molteplici passi ed ha molto in comune con *AprioriAll*. Vediamo adesso qual è la struttura base dell'algoritmo *GSP*.

3.4.1 Struttura di base

Si inizia trovando tutti i sequential pattern di lunghezza 1 (1-frequent pattern), poi quelli di lunghezza due, tre, e via via vengono generati tutti i sequential pattern di lunghezza maggiore; generalizzando i sequential pattern di lunghezza $k+1$ sono generati utilizzando i sequential pattern di lunghezza k generati al passo precedente.

L'algoritmo effettua molteplici passi sui dati.

Il primo passo determina il supporto di ogni item, cioè il numero di *data-sequence* che includono l'item. Alla fine del primo passo, l'algoritmo conosce quali item sono frequenti. Ogni tale item produce una sequenza frequente di lunghezza 1 consistente di quell'elemento.

Ogni passo successivo inizia con un *seed set*: le sequenze frequenti trovate nel passo precedente. Il seed set è utilizzato per generare nuove potenziali sequenze frequenti chiamate *sequenze candidate*, esse vengono generate facendo una giunzione delle sequenze appartenenti al seed set (vedi la tabella 3.7; si noti che la giunzione può essere fatta anche quando sono gli ultimi itemset ad essere uguali). Ogni sequenza candidata ha un item in più di una

```

insert into  $C_k$ 
select  $p.itemset_1, \dots, p.itemset_{k-1}, q.itemset_{k-1}$ 
from  $L_{k-1}$  p,  $L_{k-1}$  q
where  $p.itemset_1 = q.itemset_1, \dots,$ 
        $p.itemset_{k-2} = q.itemset_{k-2};$ 

```

Tabella 3.7: Generazione delle sequenze candidate a partire dal seed set (L_{k-1} =insieme delle sequenze frequenti di lunghezza $k - 1$).

sequenza appartenente al seed set (seed sequence); questo vuol dire che ad un dato passo le sequenze candidate avranno lo stesso numero di item. Il supporto di queste sequenze candidate è calcolato durante una scansione dei dati. Alla fine del passo l'algoritmo determina quali delle sequenze candidate sono attualmente frequenti.

Queste sequenze frequenti diventano il seed set per il passo successivo. L'algoritmo termina quando, alla fine di un passo, non ci sono sequenze frequenti o quando nessuna sequenza candidata è stata generata.

Quindi, generalizzando, al passo k -esimo, viene utilizzato come seed set l'insieme delle $(k-1)$ -sequence frequenti, ottenuto nel passo precedente, per determinare prima quali sono le sequenze di lunghezza k candidate ad essere frequenti e successivamente da queste, attraverso uno scansione dei dati, estrarne le k -sequence frequenti. Nel primo passo, tutte le sequenze di lunghezza 1, 1-sequence con supporto minimo formano il *seed set*.

Vediamo l'algoritmo in dettaglio.

Vediamo un esempio: figura 3.9.

In questa struttura di base comune anche ad *AprioriAll* ci sono delle limitazioni:

```

Input: un database di sequenze S, soglia di min-sup
Output: l'insieme completo dei pattern sequenziali
Method:
   $F_1 = \text{frequent 1-sequence}$ 
   $F = F_1$  // Insieme di tutte le sequenze frequenti
  for( $k=2$ ;  $F_{k-1} \neq \emptyset$ ;  $k++$ )do
     $C_k =$  insieme delle sequenze candidate di lunghezza  $k$ 
    foreach  $s$  contenuta in S do
      incrementa il contatore di  $\alpha \in C_k | \alpha \sqsubseteq s$ 
     $F_k = \{\alpha \in C_k | \alpha.\text{sup} \geq \text{min-sup}\}$ 
     $F = F \cup F_k$ 
return F

```

Tabella 3.8: GSP

1. Assenza di restrizioni di tempo. Spesso gli utenti possono voler specificare gli intervalli di tempo massimi tra elementi adiacenti del sequential pattern. Per esempio, in una libreria è di poco interesse che due libri vengono acquistati uno dopo l'altro, se l'ultimo è acquistato dopo tre anni.
2. Definizione rigida di una transazione. Per molte applicazioni, non è importante se gli item in un elemento di un modello sequenziale siano presenti in due differenti transazioni, soprattutto se i tempi di occorrenza di quelle transazioni sono all'interno di una piccola finestra di tempo (*time windows*). Per esempio, sempre in una libreria, fissando *time windows* ad una settimana, se un cliente compra un libro A giovedì e un libro B venerdì, e qualche settimana dopo compra nello stesso giorno i libri C e D dovrebbe sempre supportare il modello (A e B) seguito da (C e D).
3. Assenza di gerarchia (*taxonomie*). Molti insiemi dei dati hanno una gerarchia sugli item nei dati definita dall'utente, e gli utenti vorrebbero trovare pattern che includono item secondo livelli diversi di gerarchia.

Essenzialmente *GSP*, *Generalized Sequential Pattern* nasce dall'esigenza di generalizzare l'estrazione dei pattern sequenziali. Esso prende la logica che

Sequence Database		
Userid	Time	Sequence
1	25	$\langle (3, 4)(1, 2, 3)(1, 2, 6)(1, 3, 4, 6) \rangle$
2	20	$\langle (1, 2, 6)(5) \rangle$
3	10	$\langle (1, 2, 6) \rangle$
4	25	$\langle (4, 7, 8)(2, 6)(1, 7, 8) \rangle$

Frequent 1-sequence	
Sequence	Support
(1)	4
(2)	4
(4)	2
(6)	4

Frequent 3-sequence	
Sequence	Support
(1,2,6)	3
(2,6)(1)	2
(4)(2,6)	2
(4)(2)(1)	2
(4)(6)(1)	2

Frequent 2-sequence	
Sequence	Support
(1,2)	3
(1,6)	3
(2)(1)	2
(2,6)	4
(4)(1)	2
(4)(2)	2
(4)(6)	2
(6)(1)	2

Frequent 4-sequence	
Sequence	Support
(4)(2,6)(1)	2

Tabella 3.9: Esecuzione dell'algoritmo GSP (minimo supporto = 2).

sta dietro *AprioriAll* e vi incorpora: restrizioni di tempo, time windows scorrevoli, gerarchie. Per primo, vengono aggiunte le restrizioni di tempo che specificano il periodo di tempo massimo e/o minimo tra elementi adiacenti in un pattern. Secondo, viene rilasciata (*relax*) la restrizione che gli item in un elemento di un sequential pattern devono provenire dalla stessa transazione, permettendo invece che gli item devono essere presenti in un insieme di transazioni il cui tempo è all'interno di una *time window* specificata dall'utente. Terzo, data una gerarchia sugli item definita dall'utente, si permetta ai

sequential pattern di includere item attraverso tutti i livelli di gerarchia. Vediamo adesso in dettaglio come vengono trattati queste restrizioni.

3.4.2 GSP e le restrizioni

Con questi nuovi fattori il sequential pattern mining può essere definito come:

data una sequence data D , una gerarchia T , un min-gap e max-gap definiti dall'utente, un sliding window-size definita dall'utente, trovare tutte le sequenze il cui supporto è più grande del minimo supporto.

Definizione 1 *Sia T un grafo diretto aciclico su un insieme di item. Un arco in T rappresenta una relazione del tipo is_a , e T rappresenta un insieme di gerarchie. Se c'è un arco in T da p a c , noi chiameremo p il padre di c e c il figlio di p (p rappresenta una generalizzazione di c). Noi chiameremo \hat{x} un antenato di x (e x un discendente di \hat{x}) se esiste un arco da \hat{x} a x applicando una chiusura transitiva su T .*

Input. Dato un database d di sequenze chiamate *data-sequence*. Ogni *datasequence* è una lista di transazioni ordinate in modo crescente secondo il tempo di occorrenza delle transazioni (*transaction-time*). Una transazione ha i seguenti campi: *sequence-id*, *transaction-id*, *transaction-time* e gli item presenti nella transazione.

Per semplicità assumiamo che nessuna sequenza ha più di una transazione con lo stesso tempo di occorrenza e che quest'ultimo viene usato come identificatore della transazione stessa, quindi riassumendo: *sequence-id*, *transaction-id*, *item*.

Supporto. Come già sappiamo il supporto di una sequenza è definito come la frazione di tutte le sequenze di input che la contengono.

Vediamo cosa significa che una sequenza è contenuta in un'altra quando vengono aggiunte gerarchie, sliding window e restrizioni di tempo:

- in assenza di questi nuovi fattori, una sequenza di input s' contiene una sequenza s se s è una sottosequenza di s' .
- **più gerarchie:** diciamo che una transazione T *contiene* un item $x \in I$ se x è in T o x è un antenato di alcuni item in T . Diciamo che una

transazione T *contiene* un itemset $y \subseteq I$ se T contiene ogni item in y . Una sequenza di input $d = \langle d_1 \dots d_m \rangle$ contiene una sequenza $s = \langle s_1 \dots s_n \rangle$ se esistono degli interi $i_1 < i_2 < \dots < i_n$ tali che s_1 è contenuta in d_{i_1} , s_2 è contenuta in d_{i_2} , \dots , s_n è contenuta in d_{i_n} . Se non c'è gerarchia questo degenera in un semplice test di sottosequenza.

- **più sliding windows:** rilassa la definizione di quando una sequenza di dati contribuisce al supporto di una sequenza permettendo ad un insieme di transazioni di contenere un elemento di una sequenza fin tanto che la differenza dei (transaction-times) tra le transazioni nell'insieme è inferiore alla dimensione della finestra specificata dall'utente. Formalmente, una sequenza dei dati $d = \langle d_1 \dots d_m \rangle$ contiene una sequenza $s = \langle s_1 \dots s_n \rangle$ se esistono degli interi $l_1 \leq u_1 < l_2 \leq u_2 < \dots < l_n \leq u_n$ tali che

1. s_i è contenuta in $\bigcup_{k=l_i}^{u_i} d_k$, $1 \leq i \leq n$
2. $\text{transaction-time}(d_{u_i}) - \text{transaction-time}(d_{l_i}) \leq \text{window-size}$,
 $1 \leq i \leq n$

- **più restrizioni di tempo:** essi restringono l'intervallo di tempo tra gli insiemi di transazioni che contengono elementi consecutivi di sequenza. Dati window-size, max-gap e min-gap, una sequenza dei dati $d = \langle d_1 \dots d_m \rangle$ contiene una sequenza $s = \langle s_1 \dots s_n \rangle$ se esistono degli interi $l_1 \leq u_1 < l_2 \leq u_2 < \dots < l_n \leq u_n$ tali che

1. s_i è contenuta in $\bigcup_{k=l_i}^{u_i} d_k$, $1 \leq i \leq n$,
2. $\text{transaction-time}(d_{u_i}) - \text{transaction-time}(d_{l_i}) \leq \text{window-size}$,
 $1 \leq i \leq n$
3. $\text{transaction-time}(d_{l_i}) - \text{transaction-time}(d_{u_{i-1}}) > \text{min-gap}$,
 $2 \leq i \leq n$
4. $\text{transaction-time}(d_{u_i}) - \text{transaction-time}(d_{l_{i-1}}) \leq \text{max-gap}$,
 $2 \leq i \leq n$

Noi ci riferiremo al $\text{transaction-time}(d_{l_i})$ come $\text{start-time}(s_i)$, e al $\text{transaction-time}(d_{u_i})$ come $\text{end-time}(s_i)$. In altre parole, $\text{start-time}(s_i)$ e $\text{end-time}(s_i)$ corrispondono al primo e all'ultimo transaction-time dell'insieme di transazioni che contengono s_i .

Passiamo adesso ad analizzare l'algoritmo.

L'algoritmo GSP

L'algoritmo consiste in due fasi principali:

1. generazione delle sequenze candidate (*candidate generation*): come le sequenze candidate vengono generate all'inizio di ogni passo (si vuole generare tutte le possibili sequenze candidate)
2. generazione delle sequenze frequenti (*frequent generation/support counting*) come è determinato il supporto delle sequenze candidate.

Passiamo adesso ad esaminare ciascuno di questi passi considerando le restrizioni di tempo e le sliding windows; in seguito descriveremo le estensioni richieste per trattare le gerarchie.

Candidate Generation

Dentiamo con L_k e C_k rispettivamente l'insieme di tutte le sequenze frequenti di lunghezza k (*k-sequence*) e l'insieme di tutte le sequenze candidate della stessa lunghezza (*candidate k-sequence*). Vediamo cos'è una sottosequenza contigua.

Definizione. Data una sequenza $s = \langle s_1, s_2, \dots, s_n \rangle$ e una sottosequenza c , c è una *sottosequenza contigua* di s se è verificata una delle possibili condizioni:

1. c deriva da s “eliminando” un item da s_1 o da s_n .
2. c deriva da s “eliminando” un item da un elemento s_i che ha almeno due item.
3. c è una sottosequenza contigua di c' , e c' è una sottosequenza contigua di s .

Per esempio, consideriamo la sequenza $s = \langle (1, 2)(3, 4)(5)(6) \rangle$. Le sequenze $\langle (2)(3, 4)(5) \rangle$, $\langle (1, 2)(3)(5)(6) \rangle$ e $\langle (3)(5) \rangle$ sono alcune delle sottosequenze contigue di s . Invece $\langle (1, 2)(3, 4)(6) \rangle$ e $\langle (1)(5)(6) \rangle$ non lo sono.

Possiamo notare che qualsiasi sequenza di input che contiene una sequenza s conterrà anche tutte le sue sottosequenze contigue. Se non c'è nessuna restrizione di intervallo massimo (*max-gap*) allora la sequenza di input s conterrà tutte le sottosequenze di s (incluse le sottosequenze non contigue).

Questa proprietà dà le basi per la fase della generazione dei candidati che può essere a sua volta suddivisa in altre due fasi:

Sequenze frequenti di lunghezza k	Sequenze frequenti di lunghezza k	
	dopo giunzione	dopo potatura
$\langle(1, 2)(3)\rangle$	$\langle(1, 2)(3, 4)\rangle$	$\langle(1, 2)(3, 4)\rangle$
$\langle(1, 2)(4)\rangle$	$\langle(1, 2)(3)(5)\rangle$	
$\langle(1)(3, 4)\rangle$		
$\langle(1, 3)(5)\rangle$		
$\langle(2)(3, 4)\rangle$		
$\langle(2)(3)(5)\rangle$		

Tabella 3.10: Generazione dei Candidati: Esempio

1. *fase di giunzione.* Generiamo le sequenze candidate facendo una giunzione tra due sequenze che appartengono a L_{k-1} . Una sequenza s_1 viene giunta con un'altra sequenza s_2 se la sottosequenza ottenuta eliminando il primo item di s_1 è la stessa di quella ottenuta eliminando l'ultimo item di s_2 . L'item che viene aggiunto può essere inserito come parte di un elemento o come un elemento separato. Si noti che sia s_1 che s_2 sono delle sottosequenze contigue delle nuove sottosequenze candidate.
2. *fase di pruning.* Cancelliamo le sequenze candidate che hanno una sottosequenza contigua di lunghezza $k-1$ che non verifica il minimo supporto. Nel caso in cui non ci sia nessuna restrizione di *max-gap* allora cancelliamo le sequenze candidate che hanno qualsiasi sottosequenza che non verifichi il minimo supporto.

Esempio: la figura 3.10 mostra L_3 e C_4 dopo la fase di giunzione e di potatura. Nella fase di giunzione, la sequenza $\langle(1, 2)(3)\rangle$ giunta con la sequenza $\langle(2)(3, 4)\rangle$ genera $\langle(1, 2)(3, 4)\rangle$, invece giunta con $\langle(2)(3)(5)\rangle$ genera $\langle(1, 2)(3)(5)\rangle$. Le sequenze che rimangono non vengono giunte con nessuna altra sequenza in L_3 . Per esempio la sequenza $\langle(1, 2)(4)\rangle$ non viene giunta con nessuna altra sequenza dal momento che nessuna di esse è della forma $\langle(2)(4, x)\rangle$ o $\langle(2)(4)(x)\rangle$. Nella fase di potatura, $\langle(1, 2)(3)(5)\rangle$ viene eliminata dal momento che la sua sottosequenza contigua $\langle(1)(3)(5)\rangle$ non è in L_3 .

Calcolo del supporto per ogni sequenza (Support Counting)

Per determinare il supporto di ciascuna sequenza candidata andiamo a scandire i dati. In particolare per ogni sequenza nel database individuiamo tutte le sequenze candidate da essa contenute e incrementiamo il loro supporto.

Così dato l'insieme delle sequenze candidate C e una data-sequence d , noi dobbiamo trovare tutte le sequenze candidate in C che sono contenute in d . Per ridurre il numero dei candidati in C che verranno esaminati per verificare se sono contenuti o no nella sequenza d utilizziamo come struttura dati *hash-tree*.

Ridurre il numero dei candidati che hanno bisogno di essere controllati

Un nodo di un hash-tree contiene o una lista di sequenze (nodo foglia) o una tabella hash (nodo interno). In un nodo interno ogni locazione non vuota della tabella hash punta ad un altro nodo. La radice dell'albero è definita avere profondità 1. Un nodo interno che ha profondità p punta ai nodi aventi profondità $p+1$.

Per **aggiungere una sequenza candidata** s all'albero, si inizia dalla radice scorrendo l'albero fino a quando non si raggiunge una foglia. Supponendo di essere su un nodo interno avente profondità p , noi decidiamo quale ramo seguire (cammino) applicando la funzione hash al p -esimo item della sequenza. Tutti i nodi sono inizialmente creati come nodi foglia. Quando il numero dei nodi soglia supera una data soglia, il nodo foglia è convertito in nodo interno.

Per **trovare le sequenze candidate contenute in un data-sequence** partendo sempre dalla radice si applica la procedura che segue basata sui tipi di nodi su cui ci troviamo:

- *nodo interno, se esso è la radice*: applichiamo la funzione hash ad ogni item che è in d e ricorsivamente applichiamo questa procedura ai nodi nelle corrispondenti celle puntate (bucket). Per ciascuna sequenza s contenuta in d , il primo item di s deve essere contenuto in d .
- *nodo interno diverso dalla radice*: assumiamo di aver raggiunto questo nodo applicando la funzione hash ad un item x il cui *transaction-time* è t . Appliciamo la funzione hash a ciascun item in d il cui tempo di transazione è in $[t-\text{window-size}, t+\max(\text{window-size}, \text{max-gap})]$ e ricorsivamente applicare questa procedura al nodo nel corrispondente

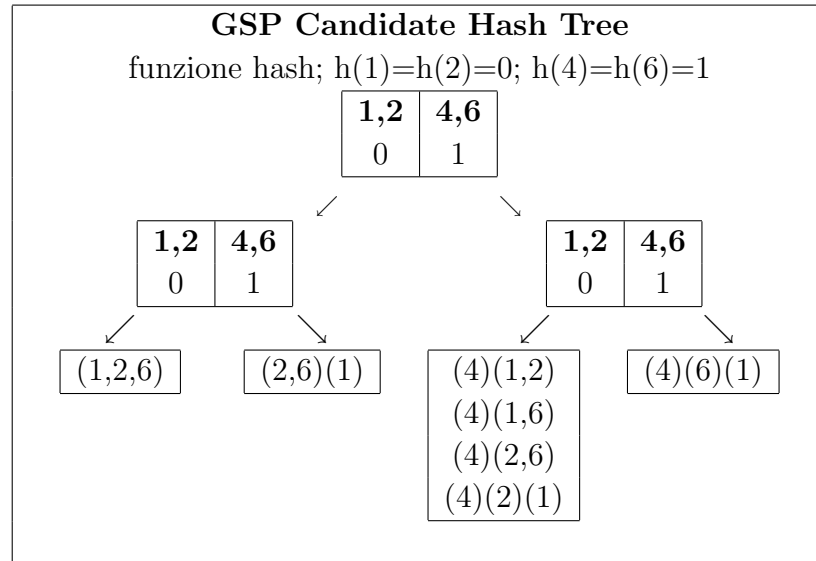


Tabella 3.11: Strutture dati interne di GSP dopo che sono state generate le sequenze candidate di lunghezza 3

bucket.

Per capire meglio questo passo supponiamo di avere una sequenza candidata s con due item consecutivi x e y . Supponiamo che x sia contenuto in d al tempo t , affinché d contenga s y deve apparire al tempo t' compreso in $[t-window-size, t+window-size]$ se y è parte dello stesso elemento di x o nell'intervallo $(t, t+max-gap]$ se y è parte dell'elemento successivo.

- *nodo foglia*: per ogni nodo foglia s noi controlliamo se d contiene s e aggiungiamo s all'insieme di output se necessario (noi discuteremo esattamente dopo come trovare se d contiene una specifica sequenza candidata). Dal momento che noi controlliamo la sequenza contenuta in questo nodo noi riusciremo a catturare tutte le sequenza.

Controllare se una sequenza dei dati di input contiene una sequenza specifica

Sia d una data-sequence e sia $s = \langle s_1, s_2, \dots, s_n \rangle$ una sequenza candidata. Supponendo di avere a disposizione una semplice procedura che riesca a trovare la prima occorrenza di s in d ad un dato tempo descriviamo la procedura che si occupa di controllare se una data-sequence contiene una sequenza specifica.

E' importante notare che dovendo rispettare delle restrizioni di tempo il calcolo del supporto per ogni sequenza candidata risulta essere più complesso di quanto lo era nel caso di *AprioriAll*. Infatti per ogni itemset della sequenza candidata bisogna controllare anche il tempo di occorrenza.

Test

L'algoritmo per controllare se una data sequence d contiene una sequenza candidata s alterna tra due fasi. L'algoritmo inizia nella forward phase dal primo elemento.

- **Forward phase:** vengono trovati gli elementi successivi di s in d tali che la differenza tra l'*end-time* dell'elemento appena trovato e lo *start-time* dell'elemento precedente è meno del *max-gap*, se la differenza non è verificata allora passiamo alla fase *backward*. Se non viene trovato nessuno elemento allora la sequenza s non è contenuta in d .
- **Backward phase:** L'algoritmo torna indietro e si arresta quando trova gli elementi precedenti all'elemento che era in esame nella fase di forward. Se s_i è l'elemento corrente e il suo *end-time*($s_i=t$), l'algoritmo va alla ricerca del primo insieme di transazioni che contengono s_{i-1} i cui relativi *transaction-time* sono dopo $t-\text{maxgap}$, cioè ogni s_{i-1} deve occorrere prima del tempo di occorrenza di s_i però deve sempre verificare il *max-gap*. L'algoritmo si muove indietro fino a quando la restrizione di max-gap tra l'elemento su cui si è fermato e l'elemento precedente è soddisfatta. L'algoritmo allora ritorna alla fase precedente per trovare gli elementi di s in d iniziando dall'elemento dopo l'ultimo elemento su cui si era arrestato (inizia da s_i). Se nessun elemento che verifica le condizioni date è trovato (cioè non c'è nessun insieme subsequent di transazioni che contengono l'elemento) allora la datasequence non contiene s .

Questa procedura è ripetuta passando dalla fase di forward a quella di backward fino a quando non tutti gli elementi vengono trovati.

Esempio: Consideriamo la tabella 3.12

Consideriamo il caso in cui il max-gap è 30, min-gap è 5 e la finestra di tempo è 0. Per la sequenza candidata $\langle(1,2)(3)(4)\rangle$, noi dovremmo trovare prima (1,2) al tempo di occorrenza 10 e dopo (3) al tempo 45. Dal momento che la restrizione di intervallo massimo non è rispettata allora saltiamo la prima

Tempo di occorrenza nella sequenza s	Itemset
10	(1,2)
25	(4,6)
45	(3)
50	(1,2)
65	(3)
90	(2,4)
95	(6)

Tabella 3.12: Esempio di sequenza di input

occorrenza di (1,2). Andiamo così alla ricerca di una occorrenza di (1,2) in un tempo di occorrenza successivo a 15, la troviamo al tempo 50. Dal momento che questo è il primo elemento non abbiamo bisogno di controllare se il max-gap tra questo elemento con l'elemento precedente è verificato. Adesso ci si muove in avanti. Visto che (3) deve occorrere più di 5 giorni (min-gap) dopo (1,2) cerchiamo la prossima occorrenza di (3) dopo il tempo 55; così troviamo (3) al tempo 65 e visto che questa volta è verificata ogni restrizione continuiamo in avanti e troviamo (4) al tempo 90. Il max-gap tra (3) e (4) è soddisfatto; abbiamo finito.

Gerarchie

L'approccio base è quello di rimpiazzare ogni sequenza di input d con una sequenza estesa d' dove ogni transazione d'_i di d' contiene sia gli item della corrispondente transazione d_i di d che gli antenati di ogni item in d_i .

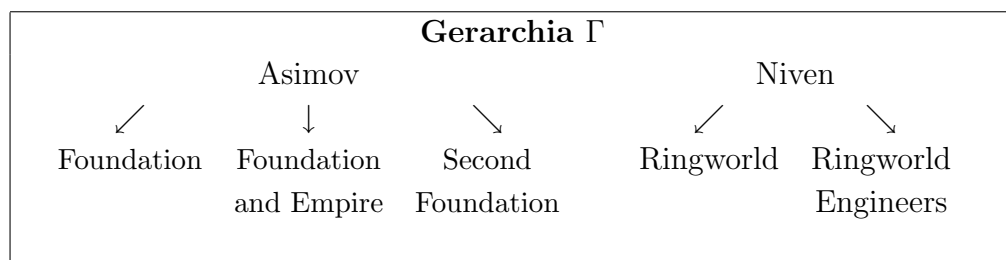


Tabella 3.13: Esempio di gerarchia

Per esempio, con la gerarchia mostrata in 3.13 la sequenza

$$< (Foundation, Ringworld)(SecondFoundation) >$$

sarà rimpiazzata da

$$< (Foundation, Ringworld, Asimov, Niven, ScienceFiction)$$

$$(SecondFoundation, Asimov, Niven, ScienceFiction) >$$

Così si può applicare l'algoritmo *GSP* alle sequenze estese.

3.4.3 GSP rispetto al tempo e alla memoria

E' difficile calcolare in modo rigoroso la complessità in tempo e spazio dell'algoritmo; tanti sono i fattori che influiscono: caratteristiche delle sequenze dei dati in input (dimensioni del database, lunghezza delle sequenze, numero di item distinti), soglia di supporto data, se ci sono o no restrizioni.

Analizziamo quindi quali sono le problematiche e le caratteristiche essenziali che influiscono, e limitiamoci a valutare la complessità al generico passo.

Gestione della memoria

A bassi livelli di supporto, C_k , l'insieme di tutte le sequenze candidate con k item può non essere contenuto interamente in memoria. Senza una buona gestione della memoria questo significa che l'algoritmo deve accedere al disco per recuperare un candidato prima di controllare se è contenuto o no in una sequenza di input, questo ovviamente incide pesantemente sulla performance.

Si osservi che nella fase di generazione dei candidati al passo k -esimo abbiamo bisogno di memorizzare sia l'insieme L_{k-1} che C_k . Nella fase successiva (counting phase) dove andiamo a calcolare il supporto noi abbiamo bisogno di memorizzare C_k e almeno una pagina per *bufferizzare* le transazioni del database.

Complessità in tempo

Facciamo adesso una breve valutazione della complessità di *GSP* in termini di tempo.

Come abbiamo potuto notare l'algoritmo esegue un certo numero di passi che dipendono dai pattern sequenziali più lunghi (ricordiamo che la lunghezza di un pattern è uguale al numero di item contenuti nel pattern), infatti poichè al generico passo i -esimo vengono trovati i sequential pattern di lunghezza i (che hanno un item in più rispetto a quelli trovati al passo precedente) l'algoritmo

esegue tanti passi quanto la lunghezza del/dei pattern più lungo/lunghi. In generale ad ogni passo vengono calcolate le sequenze candidate e poi fatta una scansione dei dati per valutarne il supporto e determinare quali di esse sono frequenti. Guardiamo la tabella 3.14 per capire meglio e valutiamo la complessità ad un generico passo.

Effetti delle restrizione di tempo e delle sliding window

Gli autori di *GSP* hanno studiato la variazione di performance dell'algoritmo in presenza di restrizioni di tempo e finestre scorrevoli. Presentiamo in breve le loro conclusioni.

Le restrizioni di intervallo minimo non degradano la performance; la ragione è che questo tipo di restrizione non tocca la generazione dei candidati, l'efficacia dell'hash-tree nel ridurre il numero dei candidati che hanno bisogno di essere controllati, o la velocità del test.

Per quanto riguarda invece la restrizione di max-gap o sliding window la performance viene penalizzata dal 5% al 30% per diverse ragioni:

1. il tempo per il test aumenta quando vengono usati sia il max-gap che sliding-window.
2. il numero dei candidati aumenta in presenza del max-gap dal momento che non possiamo più potare le sottosequenze non contigue.
3. quando viene utilizzata un sliding-window, l'effetto dell'hash-tree nel potare il numero dei candidati che devono essere controllati rispetto ad una sequenza di input in qualche modo diminuisce.

Per realistici valori di max-gap, *GSP* generalmente nella sua esecuzione è più veloce con le restrizioni che senza, visto che ci saranno un numero inferiore di sequenze candidate. Comunque, specificare una finestra di tempo aumenterà il tempo di esecuzione dal momento che sia l'overhead che il numero di sequential pattern aumenterà.

Limiti dell'algoritmo

Vediamo adesso quali sono i limiti di *AprioriAll* e *GSP* e in generale degli algoritmi basati su *Apriori* detti *Apriori-like*:

- *un enorme insieme di sequenze candidate potrebbe essere generato in un database di sequenze abbastanza lunghe.* Dal momento che le sequenze

<i>ipotesi:</i>		
	N° item distinti= r ;	
	N° sequenze di input nel database= n ;	
	lunghezza massima di una sequenza di input= h ;	
Indichiamo con $ c_i = N^\circ$ sequenze candidate al passo i-esimo		
I passo	$r \times n$	(per ogni sequenza dei dati incrementiamo il supporto di ogni item che vi è contenuto)
II passo	$ c_1 \times n$	(per ogni sequenza dei dati incrementiamo il supporto di ogni sequ. candidata che è contenuta)
III passo	$ c_2 \times n$	
\vdots	\vdots	
\vdots	\vdots	
K-esimo passo	$ c_{k-1} \times n$	dove $k \leq h$
<p>Nota: il numero dei passi dell'algoritmo, cioè la lunghezza massima di un pattern sequenziale sarà al massimo (caso molto difficile) uguale alla lunghezza massima delle sequenze di input, ecco perchè $k \leq h$.</p>		
<i>ipotesi:</i>		
1) al generico passo i-esimo C_i e L_{i-1} sono contenuti interamente in memoria principale		
2) usiamo un hash-tree come quello di figura 3.11 (binario e perfettamente bilanciato)		
complessità al passo k-esimo: $O(C_k \times n)$		
Gli accessi I/O dipendono dalle dimensioni del database e dalla capacità della memoria principale.		

Tabella 3.14: Passi di GSP

candidate di lunghezza k vengono generate da tutte le possibili permutazioni di $(k-1)$ pattern frequenti, e le ripetizioni di item è possibile, il metodo *Apriori-based* può generare un insieme realmente grande di

sequenze di candidati anche per un *seed set* moderato.

- *sono richieste molte scansioni del database.* Dal momento che la lunghezza di ogni sequenza candidata cresce di uno, per estrarre tutte le sequenze frequenti di lunghezza l sono necessari l scansioni del database. Questo, ovviamente, ha un costo molto significativo.
- i metodi *Apriori-based* incontrano delle difficoltà quando si vogliono estrarre long sequential pattern. Questo è dovuto al fatto che i pattern sequential sequenziali *lunghi* vengono fuori da un enorme numero di brevi pattern sequenziali, ma il numero di tali sequenze candidate generate è esponenziale alla lunghezza dei sequential pattern che deve essere estratti.

3.5 PrefixSpan

Adesso presentiamo un recente metodo di sequential pattern mining chiamato **PrefixSpan** (*Prefix-projected Sequential pattern mining*).

Vediamo quale sono le condizioni in cui si sviluppa questo algoritmo.

Come abbiamo già visto, gli algoritmi *Apriori-like* incontrano ancora dei problemi quando un database di sequenze è grande e/o quando i sequential pattern mining che devono essere estratti sono numerosi e/o lunghi. In particolare il loro collo di bottiglia è dato dalla generazione e test delle sequenze candidate. E' possibile assorbire lo spirito di *Apriori* ma evitare o sostanzialmente ridurre le costosi fasi di generazione e test delle sequenze candidate? E' proprio questo quello che cerca di fare *PrefixSpan*.

PrefixSpan sfrutta l'importante intuizione alla base di un altro noto algoritmo *FreeSpan*, riuscendo però a superarne i limiti.

L'idea generale alla base di *FreeSpan* è quella di usare gli item frequenti per proiettare ricorsivamente database di sequenze in un insieme di database proiettati più piccoli e crescere in ognuno di essi frammenti di sottosequenze. Questo processo partiziona sia i dati che l'insieme dei pattern frequenti che devono essere testati, e confina ogni test al corrispondente database proiettato più piccolo. *FreeSpan* ha però dei limiti:

- se un pattern appare in ogni sequenza di un database, il database proiettato non si restringe (tranne che per la rimozione di alcuni item non frequenti);

- dal momento che una sequenza di lunghezza k può accrescersi in qualsiasi punto, la ricerca della sequenza candidata di lunghezza $k+1$ necessiterà di controllare ogni possibile combinazione, questo è molto costoso.

L'idea generale che sta alla base di *PrefixSpan* è quella di esaminare solamente le sottosequenze prefisse e proiettare solamente le corrispondenti sottosequenze postfisse nei “projected-database”. In ogni database così ottenuto, i sequential pattern sono accresciuti esplorando solamente i modelli frequenti locali. Quindi, invece di proiettare database di sequenze considerando tutte le possibili occorrenze di sottosequenze frequenti (come in *FreeSpan*), la proiezione è basata solamente sui prefissi frequenti visto che qualsiasi sottosequenza prefissa può sempre essere trovata accrescendo un prefisso frequente. *PrefixSpan* estrae l'insieme completo di pattern ma riduce in modo notevole gli sforzi di generazione delle sequenze candidate. Inoltre, la proiezione prefissa sostanzialmente riduce la dimensione dei database proiettati e porta ad un calcolo più efficiente. Adesso vediamo un esempio per capire meglio il processo di mining di *PrefixSpan*, l'algoritmo sarà presentato subito dopo.

Estrazioni di pattern sequenziali con proiezione prefissa

Dal momento che gli item all'interno di un elemento di sequenza possono essere elencati (listati) in qualsiasi ordine, senza perdita di generalità, possiamo assumere che essi siano listati in ordine alfabetico o nel caso in cui gli item appartengono ad un insieme di numeri basta elencarli in ordine crescente. Per esempio, una sequenza $s' = \langle (3, 4)(1, 2, 3)(1, 2, 6)(1, 3, 4, 6) \rangle$ è listata in questo modo anziché $\langle (3, 4)(2, 1, 3)(6, 2, 1)(1, 4, 3, 6) \rangle$ oppure la sequenza $s = \langle a(abc)(ac)(ac)d(cf) \rangle$ è listata così anziché $\langle a(bac)(ca)d(fc) \rangle$. Con tale convenzione, l'espressione di una sequenza è unica.

Definizione di Prefisso, Proiezione e Postfisso. 19

Supponiamo che tutti gli item in un elemento sono listati alfabeticamente o in ordine crescente.

*Data una sequenza $\alpha = \langle e_1, e_2 \dots e_n \rangle$, una sequenza $\beta = \langle e'_1, e'_2 \dots e'_m \rangle$ ($m < n$) è chiamata un **prefisso** di α se e solo se:*

(1) $e'_i = e_i$ per $(i \leq m - 1)$;

- (2) $e'_m \subseteq e_m$;
 (3) tutti gli item in $(e_m - e'_m)$ sono alfabeticamente (o in maniera crescente) dopo quelle in e'_m .

Data una sequenza α e β tali che β è una sottosequenza di α , cioè $\beta \sqsubseteq \alpha$. Una sottosequenza α' di una sequenza α (cioè $\alpha' \sqsubseteq \alpha$), è chiamata una **proiezione** di α (rispetto prefisso β) se e solo se:

- (1) α' ha prefisso β
 (2) non esiste nessuna supersequenza propria α'' di α' (cioè $\alpha' \subseteq \alpha''$ ma $\alpha' \neq \alpha''$) tale che α'' è una sottosequenza di α ed ha anche come prefisso β .

Sia $\alpha'' = \langle e_1 e_2 \dots e_n \rangle$ la proiezione di α rispetto al prefisso $\beta = \langle e_1 e_2 \dots e_{m-1} e'_m \rangle$ ($m \leq n$). La sequenza $\gamma = \langle e''_m e_{m+1} \dots e_n \rangle$ è chiamata il **postfisso** di α (rispetto al prefisso β), denotata come $\gamma = \alpha / \beta$, dove $e''_m = (e_m - e'_m)$, denotato anche con $\alpha = \beta \cdot \gamma$.

Se β non è una sottosequenza di α , sia la proiezione che il postfisso di α sono vuoti.

Se l'elemento e''_m è vuoto noi chiameremo la sequenza suffissa come inter-suffix sequence altrimenti come intra-suffix sequence e denoteremo essa come γ_+ .

Gli item all'interno di e''_m sono chiamati intra-extension item del prefisso β , mentre gli item all'interni di e_i ($m + 1 \leq i \leq n$) sono chiamati inter-extension item del prefisso γ . Per distinguere gli elementi di e''_m dagli altri verrà sottolineato il suo primo intra-item.

Per esempio, $\langle 1 \rangle$, $\langle 11 \rangle$, $\langle 1(12) \rangle$ e $\langle 1(123) \rangle$ sono *prefissi* della sequenza $\langle 1(123)(13)4(36)(47) \rangle$, ma nè $\langle 12 \rangle$ nè $\langle 1(23) \rangle$ sono considerati come prefissi. $\langle (123)(13)4(36)(47) \rangle$ è il postfisso della stessa sequenza rispetto al prefisso $\langle 1 \rangle$, $\langle (\underline{2}3)(13)4(36)(47) \rangle$ è il postfisso rispetto al prefisso $\langle 11 \rangle$, e $\langle (\underline{3})(13)4(36)(47) \rangle$ è il postfisso rispetto al prefisso $\langle 12 \rangle$.

Esempio: PrefixSpan Avendo come input il database di sequenze S nella tabella 1 e $min_sup=2$ i pattern sequenziali in S possono essere estratti con il metodo di proiezione prefissa nei seguenti passi.

Sequence Database		
Userid	Time	Sequence
1	25	$\langle(3, 4)(1, 2, 3)(1, 2, 6)(1, 3, 4, 6)\rangle$
2	20	$\langle(1, 2, 6)(5)\rangle$
3	10	$\langle(1, 2, 6)\rangle$
4	25	$\langle(4, 7, 8)(2, 6)(1, 7, 8)\rangle$

Tabella 3.15: Database di sequenze

Passo 1: Trova i pattern sequenziali di lunghezza 1. Viene fatta una scansione di S per trovare tutti gli item frequenti. Ognuno di questi item frequenti è un pattern sequenziale di lunghezza 1. Essi sono $\langle 1 \rangle$: 4, $\langle 2 \rangle$: 4, $\langle 4 \rangle$: 4, $\langle 6 \rangle$: 3, dove $\langle pattern \rangle : count$ rappresentano il pattern il rispettivo supporto.

Passo 2: Dividi lo spazio di ricerca. L'insieme completo dei sequential pattern può essere partizionato nei seguenti quattro sottoinsiemi secondo i quattro prefissi: (1) quelle sottosequenze aventi come prefisso $\langle 1 \rangle$; (2) quelle aventi come prefisso $\langle 2 \rangle$; (3) quelle aventi come prefisso $\langle 4 \rangle$; (4) quelle aventi come prefisso $\langle 6 \rangle$.

Passo 3: Trova i sottoinsiemi di sequential pattern. I sottoinsiemi dei pattern sequenziali possono essere estratti costruendo il corrispondente *database proiettato* e processando ognuno di questi ricorsivamente. Il processo di mining può essere spiegato come segue.

Per prima cosa vengono ricercati i pattern sequenziali aventi prefisso $\langle 1 \rangle$. In particolare, in una sequenza contenente $\langle 1 \rangle$, solamente la sottosequenza prefissa con la prima occorrenza dovrebbe essere considerata. Per esempio, nella sequenza $\langle(3, 4)(1, 2, 3)(1, 2, 6)(1, 3, 4, 6)\rangle$, solamente la sottosequenza $\langle(2, 3)(1, 2, 6)(1, 3, 4, 6)\rangle$ dovrebbe essere scelta per l'estrazione dei sequential pattern aventi prefisso $\langle 1 \rangle$. Si noti che tutti gli item non frequenti non vengono considerati.

Le sequenze in S contenenti $\langle 1 \rangle$ sono proiettate in modo tale da formare il database proiettato di $\langle 1 \rangle$ ($\langle 1 \rangle$ -projected database o $S|_{\langle 1 \rangle}$), che consiste di

tre sequenze postfisse:

$\langle(2)(1, \underline{2}, 6)(1, \underline{4}, 6)\rangle$, $\langle(\underline{2}, 6)\rangle$, $\langle(2, 6)\rangle$. Scandendo il database risultante dalla proiezione ottenuto rispetto al prefisso $\langle(1)\rangle$ troviamo due intra-item frequenti (2 e 6) e otteniamo tutti i sequential pattern di lunghezza 2: $\langle(1, 2)\rangle$ e $\langle(1, 6)\rangle$.

Così $S|_{(1)}$ può essere partizionato in due sottoinsiemi: quelli aventi prefisso $\langle(1, 2)\rangle$; quelli aventi prefisso $\langle(1, 6)\rangle$.

Adesso continuando nello stesso modo possiamo esaminare $S|_{\langle(1, 2)\rangle}$ e $S|_{\langle(1, 6)\rangle}$. L'esempio è interamente presentato nella tabella seguente 3.16. *PrefixSpan* partiziona il problema ricorsivamente, cioè ogni sottoinsieme dei pattern sequenziali può essere suddiviso quando necessario, esso applica in sintesi una metodologia *dividi et impera*.

Definizione 20 Database proiettato

Sia α un pattern sequenziale nel database di sequenze S . Il database proiettato ottenuto come risultato della proiezione di S rispetto ad α denotato con α -**projected database** o $S|_{\alpha}$ è la collezione dei postfissi delle sequenze in S aventi prefisso α .

Definizione 21 Calcolo del supporto in un database proiettato

Sia α un pattern sequenziale nel database di sequenze S e β una sequenza avente prefisso α . Il **supporto** di β in $S|_{\alpha}$ denotato con $\text{support}_{S|_{\alpha}}(\beta)$ è il numero di sequenze γ in $S|_{\alpha}$ tali che $\beta \sqsubseteq \alpha \cdot \gamma$.

Dati α e β , due pattern sequenziali di un database S tali che α è un prefisso di β , si noti che:

1. $S|_{\beta} = (S|_{\alpha})|_{\beta/\alpha}$;
2. per qualsiasi sequenza γ avente come prefisso α vale che $\text{support}_S(\gamma) = \text{support}_{S|_{\alpha}}(\gamma/\alpha)$ e
3. la dimensione di $S|_{\alpha}$ non può superare quella di S .

Basato sul ragionamento fatto sopra, ecco l'algoritmo *PrefixSpan* (3.17)

Analisi

Analizziamo adesso l'efficienza dell'algoritmo.

Prefissi	Database Proiettati	InterItem frequenti	IntraItem frequenti	ξ -pattern
	$\langle(3,4)(1,2,3)(1,2,6)(1,3,4,6)\rangle$ $\langle(1,2,6)(5)\rangle,$ $\langle(1,2,6)\rangle,$ $\langle(4,7,8)(2,6)(1,7,8)\rangle,$	1,2,4,6		$\langle(1)\rangle, \langle(2)\rangle$ $\langle(4)\rangle, \langle(6)\rangle$
$\langle(1)\rangle$	$\langle(2)(1,2,6)(1,4,6)\rangle,$ $\langle(2,6)\rangle, \langle(2,6)\rangle$		2,6	$\langle(1,2)\rangle,$ $\langle(1,6)\rangle$
$\langle(1,2)\rangle$	$\langle(2,6)(6)\rangle, \langle(6)\rangle, \langle(6)\rangle$		6	$\langle(1,2,6)\rangle$
$\langle(1,2,6)\rangle$	$\langle(6)\rangle$	_____	_____	_____
$\langle(1,6)\rangle$	$\langle(6)\rangle$	_____	_____	_____
$\langle(2)\rangle$	$\langle(1,2,6)(1,4,6)\rangle, \langle(6)\rangle,$ $\langle(6)\rangle, \langle(6)(1)\rangle$	1	6	$\langle(2)(1)\rangle,$ $\langle(2,6)\rangle$
$\langle(2)(1)\rangle$	$\langle(6)(1,6)\rangle$	_____	_____	_____
$\langle(2,6)\rangle$	$\langle(1,6)\rangle, \langle(1)\rangle$	1	_____	$\langle(2,6)(1)\rangle$
$\langle(2,6)(1)\rangle$		_____	_____	_____
$\langle(4)\rangle$	$\langle(1,2)(1,2,6)(1,4,6)\rangle,$ $\langle(2,6)(1)\rangle$	1,2,6		$\langle(4)(1)\rangle,$ $\langle(4)(2)\rangle,$ $\langle(4)(6)\rangle$
$\langle(4)(1)\rangle$	$\langle(2)(1,2,6)(1,6)\rangle$	_____	_____	_____
$\langle(4)(2)\rangle$	$\langle(1,2,6)(1,6)\rangle, \langle(6)(1)\rangle$	1	6	$\langle(4)(2)(1)\rangle,$ $\langle(4)(2,6)\rangle$
$\langle(4)(2)(1)\rangle$	$\langle(6)(1,6)\rangle$	_____	_____	_____
$\langle(4)(2,6)\rangle$	$\langle(1,6)\rangle, \langle(1)\rangle$	1		$\langle(4)(2,6)(1)\rangle$
$\langle(4)(2,6)(1)\rangle$		_____	_____	_____
$\langle(4)(6)\rangle$	$\langle(1,6)\rangle, \langle(1)\rangle$	1		$\langle(4)(6)(1)\rangle$
$\langle(6)\rangle$	$\langle(1,4,6)\rangle, \langle(1)\rangle$	1		$\langle(6)(1)\rangle$
$\langle(6)(1)\rangle$		_____	_____	_____

Tabella 3.16: Esempio utilizzando PrefixSpan (minimo supporto = 2).

- **non c'è nessun bisogno di generare le sequenze candidate.** I sequential pattern più lunghi vengono generati da quelli frequenti più corti; *PrefixSpan* cerca i pattern sequenziali su uno spazio di ricerca

Algoritmo (*PrefixSpan*)

Input: un database di sequenze S , soglia di minimo supporto min_sup

Output: l'insieme completo dei pattern sequenziali

Metodo: *PrefixSpan* ($\langle \rangle$, l , $S|_{\alpha}$)

Sottoroutine *PrefixSpan* (α , l , $S|_{\alpha}$)

Parametri α : un modello sequenziale; l : la lunghezza di α ;
 $S|_{\alpha}$: il database proiettato di α , if $\alpha \neq \langle \rangle$; altrimenti il database di sequenze S .

Metodo:

1. Scandisci $S|_{\alpha}$ una volta, trova l'insieme degli item frequenti b tali che:
 - (a) b può essere assemblato all'ultimo elemento di α ;
 (cioè b è un intra-item)
 - (b) $\langle b \rangle$ può essere 'appeso' ad α in modo da formare un pattern sequenziale.
 (cioè b è un inter-item)
2. Per ogni item frequente b , 'appendi' esso ad α in modo da formare un sequential pattern α' , e dai come output α' ;
3. Per ogni α' , costruisci il database proiettato di α' ($S|_{\alpha'}$), e richiama *PrefixSpan*(α' , $l+1$, $S|_{\alpha'}$).

Tabella 3.17: Algoritmo PrefixSpan

più piccolo.

- **i database proiettati tendono a restringersi.** Come abbiamo visto un database proiettato è più piccolo di quello originale poichè vengono proiettate solamente le sottosequenze postfixe; in pratica il fattore di

restringimento può essere significativo in quanto.

1. generalmente solamente un piccolo insieme di pattern sequenziali diventa abbastanza “lungo”, così il numero delle sequenze nei database proiettati sarà abbastanza piccolo quando il prefisso cresce.
 2. la proiezione prende solo la parte postfissa di un sequenza.
- **il maggior costo di PrefixSpan è nella costruzione dei database proiettati.** Nel caso peggiore *PrefixSpan* costruisce un database proiettato per ogni pattern sequenziale. Sono state proposte varie soluzioni a questo problema.

Soluzioni proposte per diminuire il costo impiegato nella costruzione dei database proiettati.

Il tipo di proiezione di cui abbiamo parlato fino adesso è chiamata *proiezione level by level*; ad ogni passo (livello) dell’algoritmo, cioè per ogni pattern sequenziale trovato, viene fatta una proiezione. Abbiamo visto che per il momento il principale costo di *PrefixSpan* è proprio nella proiezione; quindi se il numero e/o la dimensione dei database proiettati potessero essere ridotti, la performance di *PrefixSpan* per trattare il sequential pattern mining potrebbe essere migliorato sostanzialmente. Proprio per questo è stato proposto un altro tipo di proiezione chiamata *proiezione bi-level*; vediamo in cosa consiste.

Il primo passo è sempre lo stesso: scansione di S per trovare i pattern sequenziali di lunghezza 1 che sono $\langle 1 \rangle$, $\langle 2 \rangle$, $\langle 4 \rangle$, $\langle 6 \rangle$.

Al secondo passo, invece di costruire il database proiettato per ogni sequential pattern di lunghezza 1, si costruisce una matrice triangolare inferiore M come mostrato nella tabella 3.18.

La matrice M registra i supporti di tutte le sequenze di lunghezza 2 che sono ottenuti dall’“assemblaggio” dei pattern sequenziali di lunghezza 1. Gli elementi della matrice sulla diagonale contengono un solo contatore, per esempio $M[1, 1] = 1$ e indica che la sequenza $\langle 1 \ 1 \rangle$ appare in una sequenza del database s . Gli altri elementi della matrice hanno invece tre contatori, per esempio $M[1, 2] = (1, 2, 3)$ e significa che $support_S(\langle 1 \ 2 \rangle) = 1$, $support_S(\langle 2 \ 1 \rangle) = 2$, $support_S(\langle (1 \ 2) \rangle) = 3$. Dal momento che l’informazione in una cella è simmetrica, cioè $M[1, 2] = M[2, 1]$ una matrice triangolare

1	1			
2	(1,2,3)	1		
4	(1,2,1)	(1,2,0)	1	
6	(1,2,3)	(1,0,4)	(2,1,1)	1
	1	2	4	6

Tabella 3.18: Proiezione Bi-level: matrice S

è sufficiente. Questa matrice è chiamata *S-matrix*.

Scandendo il database di sequenze S un'altra volta, la *S-matrix* può essere riempita; tutti i pattern sequenziali di lunghezza 2 possono essere identificati dalla matrice immediatamente.

Per ogni α , sequential pattern di lunghezza 2, si costruisce α -*projected database*. Per esempio $\langle 4\ 2 \rangle$ è ottenuto come *2-sequential pattern* dalla *S-matrix*. Il $\langle 4\ 2 \rangle$ -*projected database* contiene due sequenze: $\langle (1, 2, \underline{6})(1, 6) \rangle$ e $\langle \underline{6}(1) \rangle$. Scandendolo vengono trovati un intra-item frequente 6 e un inter-item frequente 1. Così viene costruita la S-matrix 2×2 per il $\langle 4\ 2 \rangle$ -*projected database*, come mostrato nella tabella 3.19.

Dal momento che c'è solo una cella con supporto 2, può essere generato

1	1
(<u>6</u>)	(1,2,1) \emptyset
	1 (<u>6</u>)

Tabella 3.19: S-matrix in $S|_{4\ 2}$

solamente il pattern di lunghezza 2 $\langle (\underline{6})\ 1 \rangle$ e non è necessaria nessun'altra proiezione. Si noti che il simbolo \emptyset sta ad indicare che non è possibile generare un tale modello. Così non c'è più bisogno di scorrere il database. Utilizzando la proiezione *bi-level* è necessario un numero di database proiettati inferiore a quello richiesto utilizzando la proiezione *level by level*. Adesso diamo una giustificazione del processo di mining utilizzando la proiezione *bi-level*

Definizione. 22 Sia α un pattern sequenziale di lunghezza l , e siano $\alpha'_1, \alpha'_2, \dots, \alpha'_m$ tutti i pattern equenziali di lunghezza $(l+1)$ aventi prefisso α all'interno del α -projected database ; la *S-matrix* di α -projected database denotata con $M[\alpha'_i, \alpha'_j]$ ($1 \leq i \leq j \leq m$) è definita come segue:

1. $M[\alpha'_i, \alpha'_i]$ contiene un contatore. Se l'ultimo elemento di α'_i ha un solo item x , cioè $\alpha'_i = \langle \alpha x \rangle$, il contatore registra il supporto della sequenza $\langle \alpha'_i x \rangle$ in α -projected database. Altrimenti il contatore è settato a \emptyset .
2. $M[\alpha'_i, \alpha'_j]$ ($1 \leq i \leq j \leq m$) è nella forma di (A, B, C) dove A, B, C sono tre contatori.
 - se l'ultimo elemento in α'_j ha solo un item x , cioè $\alpha'_j = \langle \alpha x \rangle$, il contatore A registra il supporto della sequenza $\langle \alpha'_i x \rangle$ in α -projected database. Altrimenti il contatore A è settato a \emptyset ;
 - se l'ultimo elemento in α'_i ha solo un item y , cioè $\alpha'_i = \langle \alpha y \rangle$, il contatore B registra il supporto della sequenza $\langle \alpha'_j y \rangle$ in α -projected database. Altrimenti il contatore B è settato a \emptyset ;
 - Se gli ultimi elementi di α'_i e α'_j hanno lo stesso numero di item, il contatore C registra il supporto della sequenza α'' in α -projected database, dove la sequenza α'' è α'_i ma inserendo nell'ultimo elemento di α'_i l'item nell'ultimo elemento di α'_j ma non in quello di α'_i . Altrimenti il contatore C è settato a \emptyset .

Si noti che la *S-matrix* può essere riempita con solo due scansioni del database proiettato.

Inoltre può essere fatta un'ulteriore ottimizzazione. Si noti che nella *S-matrix* in tabella (la prima) $\langle 4 \ 1 \rangle$ rappresenta un pattern sequenziale e $\langle 1 \ 2 \rangle$ no; secondo la nota proprietà di *Apriori* $\langle 4 \ 1 \ 2 \rangle$ e qualsiasi sua supersequenza non può essere un sequential pattern, così basandosi sulla matrice l'item 2 può essere escluso dal $\langle 4 \ 1 \rangle$ -projected database. Questo controllo è chiamato *3-way Apriori* e rende possibile una costruzione efficiente dei database proiettati; esso dovrebbe essere impiegato per potare item nella costruzione dei database proiettati permettendo così di ridurre le dimensioni. Vengono proiettati solamente i frammenti delle sequenze necessarie a fare crescere i pattern più lunghi.

Pseudo proiezione

Come abbiamo visto il costo maggiore di *PrefixSpan* è la proiezione, cioè

formare i database proiettati ricorsivamente. Con la proiezione *bi-level* il numero delle proiezioni viene ridotto; nel caso in cui un database proiettato può essere contenuto in memoria principale viene proposta un'ulteriore tecnica che riduce in maniera sostanziale il costo della proiezione che si chiama *pseudo proiezione*.

Esaminando un insieme di database proiettati, si può osservare che i postfissi di una sequenza spesso appaiono ripetutamente nei database proiettati ricorsivi. Nell'esempio che abbiamo visto precedentemente la sequenza $\langle(3, 4)(1, 2, 3)(1, 2, 6)(1, 3, 4, 6)\rangle$ ha postfissi $\langle(1, 2)(1, 2, 6)(1, 4, 6)\rangle$, $\langle(1, 2, 6)(1, 4, 6)\rangle$ e $\langle(1, 2, 6)(1, 6)\rangle$ come proiezioni nel database proiettato rispettivamente di $\langle 4 \rangle$, $\langle 2 \rangle$ e $\langle 4\ 2 \rangle$. Ci sono quindi pezzi ridondanti di sequenza. Se il database di sequenza/database proiettato può essere mantenuto in memoria principale tale ridondanza può essere evitata con la pseudo proiezione. Vediamo in cosa consiste questo metodo.

In caso di *main memory*, invece di costruire una proiezione fisica mettendo assieme tutti i postfissi, si possono usare dei puntatori che riferiscono (puntano) alle sequenze nel database come una pseudoproiezione. Ogni proiezione consiste di due pezzi di informazione:

- un *puntatore* alla sequenza nel database;
- un *offset* del postfisso nella sequenza.

Una volta acceduto alla sequenza, l'*offset* mi indica l'inizio della proiezione in essa.

La pseudo-proiezione evita di avere fisicamente più copie dei postfissi. Questo è efficiente sia in termini di tempo di esecuzione che di spazio. Ovviamente questo tipo di proiezione non è efficiente se usata per accessi basati su disco. Gli autori suggeriscono di utilizzare la pseudo-proiezione quando i database proiettati possono stare in memoria e alcuni loro risultati sperimentali mostrano che una soluzione integrata, proiezione bi-level per processare su disco e pseudo-proiezione quando i dati possono stare in memoria, ottiene ottimi risultati in termini di performance.

3.5.1 PrefixSpan rispetto al tempo e alla memoria

Anche per *PrefixSpan* è molto difficile calcolare in modo rigoroso la complessità in tempo e spazio; tanti sono i fattori che influiscono: caratteristiche

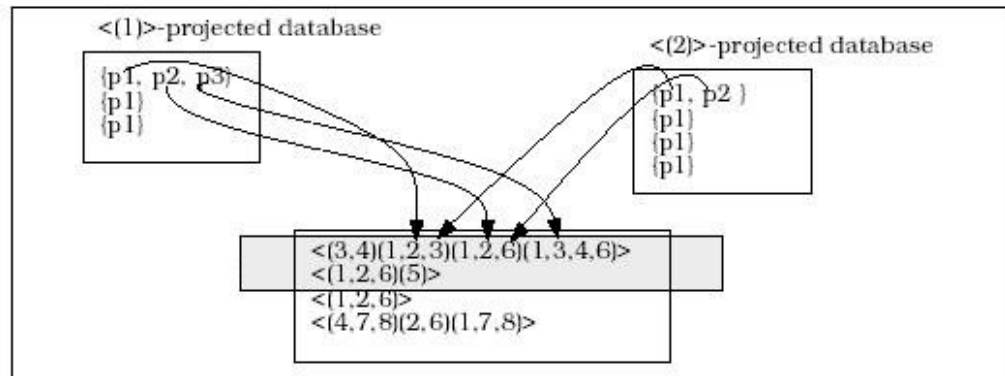


Figura 3.3: PrefixSpan: Pseudo Proiezione con finestra di memoria virtuale

delle sequenze dei dati in input (dimensioni del database, lunghezza delle sequenze, numero di item distinti, supporto degli item), soglia di supporto data; oltre a questi bisogna aggiungerne altri come il grado di selettività o restringimento dell'algoritmo ad ogni passo.

Inoltre considerazioni diverse vanno fatte in base al tipo di proiezione utilizzata.

Analizziamo quindi quali sono le problematiche e le caratteristiche essenziali che influiscono, e limitiamoci a valutare la complessità al generico passo.

Iniziamo a fare una prima analisi dell'algoritmo utilizzando la proiezione *level-by-level*; vediamo quali sono le operazioni eseguite in ciascun passo.

In generale il numero di proiezioni da fare al I passo dipende dal numero di item frequenti, supponiamo siano r ; otteniamo così r database proiettati (ovviamente uno per ogni item frequenti). Le dimensioni dei database proiettati dipendono dalle caratteristiche del database originale; comunque in generale le sottosequenze postfisse:

- hanno almeno un elemento in meno rispetto alla lunghezza delle sequenze di input;
- contengono solo gli item frequenti (tutti quelli non frequenti vengono eliminati).

Il risultato del I passo quindi sono r database proiettati. A questo punto il procedimento si ripete per ciascuno di questi database.

Il grado di restringimento dell'algoritmo dipende enormemente dalle caratteristiche del database; maggiore è il numero di item frequenti maggiore è quello dei database proiettati; maggiore è l'occorrenze degli item nelle sequenze, maggiore sarà la dimensione dei database proiettati.

In generale, comunque, ricordando quanto detto prima alla voce analisi (paragrafo 3.17) solitamente le dimensioni di questi database vanno notevolmente a diminuire e solo una piccola parte di loro viene processato fino alla fine. Insomma se pensiamo al processo di mining come ad un albero la cui radice è rappresentata dal database originale, i cui nodi interni sono rappresentati dai database proiettati e i cui cammini sugli archi rappresentano il prefisso rispetto a cui viene fatta la proiezione, difficilmente l'albero si presenta con tanti rami che giungono fino alla profondità dell'albero.

Riassumendo, al I passo, ipotizzando che il database originale abbia un numero di sequenze pari ad n e che gli item frequenti siano r , abbiamo una complessità $O(r \times n)$. Al passo successivo i database da scandire sono r , la complessità dipende ovviamente dalle loro dimensioni perchè ognuno di essi va di nuovo scandito e su ciascuno riapplicato il procedimento. Supponendo che il numero delle sequenze postfisse nei database proiettati sia in media n' ($n' < n$) e che in media gli item frequenti siano r' ($r' \leq r$) abbiamo una complessità per ogni database proietato di $O(r' \times n')$

E così per ogni altro passo tenendo conto che sia le dimensioni dei database proiettati che il numero degli item frequenti tendono a diminuire in alcuni casi sostanzialmente.

Si noti che il numero di accessi I/O dipende dalle dimensioni dei database oltre che dalla capacità della memoria principale.

Per quanto riguarda l'occupazione in memoria, ovviamente il principale costo riguarda l'occupazione dei database proiettati.

Nel caso della proiezione *bi-level*, proposta per ridurre il numero e la dimensione dei database proiettati, usando la *S-matrix* per rappresentare tutti i supporti delle sequenze di lunghezza 2, il numero dei database proiettati diventa più piccolo e richiedono meno spazio.

Ovviamente la *pseudo-proiezione* è più efficiente rispetto agli altri due metodi di proiezione.

3.6 Estensioni degli algoritmi di mining per speciali scopi

Passiamo adesso a discutere in breve di alcuni possibili attuali estensioni degli algoritmi di mining.

Algoritmi paralleli

L'enormità dei dati può rendere gli algoritmi sequenziali molto costosi in termini di spazio e tempo di esecuzione. Come in molte altre applicazioni l'esecuzione parallela, *parallel processing*, può ridurre (specialmente) l'*overhead* di tempo enormemente. Così trovare algoritmi efficienti è un'area di interesse molto importante.

Una prima proposta [16] propone quattro algoritmi; gli autori dell'articolo affermano che sono possibili diverse strategie di parallelizzazioni dipendenti dal numero di oggetti nei dati di input, dal numero di eventi e dalle restrizioni di tempo. Questa prima formulazione sostengono diventi inefficiente quando il numero delle sequenze candidate diventa elevato così ne viene proposta un'altra in cui i candidati sono anche partizionati. Non è data nessuna implementazione.

Anche l'algoritmo *SPADE* prima discusso ha alcune utili caratteristiche parallelizzabili come dividere il problema in sottosezioni che possono essere risolte indipendentemente l'uno dall'altro.

Questo campo di ricerca è ancora all'inizio e in futuro vari saranno gli studi su questa direzione.

Algoritmi incrementali

Molto spesso i dati da processare sono sottoposti a cambiamenti; pertanto molte volte potrebbe essere più utile utilizzare degli algoritmi incrementali per attuare il processo di mining anziché applicare di nuovo l'intero processo al database originale. Algoritmi incrementali proprio per questo, perché estraggono i pattern sequenziali in maniera incrementale, cioè utilizzando le informazioni ottenute nel passo precedente di mining, anziché riapplicare il processo ex-novo senza sfruttare le conoscenze estratte prima. Nel campo del sequential pattern mining, in particolare, questo è più importante dal mo-

mento che i dati sequenziali da cui ricavare la conoscenza utile sono spesso in continuo cambiamento. Per questo motivo vari ricercatori stanno cercando di trovare degli algoritmi che siano più efficienti nel trattare il mining incrementale.

L'incremental mining problem è definito in [30] come segue:

Definizione 23 *Dato un database D di sequenze, esso è aggiornato inserendo un insieme di sequenze Δ^+ e cancellando un insieme Δ^- , il database ottenuto è denotato con Δ' , l'insieme delle sequenze che non hanno subito variazione sono denotate con $D^- = D - \Delta^- = D' - \Delta^+$, γ_D^s rappresenta il valore del supporto di s nel database D , $|D|$ rappresenta il numero di sequenze in D . Lo scopo del mining incrementale è quello di trovare tutte le sequenze frequenti massimali nel database D' dato Δ^- , D^- , Δ^+ e il risultato del mining su D .*

$$D \Rightarrow \begin{array}{|c|} \hline \Delta^- \\ \hline D' \\ \hline \Delta^+ \\ \hline \end{array} \Leftarrow D'$$

Tabella 3.20: Definizione di D , D' , Δ^- , Δ^+ e D^- .

Questo è un recente campo di studi che sarà ampiamente approfondito.

Algoritmi multidimensional

Fino adesso abbiamo sempre discusso del sequential pattern mining ad una singola dimensione, cioè quando siamo alla ricerca dei pattern sequenziali consideriamo un solo attributo assieme al suo *time-stamp*. Con il termine *mining multiple dimensional sequential pattern* intendiamo un mining che tenga conto di altre dimensioni di informazioni, ovviamente esso può darci pattern più utili.

Per esempio noi possiamo ottenere un pattern frequente da un database di un supermarket che dice che la maggior parte di clienti che compra il prodotto1 compra anche il prodotto2 in un definito intervallo di tempo utilizzando il sequential pattern mining; usando il multidimensional sequential pattern mining noi possiamo trovare inoltre differenti gruppi di persone che hanno diversi pattern di acquisto. Questo nuovo tipo di mining è stato per la prima volta analizzato in [21]. In esso vengono introdotti vari attributi diversi

dal *transaction_id* ed essi vanno a formare un insieme di dati sequenziali multidimensionali come mostrato nella tabella 3.21.

Lo scopo di questo speciale mining è quello di ottenere pattern sequenziali più interessanti con diversi attributi dimensionali. Di recente è stato anche proposta una estensione diretta di *PrefixSpan*.

cid	cust-grp	city	age-grp	sequence
10	business	Boston	middle	$\langle (b, d) cba \rangle$
20	professional	Chicago	young	$\langle (b, f) (c, e) (f, g) \rangle$
30	business	Chicago	middle	$\langle (a, h) abf \rangle$
40	education	New York	retired	$\langle (b, e) (c, e) \rangle$

Tabella 3.21: Un database di sequenze multidimensionale

Ancora oggi il multidimensional mining non è stato esplorato assieme con altre restrizioni come *time-gap*, *sliding window* e restrizioni di item. Gli argomenti delle future ricerche in questo campo saranno: multidimensional pattern mining con restrizioni, multidimensional pattern mining interattivi o multidimensional pattern mining integrati con gerarchie (taxonomies).

Capitolo 4

Implementazione di un algoritmo per inferire Sequential Pattern: PrefixSpan

4.1 Introduzione

Una volta discusso dei pattern sequenziali, di come essi siano concepiti ad hoc per estrapolare conoscenza utile in dati che hanno un'intrinseca natura sequenziale, e di Web Usage Mining, andiamo adesso a focalizzare la nostra attenzione sull'obiettivo finale di questa tesi: scegliere un algoritmo per inferire pattern sequenziali al fine di estrapolare pattern d'uso nel dominio Web. Nei capitoli precedenti, abbiamo visto come Web Usage Mining sia non solo un campo emergente ma anche in continua espansione e come molte delle sue applicazioni hanno a che fare col fattore tempo. Da questa semplice osservazione risulta chiaro pensare di poter sfruttare le caratteristiche del modello di mining *sequential pattern* nel contesto web, e quindi applicare algoritmi di *sequential pattern mining* in molte applicazioni del Web Usage Mining.

In questo capitolo, discuteremo di alcune caratteristiche che l'algoritmo scelto deve avere nel particolare dominio del Web Log Mining, quindi individueremo il perchè della scelta di un determinato algoritmo, e infine discuteremo delle scelte implementative.

4.2 Scelta dell'algoritmo

4.2.1 Caratteristiche

Applicare tecniche di mining di sequenze a quelle contenute in un dato Database di *Web access log*, consiste nello scoprire un insieme di item, come ad esempio page view web, condivise da un gran numero di visitatori. Per esempio, si consideri un database di log di accesso web a livello server, dove gli attributi richiesti rappresentano le page view di un generico utente e ogni record rappresenta un insieme di visite ordinate in base al tempo dell'utente rispetto ad un certo periodo. I pattern estratti sono le page view più frequentemente accedute lungo le visite degli utenti web. Per esempio:

La pagina del libro *The Data Webhouse Toolkit* viene visitata prima della pagina del libro *Data Mining Your Website* per il 71% delle volte nella stessa visita.

Il compito di trovare tutti i pattern sequenziali frequenti in un database di grandi dimensioni (*large database*) è abbastanza complesso.

Lo spazio di ricerca è potenzialmente dell'ordine di A^P , dove A è l'insieme degli item del database e P è la lunghezza del pattern d'uso più lungo (in termini di item) che può essere trovato. In siti Web molto noti, come ad esempio *Yahoo.com* e *amazon.com* con milioni di clienti e centinaia di migliaia di pagine web disponibili, la lunghezza delle sequenze cresce enormemente, incrementando lo spazio di ricerca esponenzialmente.

Dal momento che la dimensione dei Web log file cresce abbastanza rapidamente, i normali algoritmi possono essere non scalabili; inoltre, i contenuti della gran parte dei siti web cambia col tempo, rendendo alcune parti dei Web Log irrilevanti per l'analisi corrente. Anche gli obiettivi di analisi possono cambiare nel tempo con le esigenze di business.

Così, da una parte, c'è bisogno (forse sia in termini di memoria che di spazio disco) di miglioramenti in termini di scalabilità, e dall'altra parte, di introdurre restrizioni negli algoritmi di mining (tipo restrizioni di tempo, gerarchie di concetto), per scoprire conoscenza rilevante e corretta. In definitiva è necessario un'approccio scalabile e flessibile.

Un altro elemento da prendere in considerazione è il tipo di pattern sequenziali più adatti ai dati del clickStream.

In [18] vengono descritte delle prove sperimentali realizzate su reali dati d'u-

so. Gli autori descrivono un efficiente *framework* per Personalizzazione Web basata su pattern sequenziali. Su dati come web-log i loro risultati sperimentali mostrano che pattern più restrittivi, come pattern sequenziali contigui (per esempio cammini frequenti di navigazione), sono più adatti per task predittivi, tali come il prefetching (che riguarda il predire quale item è acceduto dal prossimo utente), mentre pattern con meno restrizioni, come itemset frequenti o generali pattern sequenziali, sono alternative più efficaci nel contesto di *Web Personalization e recommender system*.

Visto che in questa tesi, siamo più interessati ad applicazioni come Personalizzazione (siti web adattivi, sistemi di raccomandazione, customizzazione), Business Intelligence etc è risultato naturale orientarsi nella scelta verso quegli algoritmi che calcolano pattern sequenziali non strettamente contigui.

4.2.2 PrefixSpan con pseudo proiezione

Tanti sono in letteratura gli algoritmi proposti per il *sequential pattern mining*, molti fra i più importanti sono stati presi in considerazione in questa tesi.

La scelta di un particolare algoritmo è frutto di un attenta analisi e di un accurato confronto.

In questo capitolo, presentiamo un recente algoritmo che meglio risponde alle nostre esigenze e alle caratteristiche da noi precedentemente individuate, l'algoritmo scelto è: *PrefixSpan con pseudo proiezione*.

Per spiegare questa scelta rivediamo, in breve, gli altri algoritmi precedentemente esaminati e vediamo quali sono i loro limiti o le loro carenze per l'applicazione nel Web Log Mining.

Gli algoritmi *Apriori-like*, nonostante siano stati determinanti negli studi del sequential pattern mining, risultano avere dei grossi limiti per il campo d'uso in esame in questa tesi. Infatti, oltre al fatto che in generale questi algoritmi richiedono molte scansioni del database, essi, in presenza di database con sequenze di lunghezza rilevante tendono a generare un enorme insieme di potenze candidate e presentano particolare difficoltà nell'estrazione di *long sequential pattern* (Capitolo 3.1, paragrafo 3.4.3).

Per quanto riguarda *GFP1* e *GFP2*, abbiamo visto che sono algoritmi

proposti molto recentemente e che presentano la novità di trattare sia pattern sequenziali contigui che non. Come abbiamo avuto modo di constatare nel paragrafo precedente, questo nuovo approccio non è al momento di nostro interesse. Gli esperimenti [18], infatti, mostrano che nel nostro campo di interesse è più adatto prediligere pattern sequenziali non necessariamente contigui.

Spirit: nonostante le sue innovazioni in termini di restrizioni non risulta essere di particolare efficienza rispetto ad altri algoritmi come SPADE e PrefixSpan.

SPADE e *cSPADE*, in generale risultano essere degli algoritmi efficienti per l'estrazione di pattern sequenziali. Rispetto a *PrefixSpan*, essi utilizzano un formato dei dati da analizzare verticale (Capitolo 3, paragrafo 3.3.1) e questo richiede un notevole *overhead* di *preprocessing* impiegato nel convertire il database di input nel formato verticale ([17]).

Rimangono da esaminare gli altri algoritmi che, diciamo, appartengono alla famiglia di PrefixSpan.

FP-growth e *FreeSpan* risultano essere i primi studi di algoritmi basati su proiezione; PrefixSpan non solo segue l'idea base della proiezione ma con l'utilizzo dei postfissi ne migliora notevolmente l'efficienza.

SLPMiner presenta la grande novità di un supporto di restrizione decrescente (non fisso); questo è uno studio ancora in fase sperimentale e soprattutto non si conoscono i risultati di una sua applicazione in campo Web.

Prefix-Growth è molto simile a *PrefixSpan* con l'eccezione che in *Prefix-Growth* è possibile inserire vari tipi di restrizioni nel mining; la possibilità di arricchire il mining con una maggiore selettività quale le restrizioni al momento per semplicità non è stata presa in considerazione, si rimanda a futuri miglioramenti dell'algoritmo per aggiungere tipi di restrizione rilevanti al dominio in esame.

Adesso sintetizzando indichiamo quali sono le principali caratteristiche di *PrefixSpan* che ne hanno determinato la scelta.

PrefixSpan è uno fra i più efficienti algoritmi per il sequential pattern mining ed è anche in grado di trattare database di grande dimensioni.

Si noti che in PrefixSpan,

- non è necessaria nessuna generazione dei candidati, compito che appesantisce notevolmente gli algoritmi;
- il metodo di proiezione basata su prefisso fa sì che il database proiettato che verrà preso in esame nel passo successivo tende a diminuire la propria dimensione rendendo, di conseguenza, l'algoritmo più veloce; inoltre, non soltanto il database tende a restringersi, ma anche: solamente un sottoinsieme di pattern sequenziali diventa abbastanza lungo, così il numero delle sequenze nei database proiettati saranno abbastanza piccoli quando il prefisso cresce; la proiezione tiene conto solo della parte postfissa di una sequenza.

Come detto in precedenza, il costo maggiore pagato da Prefix è nella proiezione.

Utilizzando come soluzione la pseudo proiezione il risultato dell'algoritmo prescelto è ottimale: la pseudo proiezione assicura un basso uso di memoria e riduce il costo di creare una proiezione.

La scelta fatta, è quindi di realizzare *PrefixSpan con pseudo proiezione*; questa implementazione *main memory* dell'algoritmo sarà la *versione 1.0* e in futuro potrà facilmente essere portata *out of memory*.

In sintesi, *PrefixSpan* con la sua enorme flessibilità risponde in maniera esauriente alle nostre esigenze di efficacia e di efficienza.

4.3 Scelte implementative

Una volta scelto l'algoritmo da implementare possiamo ad analizzare quali caratteristiche devono avere i dati che verranno analizzati, quali sono le scelte che sono state fatte e alla fine i risultati ottenuti.

4.3.1 Dati di input

Nel Capitolo 3, paragrafo 3.2, abbiamo visto come viene definito formalmente un pattern sequenziale e tutti gli elementi necessari a comprendere tale

modello indipendentemente dal dominio in esame.

Adesso analizziamo questo modello riferendoci al dominio di nostro interesse, il dominio web.

Innanzitutto una sequenza è proprio una sessione utente e gli item sono le pagine accedute in quella sessione. Alcune cose cambiano.

Un *sequence database* S è un insieme di tuple $\langle sid, s \rangle$, dove

- sid è un *sequence_id*, nel nostro caso identificatore di sessione utente (è anche possibile considerare un *id* utente per cui la sequenza è data dagli insiemi di pagine visitate nel tempo)
- s è una sequenza di pagine visitate sequenzialmente in quella sessione utente, cioè una sequenza è una lista ordinata di coppie formate da un item (e non più di itemset) e un timestamp¹ (tempo di visita della pagina) in formule:
una sequenza s è denotata da $\langle s_1, s_2, s_3, \dots, s_l \rangle$ dove $s_j = (P_k, t_{j-1})$, e vale che $t_0 < t_1 < \dots < t_{l-1}$, per $P_k \in I$, $1 \leq j \leq l$, $k \in N$, $l \in N$.

A differenza del caso generale, un possibile esempio di dati nel caso Web potrebbe essere:

(x -sessione utente) $\Rightarrow P_1(t_{x,0}) P_2(t_{x,1}) P_4(t_{x,2}) P_3(t_{x,3})$

(y -sessione utente) $\Rightarrow P_2(t_{y,0}) P_5(t_{y,1}) P_3(t_{y,2})$

(z -sessione utente) $\Rightarrow P_1(t_{z,0}) P_4(t_{z,1})$

per esempio nella prima sessione utente sono state visitate le seguenti pagine:

P_1 al tempo $t_{x,0}$, P_2 al tempo $t_{x,1}$, P_4 al tempo $t_{x,2}$, P_3 al tempo $t_{x,3}$, dove $t_{x,0} < t_{x,1} < \dots < t_{x,n}$.

Supponendo che $\xi = 2$, gli ξ - pattern sono (P_2, P_3) e (P_1, P_4) .

Nel nostro caso di studio il problema di mining può essere riformulato così

nel seguente modo:

Data input:

¹Quando non è necessario accedere a questo campo, esso può essere anche omesso, così è stato fatto nei dati da noi considerati.

1. sequence database S della forma $\langle sid, s \rangle$ dove $sid = session_id$ e $s = \langle (P_{i1}, t_0), (P_{i2}, t_1) \dots (P_{in}, t_{n-1}) \rangle$, pagine visitate ad dato istante
2. supporto minimo ξ

Data output:

1. ogni ξ -pattern della forma $\alpha = (P_{i1}, P_{i2} \dots P_{in})$

Facciamo adesso alcune considerazioni.

Abbiamo discusso nel capitolo 2, nei paragrafi 2.3.3 e 2.3.1 di altre problematiche intensivamente studiate in altri lavori come tesi e progetti legate all'identificazione delle sessioni utente; quelle prese da noi in esame non sono state scelte indiscriminatamente ma attraverso un preprocessing sono state analizzate solamente quelle ritenute interessanti (per esempio le sessioni in cui si accede giornalmente alla pagina di Repubblica per essere aggiornati sono state scartate) e sono state utilizzate delle euristiche *time-oriented* (basate su tempo) per la loro individuazione.

Per semplicità e maggiore efficienza nel calcolo è stato scelto di identificare ogni sessione, ogni item, con un numero. Per fare questo si è cercato di dare delle stime tenendo conto anche del linguaggio di implementazione (*JDK1.4.1*).

Ad ogni sessione è stato associato un intero (*int 4 byte*) pensando che possa essere sufficiente. Per quanto riguarda le pagine accedute in una sessione utente, è difficile stimarle perchè questo numero cambia in base all'utente (è quindi un parametro), alla sessione in esame, etc; a questo campo è stato assegnato un intero (*int 4 byte*). Associamo univocamente ad ogni pagina un numero (essendo il numero di pagine in qualunque caso notevole utilizziamo un *Long*), quindi 8 byte per il campo.

I file contenenti i dati da analizzare saranno di due tipi: file binari e file di testo.

I primi avranno il seguente formato:

1. I campo contenente l'identificatore della sessione *sid*
2. II campo contenente n , cioè il numero degli elementi della sequenza s
3. III campo contenente la pagina visitata

4. IV campo contenente la pagina visitata

.....

e così via per ogni pagina visitata nella sessione.

La codifica sarà quindi :

<i>sid</i>	<i>n° pagine visitate</i>	<i>I pagina</i>	<i>II pagina</i>	<i>III pagina</i>	<i>...</i>	<i>ultima pagina</i>
------------	---------------------------	-----------------	------------------	-------------------	------------	----------------------

Supponendo che nella sessione utente x siano state accedute in ordine di visita le seguenti pagine P_3, P_1, P_6, P_4 la codifica sarà:

x	4	P_3	P_1	P_6	P_4
-----	---	-------	-------	-------	-------

Nel caso di file di testo, il formato sarà: *identificatore sessione* , *identificatore pagina*.

La tabella 4.1 rappresenta un esempio delle possibile righe di un file in questo caso.

<i>sessionid</i>	<i>urlid</i>
0	9461
0	7655
1	7655
1	1757
1	7655
2	1996
3	1821
4	9173
4	7008
4	9173
4	7008
4	4389
4	8667
...	...

Tabella 4.1: Dati di input utilizzando un file di testo.

4.3.2 Scelte di Progettazione e Implementazione

Il linguaggio usato per l'implementazione dell'algoritmo è *Java*; questa scelta deriva dal fatto che non solo Java è un potente linguaggio ad oggetti oggi molto noto e usato, ma anche perchè, il lavoro che seguirà quello fatto in questa tesi, è quello di adattare l'algoritmo implementato in un altro ambiente di sviluppo di data mining, il *KDDML*, già esistente, scritto in java (di questo ne parleremo meglio negli sviluppi futuri) e sviluppato presso il Dipartimento di Informatica di Pisa.

La grande mole di dati che vengono ogni giorno memorizzati porta sia alla necessità di una loro analisi automatica e sia alla ricerca e necessità di un calcolo sempre più efficiente.

Alla prima necessità risponde il KDD alla seconda rispondono tecniche come il parallelismo, e una tecnica nata recentemente l'**Hyper threading**².

Visto che il lavoro fatto in questa tesi verrà successivamente adattato in un sistema di mining notevolmente più complesso, risulta naturale pensare di non limitarsi ad uno studio basato su un calcolo uniprocessore ma avere la possibilità di un calcolo più efficiente anche in ambienti diversi, siano essi multiprocessori o con tecnologia *hyper threading*. Pertanto le scelte fatte sono state determinate dalla necessità di dare all'algoritmo una grossa flessibilità in base all'ambiente di esecuzione e alle scelte dei futuri utenti; rimandiamo alla descrizione delle classi per capire meglio quanto detto.

Ricordiamo qual è l'idea che sta alla base di PrefixSpan: analizzare il database dei dati, trovare gli item frequenti che costituiscono quindi i pattern di lunghezza 1 e proiettare per ognuno di essi il database di dati; il procedimento si riapplica ricorsivamente sui database ottenuti dalla proiezione accrescendo via via i pattern sequenziali trovati al passo precedente con gli item frequenti del passo in esame, il calcolo termina quando o nessun item

²Hyperthreading è una nuova tecnologia brevettata da Intel che fa "apparire" al sistema operativo un singolo processore come se in realtà si trattasse di due processori distinti. In pratica Hyperthreading mette a disposizione del sistema operativo due processori "logici" che sono in grado di operare simultaneamente, eseguendo task differenti e condividendo le stesse risorse hardware. Una soluzione di questo tipo ovviamente non offre le stesse prestazioni di un sistema a due processori, ma garantisce comunque un sostanziale incremento rispetto ad un sistema monoprocessore privo di questa tecnologia.

frequente è trovato o quando non esistono più dati da analizzare.

L'idea che sta alla base dell'algoritmo implementato è: prendere in ingresso il database da analizzare e attivare un thread (chiamiamolo il *thread0*) che facendo la prima scansione di questo database trova gli item frequenti. Per ognuno di essi, una volta fatta la proiezione del database originale in base all'item in esame, questo thread controlla se è possibile attivarne un altro a cui affidare il calcolo; nel caso contrario richiama una funzione ricorsiva (*span*) per analizzare questa partizione di dati. Si noti che il numero massimo di thread attivabili è un parametro di ingresso.

In questa funzione ricorsiva è possibile, a sua volta, dopo aver analizzato i dati e trovato gli item frequenti, attivare, (se consentito) un altro thread (appartenente sempre alla stessa classe di thread) o richiamare nuovamente la funzione ricorsiva per continuare l'analisi sui nuovi dati.

Esiste un altro caso: se le dimensioni del database proiettato non superano un certo numero di sessioni (anche questo valore è parametrico) oppure il numero dei thread attivi consentito è 1 (quindi solo il *thread0*) l'intero calcolo viene affidato ad una seconda funzione ricorsiva (*onlySpan*).

In sintesi, l'attivazione di un dato thread e quindi il suo overhead devono essere motivati da un certo numero di sessioni da analizzare oltre che dal numero prefissato di thread attivabili, questo ovviamente per apprezzare i benefici del multithreading.

L'algoritmo scelto è un algoritmo main-memory, pertanto si è cercato di ridurre al massimo l'occupazione di memoria.

Le classi realizzate sono in tutto sei, passiamo alla descrizione di ciascuna.

PrefixSpan

Questa classe è quella che contiene il *main* e che attiva la classe principale *PrefixSpanThread*. Riceve i seguenti parametri di ingresso:

1. (*args[0]*) soglia di minimo supporto (es. 40%)
2. (*args[1]*) file di input da cui prelevare i dati
3. (*args[2]*) file di output su cui memorizzare i risultati
4. (*args[3]*) numero massimo di thread attivabili

5. (*args[4]*) numero minimo di sessioni in un database per cui è concesso attivare un thread.

I primi tre parametri sono semplici da comprendere.

Il terzo parametro corrispondente al numero massimo di thread attivabili. Visto che l'algoritmo implementato utilizza un approccio *divide et impera*, è stata prevista la possibilità di attivare ad altri livelli dei thread a cui affidare porzioni di calcolo. Ovviamente per ottenere un buon risultato in termini di efficienze e maggiore flessibilità, questo numero dei thread è stato pensato come parametro; nel caso in cui esso venga omissso, l'intero calcolo è affidato ad un solo thread (in tale caso non va indicato neanche il quarto parametro indicante il numero minimo di sessioni per cui è consentito attivare un thread); in questo caso l'unico thread attivo è quello attivato dal main.

Una volta recuperati i parametri di ingresso e attivato il thread principale, l'oggetto in esame si sospende attendendo la terminazione del thread principale; questo avviene eseguendo un particolare metodo messo a disposizione dalla classe Thread java *nomeThread.join()*.

Dopo la terminazione del thread principale e quindi dopo averne ottenuto il risultato, si procede a stampare i pattern sequenziali trovati sul file di output.

Vista la grande flessibilità offerta dal linguaggio XML e la sua crescente diffusione in rete è stato scelto tale formato per la memorizzazione dei risultati ottenuti. Infatti grazie all'XML è possibile memorizzare, scambiare e presentare una grande quantità di informazione. Una dei punti di forza di questo linguaggio di mark-up è rappresentato dalla sua flessibilità di rappresentare diversi tipi di informazione provenienti da fonti diverse, facilitando così l'integrazione dei dati su Internet e offrendo un'altra notevole possibilità: "trasformare il Web in un database semi-strutturato". I documenti XML possono essere considerati come sorgenti di dati semi-strutturati. A questa motivazione va aggiunto il fatto che anche il KDDML, ambiente di sviluppo di applicazioni di data mining in cui verrà inserito opportunamente Prefix-Span, utilizza file in formato XML.

Il metodo utilizzato è il seguente:

```
public void esportaXML(isVector allSP,String fileOut,Database db,String  
min_sup,isVector itemPresenti)
```

I parametri rappresentano rispettivamente:

1. il vettore contenenti tutti i pattern sequenziali trovati; *allSP* è un'istanza della classe *isVector* (estensione della classe java *Vector*);
2. *fileOut*, nome del file xml su cui memorizzare i risultati ottenuti;
3. un oggetto della classe *database* in cui sono memorizzati i dati di input;
4. un'altra istanza della classe *isVector* contenente tutti gli item distinti presenti nei dati di input.

Nel file xml, oltre ai pattern sequenziali trovati (contenuti in una istanza della classe *isVector allSP*), e al loro specifico supporto (*min_sup*), vengono visualizzate altre caratteristiche dei dati analizzati come numero di sequenze esaminate, numero medio di item per sequenza, numero degli item distinti presenti nel database etc.

PrefixSpanThread

Rappresenta la classe principale del sistema e contiene il cuore dell'intero calcolo.

Ecco il costruttore di PrefixSpanThread:

```
public PrefixSpanThread22(boolean OKonlySpan, SP pattern, int min_sup,
    Database db, isVector myAllSP, isVector itemFrequenti, numTotThread
    countThread, int maxSession)
```

I seguenti parametri rappresentano:

1. la variabile booleana *OKonlySpan* indica se devo affidare il calcolo interamente alla funzione ricorsiva *onlySpan*, in caso contrario è concesso attivare altri thread;
2. L'istanza della classe *SP* rappresenta il pattern sequenziale in esame, per cui verrà fatta la proiezione del database e la conseguente analisi.
3. *min_sup* è il minimo supporto.
4. *db*, istanza della classe *database*, rappresenta i dati da analizzare per il pattern in esame.
5. In *myAllSP* verranno memorizzati i pattern sequenziali trovati.

6. In *itemFrequenti* vengono memorizzati gli item frequenti del passo precedente da prendere in considerazione; infatti se un item è infrequente al generico passo i -esimo, a maggiore ragione sarà infrequente al passo $i+1$ -esimo visto che il database in esame è una riduzione di quello del passo precedente.
7. Un oggetto della classe *numTotThread* permette di accedere in maniera sincronizzata alla variabile indicante il numero massimo di thread attivabili; quando si vuole attivare un thread si controlla questa variabile, è una sorta di P e V .
8. La variabile *maxSessioni* come detto prima indica il minimo numero di sessioni per cui è consentito attivare un thread.

Una volta attivata una generica istanza di questa classe si procede alla ricerca degli item frequenti richiamando il metodo :

```
public isVector cercaItem(isVector itemFrequenti, SP pattern, Database db,  
    int min_sup)
```

Questo metodo scandisce il database da proiettare analizzando solo gli item frequenti del passo precedente (cioè quelli contenuti in *itemFrequenti*) e calcolandone il supporto; restituisce un vettore di due elementi: il vettore contenente gli item attualmente frequenti e il vettore contenente il corrispondente supporto.

Per tutti gli item frequenti trovati viene creato il corrispondente database proiettato, richiamando uno dei due costruttori della classe database, e viene fatta una scelta fra le seguenti possibilità:

- attivare un thread istanziando un altro elemento della classe;
- richiamare la procedura ricorsiva *spann*
- richiamare un'altra procedura ricorsiva *onlySpan*

Questa scelta viene determinata tenendo conto, in ordine:

1. prima delle dimensioni del database da esaminare, se questa è inferiore al parametro di ingresso *maxSessioni* allora viene attivata una semplice funzione ricorsiva che continua l'analisi dei dati richiamando se stessa all'interno, fino ad esaurire l'analisi di tutte le sessioni (il metodo è *onlySpan*);

2. se la dimensione dei dati è abbastanza grande (notiamo che essendo *maxSessioni* un parametro siamo noi a decidere questo “abbastanza”), vuol dire che ha senso attivare un thread per il calcolo; quindi se il numero dei thread attivabili è inferiore a quello consentito, istanziamo un altro thread.
3. altrimenti richiamiamo un'altra procedura ricorsiva *span*; quest'ultima, effettua un'ulteriore analisi dei dati, diciamo una passata, e se è necessario fare ulteriori passate, sceglie se procedere con la ricorsione o attivare un Thread (va a testare il valore della classe *countThread* che tiene traccia dei thread attivi, accesso sincronizzato).

In sintesi, mentre la procedura *span* oltre alla ricorsione consente di attivare anche un thread, la procedura *onlySpan* affida completamente il calcolo alla ricorsione.

Ecco le rispettive interfacce:

```
public void spann(SP pattern, int min_sup, Database db, isVector myAllSP,
                 isVector itemFrequenti, int maxSession)
```

```
public void onlySpann(SP pattern, int min_sup, Database db, isVector
                     myAllSP, isVector itemFrequenti)
```

Questa necessità di differenziare i casi di calcolo nasce dall'esigenza di dare a *PrefixSpan* una certa flessibilità nell'adattarsi alla quantità di dati elaborati e alla particolare architettura sulla quale verrà eseguito.

Una volta fatti partire la ricorsione e/o i thread per esaminare i database proiettati più piccoli, il thread diciamo padre aspetta l'esecuzione dei thread figli eseguendo una *join()* come nel caso del *main*.

Database

Questa classe si occupa di gestire tutto ciò che riguarda i dati da analizzare. L'idea base di *PrefixSpan* è quella di analizzare un database, ridurlo in base al pattern sequenziale trovato e continuare a processare il nuovo database per accrescere il pattern. Pertanto, in questa classe, così come indicato dai due costruttori, individuiamo due tipi di database su cui lavorare; il database originale e i database proiettati.

Il costruttore che crea il database originale è il seguente:

```
public Database (String nomeFile) throws Exception
```

Il costruttore che crea i database proiettati è il seguente:

```
public Database (SP p, Database db)
```

Il database originale viene generato caricando i dati o da un file binario, utilizzando la procedura *load_orizzontal_binary*, o da un file di testo, utilizzando la procedura *load_vertical_ascii* (in base all'estensione del file passato come parametro). Ogni sessione letta da file viene memorizzata in una istanza della classe *sessione* e aggiunta al vettore *sequenze*, campo della classe *Database*.

Il costruttore dei database proiettati, prende in ingresso il pattern (oggetto della classe *SP*), in base a cui deve essere fatta la proiezione, e il database da proiettare *db*. All'interno del costruttore la proiezione viene fatta richiamando il metodo *proiezione(SP p, Database db)*; la ricerca avviene in questo modo: in ogni sessione del database si va a ricercare il primo item del pattern in esame, successivamente da quell'indice in poi si va alla ricerca dell'item successivo e così via fino quando non si trova anche l'ultimo item e l'indice in cui questi occorre, a questo punto è proprio l'indice successivo a quello dell'ultimo item del pattern che viene restituito e viene memorizzato nel campo *offset*; questo implica che la prossima analisi partirà ovviamente dall'indice memorizzato in *offset*. In sintesi, mentre nell'idea originale della pseudo-proiezione veniva utilizzato un puntatore alla sequenza e l'offset al suo interno, visto che Java non ha puntatori, abbiamo pensato di risolvere il problema utilizzando il vettore *offset*; esso contiene nella posizione *i*-esima l'offset della sessione memorizzata a sua volta nella posizione *i*-esima del vettore *sequenze*.

Ovviamente l'istanza che rappresenta il database originale avrà i campi *prefisso* e *offset* uguali a null.

numTotThread

Questa classe permette di accedere in maniera sincronizzata alla variabile che indica il numero massimo di thread attivabili; se è possibile attivare un thread la variabile viene incrementata. La classe contiene al suo interno due campi: *value* e *MAX_THREAD_AVAILABLE* che rappresentano rispettivamente il numero di thread al momento attivi e il numero massimo

di thread attivabili consentito, quest'ultimo valore viene passato al momento della creazione dell'oggetto:

```
public numTotThread(int count)
```

Attraverso il metodo sincronizzato *isAvailable()* si viene a conoscenza se è possibile o no attivare un thread, in caso positivo il campo *count* viene incrementato.

Corrispondentemente, l'ultima istruzione eseguita da un thread è richiamare il metodo *V* che decrementa il campo *count*.

isVector

Questa classe estende la classe di Java *Vector* e fornisce dei metodi per la memorizzazione dei pattern sequenziali all'interno del vettore e degli item in generale.

Nel caso dei pattern sequenziali, i metodi *public int get_lenghtIndex(Object chiave)* e *public void addSP(SP pattern)* permettono di inserire i pattern in base alla loro lunghezza consentendo quindi di recuperarli tramite una ricerca binaria, eliminando sia i pattern ripetuti che quelli contenuti. Anche nel caso degli item, viene fatta memorizzazione e ricerca binaria.

Sessione

La classe si occupa di mantenere le informazioni relative a ciascuna sessione. Il costruttore della classe è il seguente:

```
public Sessione (int sid, long[] pag)
```

I due parametri rappresentano rispettivamente l'identificatore della sessione e le pagine accedute sequenzialmente che vengono assegnati rispettivamente ai campi privati *sid* e l'array di long *pag*. Contiene vari metodi per accedere a queste informazioni.

SP

Questa classe mantiene tutte le informazioni relative ai pattern sequenziali. Contiene i seguenti campi *private Vector sp* e *private int sup* dove *sp* contiene gli item appartenenti al pattern e *sup* memorizza il suo supporto.

4.3.3 Test ed esperimenti

In questo paragrafo discutiamo dei test applicati a PrefixSpan, implementato come descritto nel paragrafo precedente, e dei risultati ottenuti.

I test sono stati eseguiti su un Intel Pentium 4 CPU 2.00GHz con 1.00G di RAM, 1024 MB di memoria virtuale e Microsoft Windows XP Professional version 2002.

I file contenenti i dati analizzati sono file di testo e rispettano il formato discusso nel paragrafo 4.3.1; la tabella 4.2 ne riassume le caratteristiche. I file *Nasa* e *Berkeley* derivano dalla stessa fonte di dati dove l'identificazione delle sessioni è stata ottenuta applicando diversi timeout.

File di input	Numero di sessioni	Dimensioni (KB)
nasa5m	389352	37.830
nasa15m	320687	37.639
nasa30m	296222	37.549
berkley5m	405968	42.177
berkeley15m	350638	42.082
berkeley30m	320251	42.018
clickworld30m	1928087	104.878

Tabella 4.2: Dati analizzati

I grafici che seguono rappresentano i test effettuati su PrefixSpan al variare del supporto, dei file di input e fissando ad 1 il numero dei thread attivi. Questo caso corrisponde all'attivazione di un solo thread e quindi all'esecuzione dell'intero calcolo da parte della procedura interamente ricorsiva (*onlySpan*); questa ovviamente è la soluzione più efficiente nel nostro caso viste le caratteristiche della macchina su cui lavoriamo.

Il primo grafico (4.1) rappresenta l'esecuzione di PrefixSpan sui dati contenuti nei file nasa.

Per una migliore comprensione, abbiamo suddiviso il grafico precedente in due grafici, il primo (figura 4.2) guarda al comportamento di PrefixSpan per valori bassi di supporto, il secondo (figura 4.3) per valori alti.

La stessa cosa è stata fatta per i dati contenuti nei file berkeley; il grafico generale (figura 4.4) è stato spezzato in altri due, ripettivamente figura 4.5 e figura 4.6.

Si noti che l'algoritmo per i file derivati dagli stessi dati, cioè i vari nasa5m,

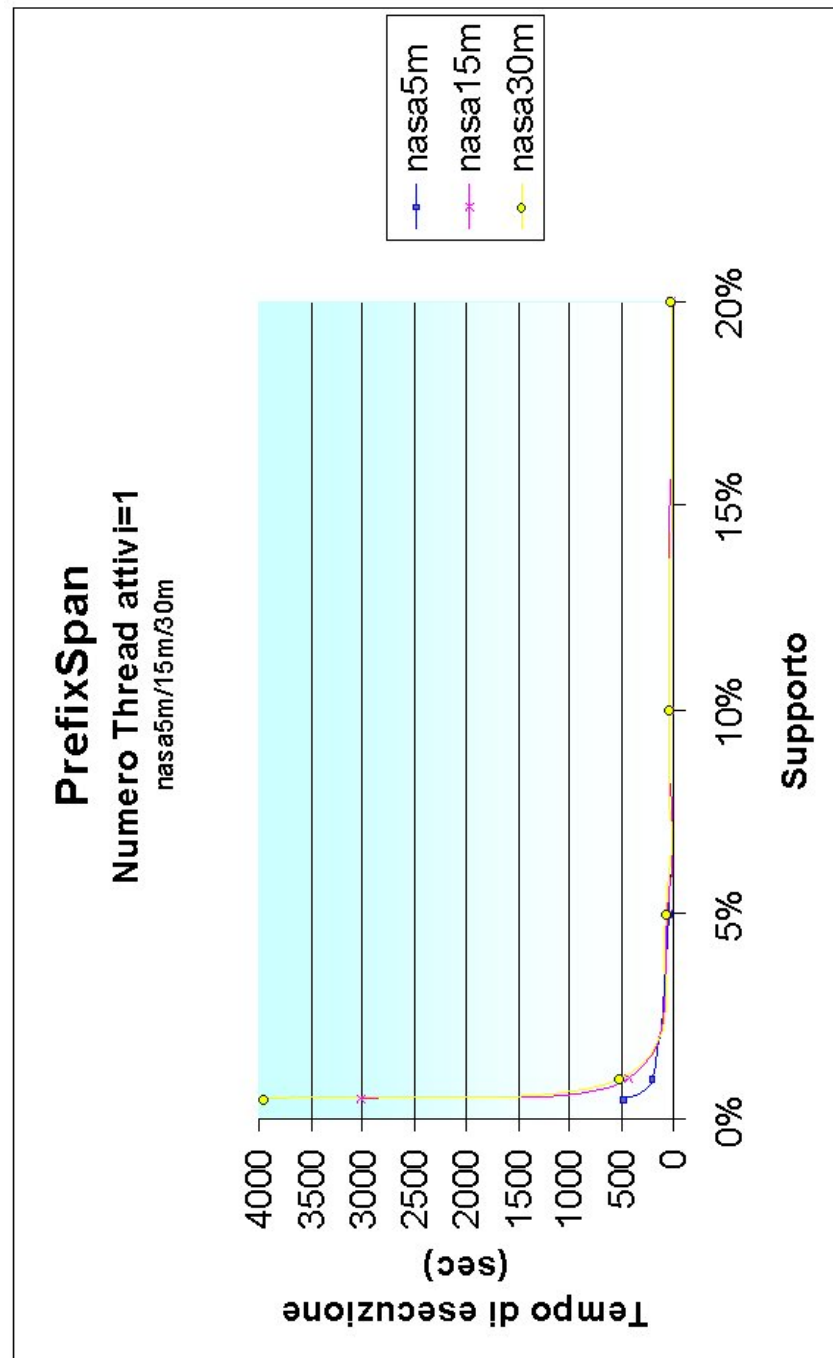


Figura 4.1: Tempo di esecuzione di PrefixSpan al variare del supporto e dei file di input.

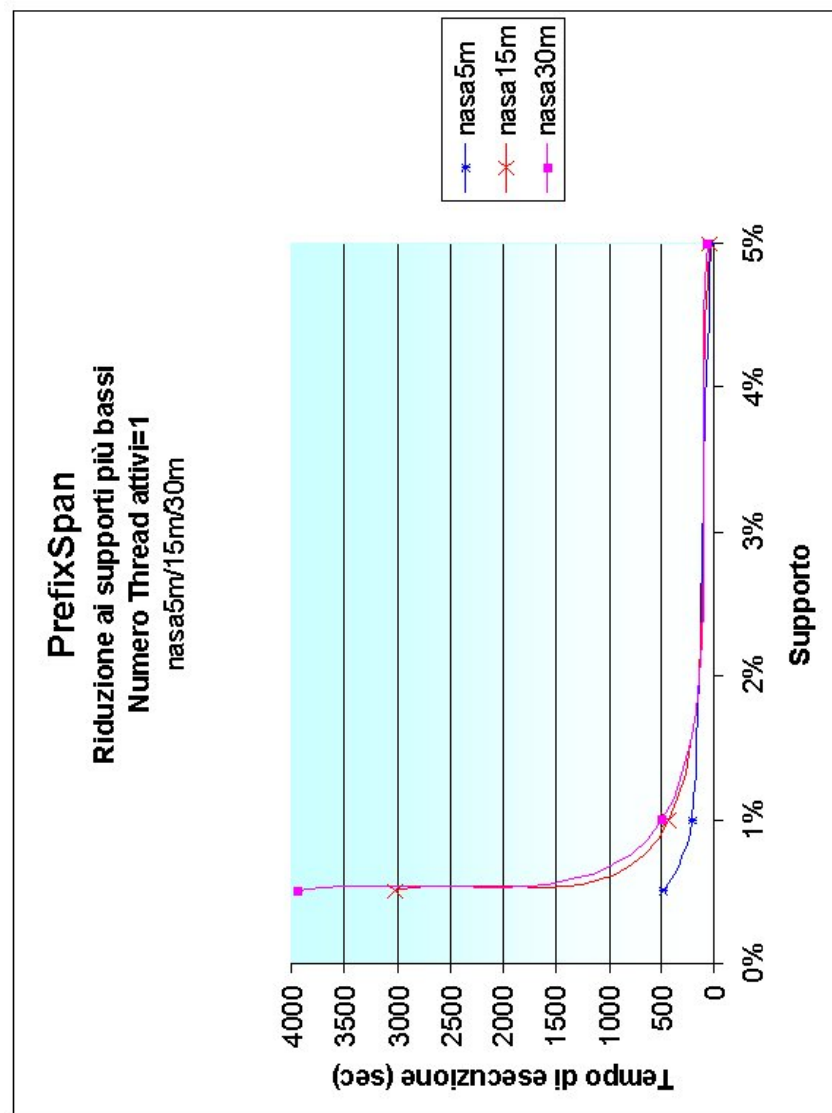


Figura 4.2: Tempo di esecuzione di PrefixSpan al variare del supporto e dei file di input.

nasa15,etc e berkeley5m, berkeley15 etc, ha comportamenti molto simili soprattutto per valori alti del supporto.

Nel grafico 4.7 si confronta l'esecuzione di PrefixSpan su i file nasa30m, berkeley30m e clickworld30m. In questo caso, ben si evidenziano le diverse caratteristiche dei file di input, sia in termini di dimensioni che nella natura

dei dati; per esempio in clickworld gli item sono meno densi rispetto agli altri due file, questo è legato alle dimensioni dei dati da analizzare.

Il grafico della figura 4.8 rappresenta, invece, l'occupazione di memoria di PrefixSpan sui dati contenuti nel file nasa30m, berkeley30m e clickworld30m al variare del supporto. Ovviamente per file di dimensioni maggiori viene occupata un maggior quantità di memoria.

Il grafico della figura 4.9 rappresenta l'esecuzione di PrefixSpan su *Tera*, una macchina del dipartimento di Pisa che ha due processori e che gode della tecnica *hyperThreading*.

Si può subito notare che ovviamente l'attivazione di un unico thread per l'intero calcolo è il caso più lento.

Si nota dal grafico che con un numero maggiore di thread maggiore si ha un'efficienza maggiore. In generale si noti che i fattori determinanti sono due: da una parte il numero di thread attivabili a disposizione dall'altra il limite inferiore al numero di sessioni consentito per attivare un nuovo thread; per semplicità li indichiamo rispettivamente con *maxThreadAttivabili* e *maxSessioni*.

Nel caso in cui $\text{maxThreadAttivabili} = 2$ l'esecuzione più veloce è per $\text{maxSessioni} = 1000$; una spiegazione a questo fenomeno è che per maxSessioni uguale 100 ci sono troppe commutazioni di contesto non giustificate da tempi di elaborazioni (in questo caso la dimensione del database proiettato da esaminare è troppo piccolo per giustificare l'overhead del thread attivato); invece per $\text{maxSessioni} = 10000$ ci sono numerosi intervalli di tempo in cui non è possibile attivare un thread pur avendo un processore a disposizione. Per esempio, supponiamo che il thread principale abbia trovato nel database originale 10 item; attiva un thread (thread1) per esaminare il primo item e i relativi dati mentre lui affida il calcolo relativo al secondo item alla sua funzione ricorsiva *spann*; sia il thread1 che thread0 non riescono ad attivare altri thread perchè sono inibiti dal valore di *maxThreadAttivabili*. Potrebbe succedere che nel momento in cui il thread0 inizia ad elaborare database di dimensioni inferiori a 10000 affidando l'esecuzione ad un'altra funzione ricorsiva *onlySpann* (la quale non testa più se è possibile attivare altri thread), il thread1 possa terminare: in questo caso il thread0 elabora su di un processore mentre l'altro è libero, bisogna attendere necessariamente la fine dell'esecuzione di *onlySpann* affinché il thread0 per elaborare altri item attivi un altro thread. In sintesi, per periodi troppo lunghi il valore di *maxSessioni* troppo

grande lascia non utilizzato un processore.

Nel caso di `maxThreadAttivabili=3` l'esecuzione più veloce è per `maxSessioni = 100`; una spiegazione a questo fenomeno è che in questo caso, a differenza di quello appena spiegato, i processori a disposizione per il calcolo rimangono per meno tempo non inutilizzati, in quanto se dei tre thread attivi uno termina la sua esecuzione, essendo `maxSessioni` un valore basso è più probabile che uno dei due thread ancora possa attivarne un altro.

Nel caso di `maxThreadAttivabili=4` l'esecuzione più veloce è per `maxSessioni = 10000`; una giustificazione a questo fenomeno è che l'accesso sincronizzato ad una variabile condivisa comporta un overhead maggiore all'aumentare dei thread che vi accedono (la variabile è `maxThreadAttivabili`), e inoltre, la possibilità che un processore libero non venga inutilizzato è limitato dal numero superiore di thread attivi.

Corrispondenze dei Pattern Sequenziali trovati

Per una maggiore comprensione, i pattern sequenziali ottenuti eseguendo `PrefixSpan` con pseudo proiezione sui dati contenuti nei file `nasa5m`, `nasa15m` e `nasa30m` sono stati convertiti nelle corrispondenti url. Precisamente la corrispondenza è stata fatta sui dati nasa limitatamente solo ai file html (diciamo file nasa ridotti), sono stati cioè ripuliti da immagine o altro che non era di nostro interesse.

Le tabelle 4.4, 4.5 e 4.3 presentano tre pattern sequenziali trovati e le relative corrispondenze tra *url* e numeri. Per esempio, la tabella 4.3 mostra uno dei pattern trovati in cui, in generale, si può notare che un utente che accede alle pagine relative alla missione *apollo* accede anche alle pagine relative alle missioni *gemini* e *mercury*.

Confronti con Clementine

Una volta realizzato `PrefixSpan`, alcuni risultati ottenuti sono stati confrontati con quelli di un altro sistema di data mining *Clementine* (realizzato da SSPS [3]).

Su *Clementine* sono stati analizzati i dati dei seguenti file (i test su Cle-

SP:	$\langle 6095, 2738, 1219 \rangle$
numero=1672	url= www.nasa.org/history/mercury/mercury.html
numero=4637	url= www.nasa.org/history/gemini/gemini.html
numero=1863	url= www.nasa.org/history/apollo/apollo.html

Tabella 4.3: Pattern sequenziale e relativa corrispondenza

SP:	$\langle 58, 2006, 1631, 1005, 7739 \rangle$
numero=58	url= www.nasa.org/shuttle/missions/sts-69/mission-sts-69.html
numero=2006	url= www.nasa.org/shuttle/missions/sts-73/mission-sts-73.html
numero=1631	url= www.nasa.org/shuttle/missions/sts-74/mission-sts-74.html
numero=1005	url= www.nasa.org/shuttle/missions/sts-72/mission-sts-72.html
numero=7739	url= www.nasa.org/shuttle/missions/sts-75/mission-sts-75.html

Tabella 4.4: Pattern sequenziale e relativa corrispondenza

mentine sono stati eseguiti su un Intel pentium 4 CPU 2.00GHZ, 512 Mb RAM):

- nasa5m. Tempo esecuzione Clementine 19'52" (supporto 10%).
- nasa15m. Tempo esecuzione Clementine 47'30" (supporto 10%).
- berkeley15m. Tempo esecuzione Clementine 1'40" (supporto 10%).
- clickworld30m. Esecuzione fallita, spazio non sufficiente.

I pattern sequenziali trovati sono uguali, i tempi di PrefixSpan sono invece:

- nasa5m. Tempo esecuzione PrefixSpan 26" (supporto 10%).
- nasa15m. Tempo esecuzione PrefixSpan 32" (supporto 10%).

SP:	$\langle 1672, 4637, 1863, 1996 \rangle$
numero=1672	url= www.nasa.org/history/apollo/apollo-8/apollo-8.html
numero=4637	url= www.nasa.org/history/apollo/apollo-9/apollo-9.html
numero=1863	url= www.nasa.org/history/apollo/apollo-10/apollo-10.html
numero=1996	url= www.nasa.org/history/apollo/apollo-11/apollo-11.html

Tabella 4.5: Pattern sequenziale e relativa corrispondenza

- berkeley15m. Tempo esecuzione PrefixSpan 1'48" (supporto 10%).
- clickworld30m. Tempo esecuzione PrefixSpan 9'33" (supporto 5%).

Pur tenendo conto di una maggior quantità di RAM utilizzata per i test su PrefixSpan, appare evidente la maggiore efficienza di quest'ultimo rispetto a Clementine.

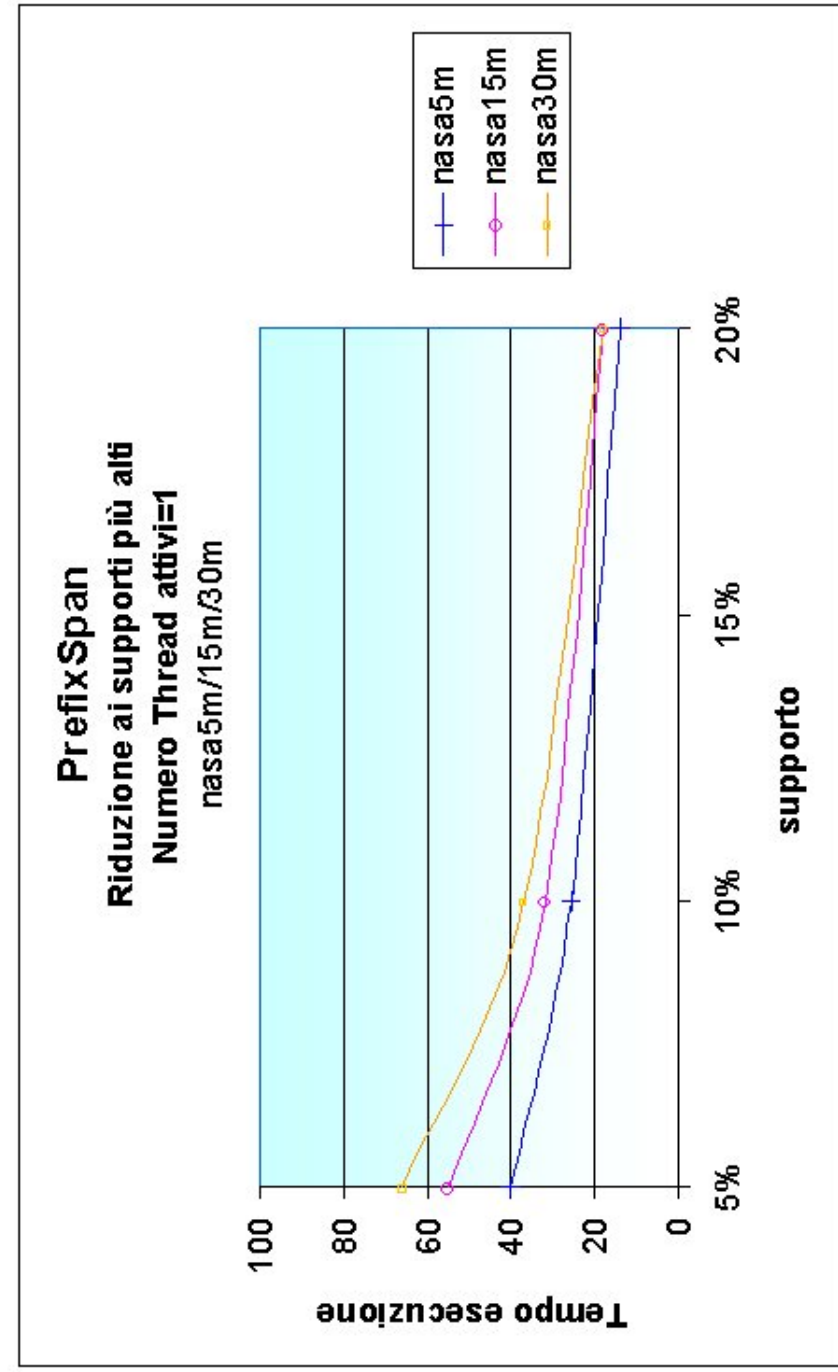


Figura 4.3: Tempo di esecuzione di PrefixSpan al variare del supporto e dei file di input.

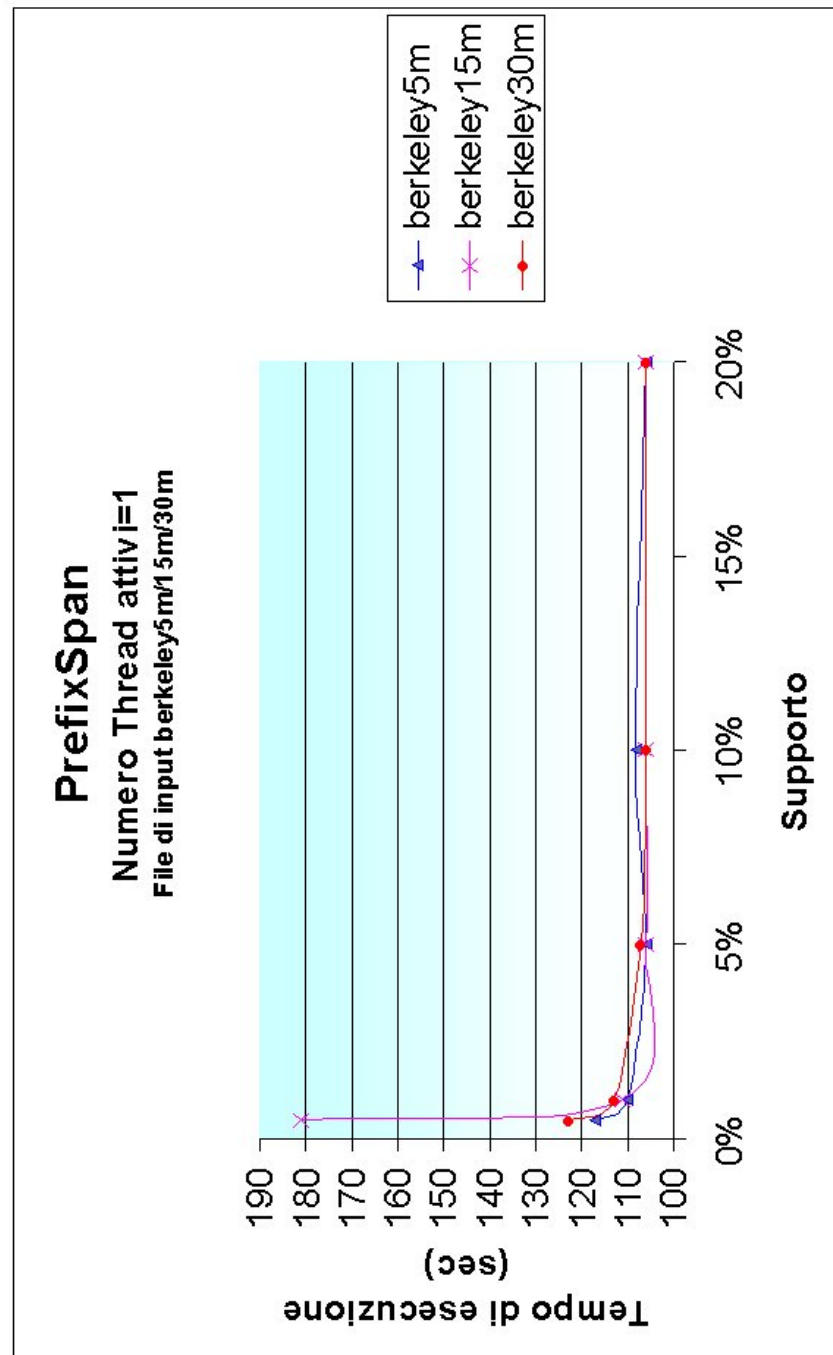


Figura 4.4: Tempo di esecuzione di PrefixSpan al variare del supporto e dei file di input.

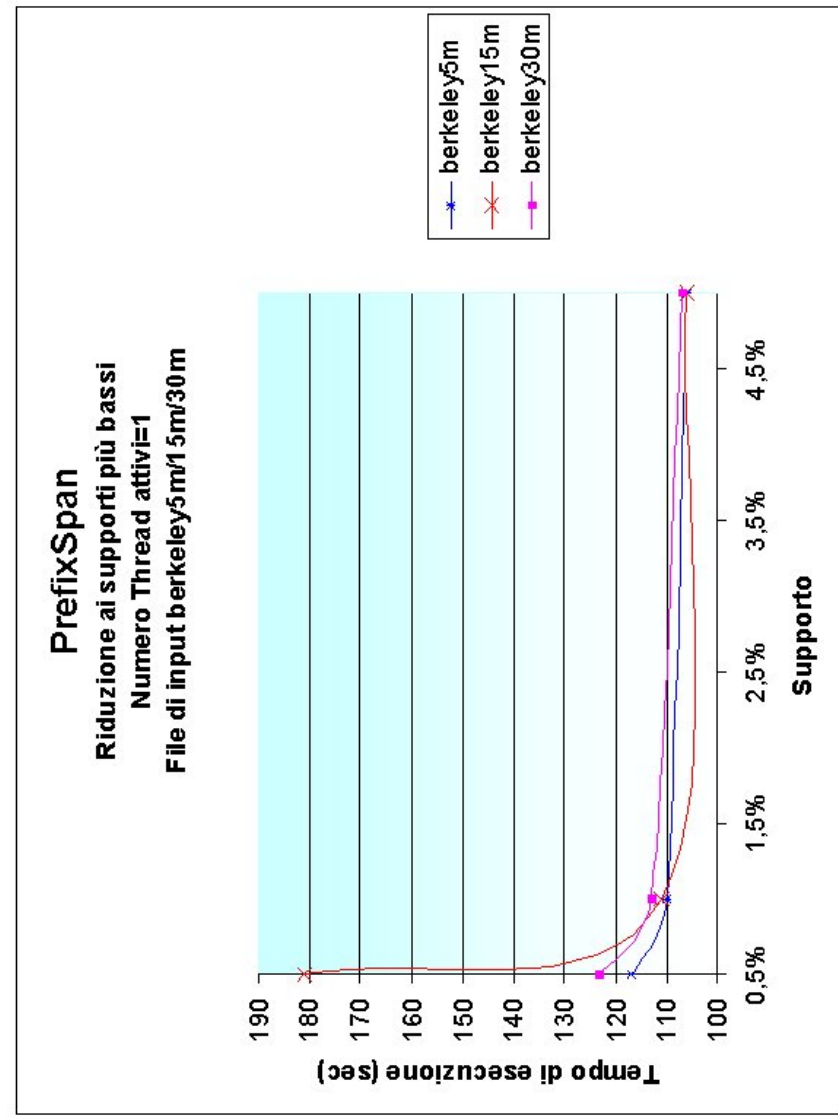


Figura 4.5: Tempo di esecuzione di PrefixSpan al variare del supporto e dei file di input.

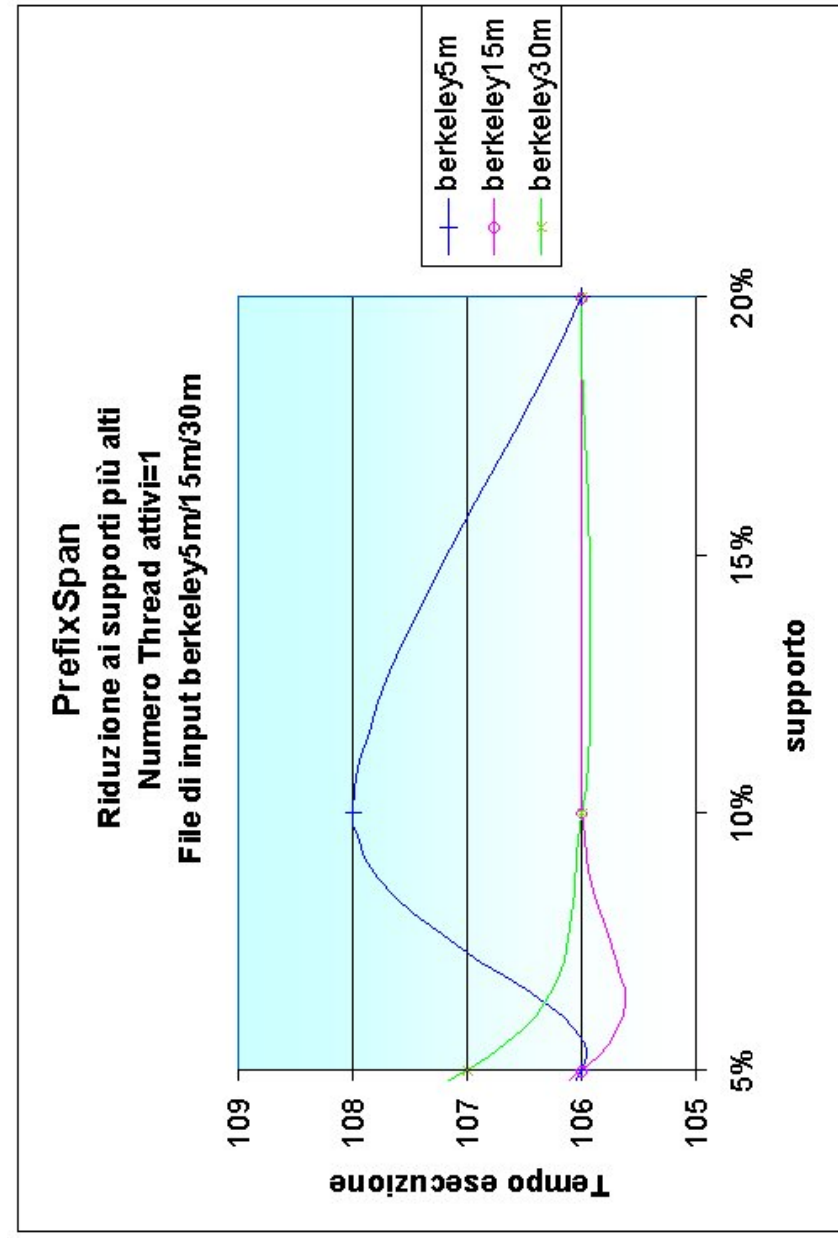


Figura 4.6: Tempo di esecuzione di PrefixSpan al variare del supporto e dei file di input.

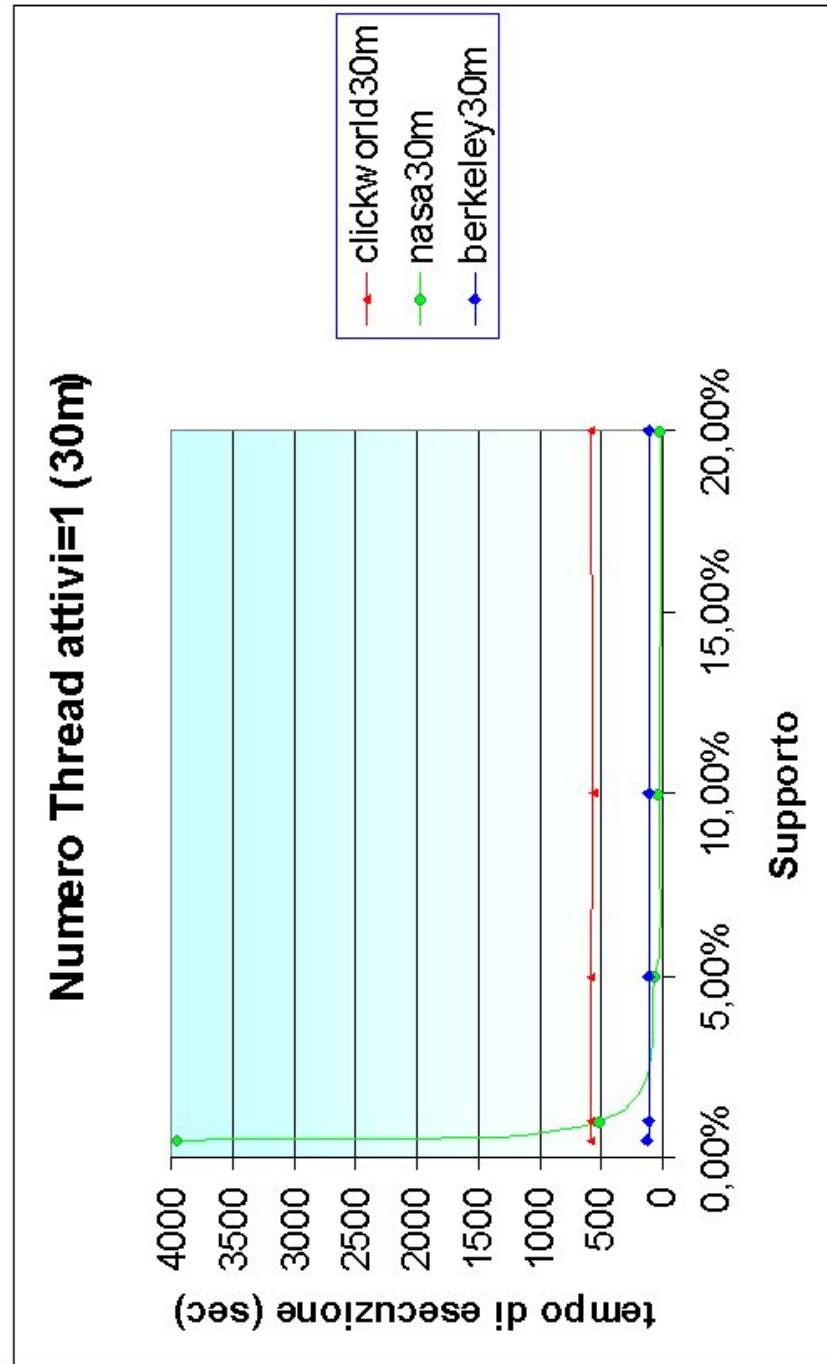


Figura 4.7: Tempo di esecuzione di PrefixSpan al variare del supporto e dei file di input.

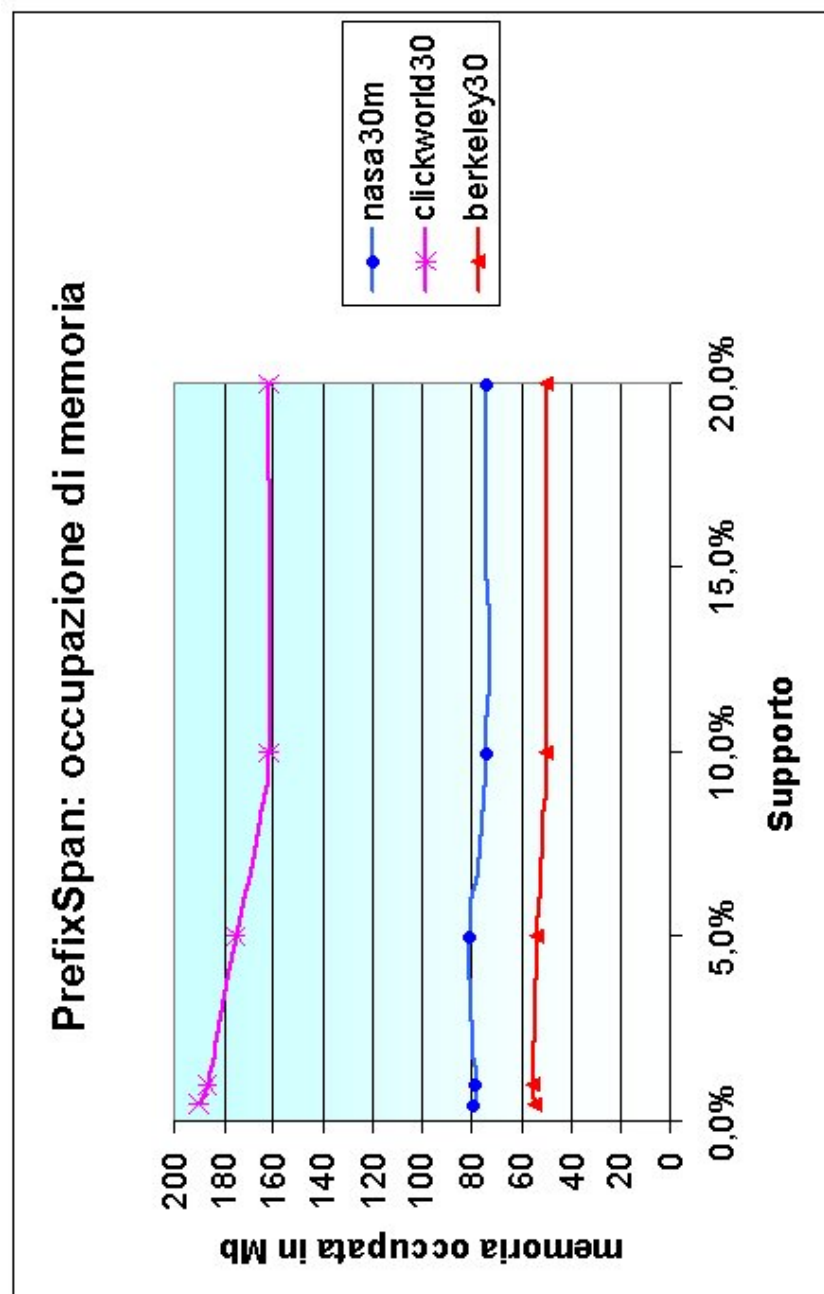
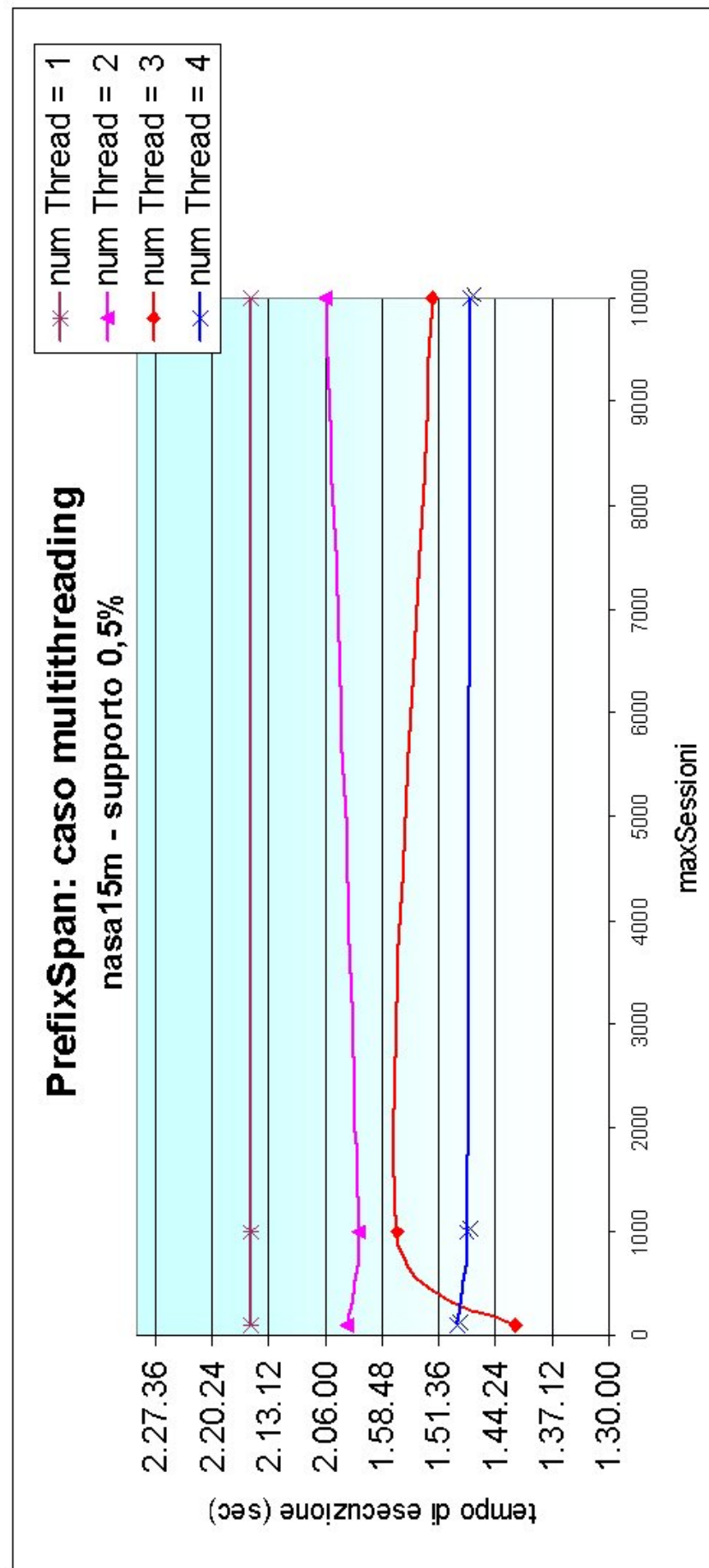


Figura 4.8: PrefixSpan: occupazione di memoria al variare del supporto e dei file di input.



Capitolo 5

Conclusioni e Sviluppi Futuri

Le informazioni contenute nel web sono di vasta portata e tendono a crescere di giorno in giorno; il numero di visitatori così come il numero di siti cresce altrettanto velocemente. Non sorprende la necessità di dotarsi di strumenti sofisticati per l'analisi di accessi al web al fine di migliorare l'organizzazione delle informazioni contenute in tali siti e l'accessibilità.

In questa tesi, viene presentato uno dei possibili strumenti per questa analisi.

5.1 Conclusioni

Il lavoro che abbiamo svolto può essere suddiviso in più parti.

La prima parte si è concentrata sullo studio del dominio di applicazione in esame, il dominio web, e in particolare su studi e problematiche di un'area di ricerca il *Web Usage Mining*.

La seconda parte riguarda lo studio e l'analisi di un recente modello di data mining: i *pattern sequenziali*.

E' stata svolta una prima fase di ricerca in letteratura del lavoro sviluppato da altri ricercatori sullo argomento.

A questo è seguito un'attenta analisi di tale modello indipendentemente del dominio di applicazione; abbiamo quindi studiato cosa sono i pattern sequenziali, quali sono le peculiarità, a cosa servono.... Una volta individuata la loro natura si è passati allo studio dei vari algoritmi di calcolo proposti, dai primi algoritmi più semplici a quelli più recenti e complessi. L'analisi di tali algo-

ritmi ha portato all'individuazione di caratteristiche comuni e particolarità di alcuni, e una loro classificazione da noi realizzata in base a questi elementi.

La terza parte, si basa sulla fusione degli studi ottenuti dalle parti precedenti: la scelta di un particolare algoritmo di estrazione di pattern sequenziali (sequential pattern mining) le cui caratteristiche soddisfano le esigenze del dominio web.

E' stato così implementato un nuovo ed efficiente algoritmo per il mining di pattern sequenziali da dati di accesso web: **PrefixSpan con pseudo proiezione**.

L'algoritmo scelto è un algoritmo main memory e l'implementazione fatta risponde anche ad esigenze di parallelismo e hyperThreading.

Sono stati fatti dei test su *PrefixSpan*; il confronto con altri sistemi adibiti allo stesso scopo (Clementine) evidenziano la sua superiorità in termini di efficienza.

Infine sono stati fatti degli esperimenti analizzando il comportamento di questo algoritmo su una macchina multiprocessore con hyperThreading. I risultati ottenuti mostrano la correttezza delle scelte fatte.

5.2 Sviluppi Futuri

L'uso del web come mezzo per attività commerciali è aumentato molto nel recentemente. Di pari passo con l'aumento dell'attenzione da parte degli uomini d'affari è aumentato anche lo sforzo dei ricercatori per realizzare strumenti sempre più sofisticati nel tentativo di comprendere ed eventualmente prevedere il comportamento dell'utente medio. Questo campo di ricerca è ancora molto giovane ed esistono ancora molte nuove tecniche da sperimentare ed aree totalmente inesplorate.

Proseguimento del lavoro svolto

La principale motivazione che ha determinato lo studio descritto in questa tesi è quello di integrare lo strumento realizzato in un progetto più ampio: il **KDDML**.

Questi è un ambiente di sviluppo per applicazioni di estrazione di conoscenza da basi di dati che ha avuto come obiettivo principale quello di consentire un alto livello di interoperabilità tra i diversi metodi di data mining e di

rappresentazione della conoscenza. Esso è basato interamente sul linguaggio di mark-up *XML* che si è rilevato dotato di un forte potere espressivo e quindi capace di offrire il grado di interoperabilità richiesto. Il KDDML è stato realizzato recentemente presso il Dipartimento di Informatica dell'Università di Pisa.

Gli sviluppi futuri coinvolgono anche direttamente i naturali miglioramenti che possono essere fatti all'algoritmo: quali ad esempio versione *out memory* di PrefixSpan, in particolare un possibile suo sviluppo, come consigliato dagli autori ([?]), prevede una soluzione che integri la proiezione *bi-level* per processing basati su disco e *pseudo proiezione* quando l'intero database può essere interamente contenuto in memoria principale.

Un possibile successivo miglioramento potrebbe essere quello di aggiungere all'algoritmo un supporto di restrizione non fisso (come discusso nel capitolo 3, paragrafo 3.3.1).

Studi futuri

In generale gli algoritmi per il sequential pattern mining tendono a generare un enorme numero di sequenze che spesso non sono di interesse all'utente. Per esempio un'analista di marketing può essere interessato all'attività di quei clienti online che hanno visitato certe pagine in uno specifico periodo di tempo. In questo caso i pattern trovati devono rispettare certe regole e condizioni che possiamo sintetizzare come segue:

- restrizioni sull'attributo page view (tipo di pagina, nome, data di accesso...)
- restrizione sull'attributo visita utente (lunghezza, durata, min e max intervallo fra le page view...)
- restrizione sull'attributo sequenze di utente (lunghezza, numero di visite, lunghezza e durata media di visita...)

Un ulteriore miglioramento è quello di inserire delle restrizioni direttamente nel processo di mining che permettano di restringere lo spazio di ricerca solo a quello di interesse.

I Web Log continueranno a crescere in dimensione e nuovi dati saranno disponibili, questo determinerà un impulso ulteriore alla ricerca di una

maggiore scalabilità e di algoritmi in grado di tener conto dell'evoluzione dei bisogni di buissness e siti web (*algoritmi incrementali*).

In conclusione l'importanza del **Web Usage Mining** continuerà a crescere con la popolarità del WWW e indubbiamente avrà un impatto significativo sullo studio dei comportamenti degli utenti online.

5.3 Ringraziamenti

Rivolgo un particolare ringraziamento ai relatori Prof. Franco Turini e Prof. Salvatore Ruggieri per la loro guida e i loro consigli nella stesura di questa tesi.

Ringrazio la Dott.sa Campa e il Dott. Rinzivillo per la loro disponibilità oltre che per la loro simpatia.

Ringrazio immensamente i miei genitori, Giorgio e Giuseppina, e tutta la mia famiglia, grazie al loro affetto, al loro continuo sostegno, ai loro sacrifici è stato possibile per me arrivare fin qui.

Un ringraziamento speciale va a tutti coloro che mi sono stati vicini in questi studi, che hanno reso più bella questa mia esperienza e che mi hanno incoraggiato e sostenuto nei momenti difficili (e sono davvero tanti!!!!).

Ultimi ringraziamenti, ma non per questo meno importanti, vanno a Luca, mio fedele compagno di avventura, il cui amore e la cui comprensione sono stati determinanti in questi ultimi anni di studi.

Un grazie di cuore veramente a tutti.

Bibliografia

- [1] R. Agrawal e R. Srikant: *Mining Sequential Pattern*. ICDE'95.
- [2] R. Agrawal e R. Srikant: *Mining Sequential Pattern: Generalizations and Performance Improvement*. 1996.
- [3] *Clementine*. <http://www.spss.com>
- [4] V. Devedzic: *Knowledge Discovery and Data Mining in Databases*. 2000.
- [5] U. Fayyad, G. Piatetsky-Shapiro e P. Smyth: *From Data Mining to Knowledge Discovery in Databases*. 1996.
- [6] W.Frawley, G.Piatetsky-Shapiro, and C.Matheus.*Knowledge Discovery in Databases: An Overview*, 1991.
- [7] C. Gozzi: *Web Mining - State of the Art*,CNUCE-CNR. 2001.
- [8] C. Gozzi: *Web Mining for Personalization*,CNUCE-CNR. 2002.
- [9] M.N. Garofalakis, R. Rastogi e K. Shim: *SPIRIT: Sequential Pattern Mining with Regular Expression Constraints*. 1999
- [10] J. Han, J. Pei, B. Mortazavi-Asl, H. Pinto, Q. Chen, U. Dayal e M. C. Hsu: *FreeSpan: Frequent Pattern-Projected Sequential Pattern Mining*. 2000
- [11] J. Han, J. Pei, B. Mortazavi-Asl e H. Pinto: *PrefixSpan: Mining Sequential Patterns Efficiently by Prefix-Projected Pattern Growth*. 2001
- [12] J. Pei, J. Han e W. Wang: *Mining Sequential Patterns with Constraints in Large Databases*. 2001.
- [13] J. Han, J. Pei e Y. Yin: *Mining Frequent Patterns without Candidate Generation*. 1999.

- [14] N. Ikizler, *Mining Sequential Patterns: an Overview*. 2001.
- [15] M. V. Joshi, G. Karypis e V. Kumar: *A Universal Formulation of Sequential Pattern*. 1999.
- [16] M. V. Joshi, G. Karypis e V. Kumar: *Parallel Algorithms for Mining Sequential Associations: Issues and Challenges*, Technical Report No. 00-002, Dept. of Computer Science, University of Minnesota. 2000.
- [17] B. Mortazavi-Asl. *Discovering and Mining User Web-Page Traversal Patterns*. 2001.
- [18] B. Mobasher, H. Dai, M. Nakagawa *Using Sequential and Non-Sequential Patterns in Predictive Web Usage Mining Tasks*. ICDM'2002.
- [19] H. Mannila, H. Toivonen e A.I. Verkamo: *Discovery of frequent episodes in event frequent*. Data Mining and Knowledge Discovery. 1997.
- [20] H. Mannila, H. Toivonen e A.I. Verkamo: *Discovering frequent episodes in sequences*. 1995.
- [21] H. Pinto, J. Han, J. Pei, K. Wang, Q. Chen e U. Dayal. *Multi-dimensional sequential pattern mining*. In CIKM, pages 81-88, 2001.
- [22] Z. Qiankun: *A survey of Sequential Pattern Mining*. 2002
- [23] A. Romei: *Data Mining su risorse Web: Stato dell'Arte*. 2002
- [24] A. Romei. *Implementazione di un query language per Knowledge Discovery*. Tesi di Laurea, relatore Prof. Franco Turini, 2002.
- [25] J. Srivastava, R. Cooley, M. Deshpande e P. Tan: *Web Usage Mining: Discovery and Application of Usage Patterns from Web Data*. 2000.
- [26] M. Seno e G. Karypis: *SLPMiner: An Algorithm for Finding Frequent Sequential Patterns Using Length-Decreasing Support Constraint*. 2002.
- [27] World Wide Web Committee Web Usage Characterization Activity. <http://www.w3.org/1999/05/WCA-terms/>.
- [28] M.J. Zaki: *Sequence Mining in Categorical Domains: Incorporating Constraints*. 2000.

-
- [29] M.J. Zaki: *SPADE: An efficient algorithm for mining frequent sequences*. 2001.
 - [30] Minghua Zhang, Ben Kao, David Wailok Cheung e Chi Lap Yip: *Efficient algorithms for incremental update of frequent sequences*. In PAKDD, pages 186-197. 2002.
 - [31] Y-L. Chen, S-S. Chena and P-Y. Hsub: *Mining Hybrid Sequential Patterns And Sequential Rules*. 2002.