

# Capitolo 1

## Introduzione

### 1.1 Inquadramento generale

L’analisi e la comprensione di grosse quantità di dati in questi ultimi anni sono diventate una pratica fondamentale per comprendere i fenomeni che ci circondano. In particolare, quando questi fenomeni sono descritti da una posizione geografica e da un momento temporale preciso ci permettono di realizzare delle deduzioni che altrimenti sarebbero impensabili.

In questo ramo della ricerca si inserisce il *Data Mining*, ovvero l’insieme di tecniche e metodologie che hanno per oggetto l’estrazione di informazioni utili da grandi quantità di dati attraverso metodi automatici o semi-automatici (*machine learning*).

Nel lavoro svolto in particolare viene effettuata una *analisi delle associazioni*, una tecnica particolare di data mining utilizzata per la ricerca di connessioni di eventi con determinate caratteristiche.

### 1.2 Breve descrizione del lavoro

L’algoritmo preso in esame è lo *Spatio-Temporal Breath-first Miner (STB-FM)* definito da Piotr S. Maciąg and Robert Bembenik. Questo particolare algoritmo permette di definire delle sequenze di tipologie di eventi connesse nello spazio e nel tempo. Viene definito un vicinato basato su un raggio spaziale e un intervallo di tempo dal quale valutare se il singolo evento è ”vicino” ad altri. Questa valutazione viene fatta per tutti gli eventi di uno stesso tipo

rispetto a tutti gli eventi di un altro tipo. Da queste valutazioni si ricava un valore di *connessione* tra tipi compreso tra  $[0, 1]$ , più questo valore tende a 1 più i rispettivi tipi sono associati.

Questo lavoro viene fatto su diverse combinazioni di tipi, queste combinazioni vengono unite in sequenze e a ogni sequenza viene associato il valore di associazione.

es.

$[A, B, C] - 0.8$

$[B, D] - 0.5$

Tramite questi valori dovremmo essere in grado di capire quanto degli eventi sono associati e, in questi casi, cercare di prevenire/supportare (a seconda dei casi) l'evento di tipologia successiva.

Il caso preso in esame è quello dei crimini avvenuti a Boston, utilizzando il database fornito dal Boston Police Department (BPD) in cui sono registrati tutti i crimini avvenuti a Boston dal 2015, etichettandoli con la tipologia (di crimine), le coordinate GPS dell'evento e il momento in cui avvengono, con il giorno e l'orario.

es.

Offence Code	Date	Lat	Long
LarcFromMotVehic	01/01/2018 00:00	4.235.314.550	-7.107.763.936
ResidentialBurglary	01/01/2018 00:00	4.229.755.533	-7.105.970.910
AggravatedAssault	01/01/2018 02:23	4.235.040.583	-7.106.512.526

*Tabella 1.1: esempio eventi*

Esso rispetta tutti i vincoli di applicazione di questo algoritmo, vi è un gran numero di eventi etichettati per tipologia, geolocalizzati spazialmente e temporalmente, pertanto è stato scelto per l'applicazione pratica.

### 1.3 Scopo e prospettive

Lo scopo di questo lavoro è quindi quello di implementare il l'algoritmo *Spatio-Temporal Breath-First Miner (STBFS)* per capire le sue applicazioni

a casi concreti come quello dei crimini di Boston e analizzarne l'efficacia anche in termini di tempi di computazione.

Esso si apre a possibili sviluppi futuri anche in contesti completamente diversi rispetto a quello preso in esame, come ad esempio l'analisi dell'incidenza di epidemie.

## 1.4 Struttura della tesi

La tesi è strutturata nel seguente modo:

- Nel **capitolo due** si parla della base teorica su cui si basa l'algoritmo, in particolare i calcoli che si effettuano e la struttura dati utilizzata nel paper (anche possibili alternative come algoritmo apriori?)
- Nel **capitolo tre** si analizza in modo più approfondito il dataset utilizzato e le varie considerazioni fatte
- Nel **capitolo quattro** si parla dell'implementazione effettuata
- Nel **capitolo cinque** si analizzano i risultati ottenuti sia in termini di tempi di computazione che in termini di significato degli stessi
- **Conclusioni** e prospettive future



# Capitolo 2

## Stato dell'Arte

### 2.1 Paper

Come precedentemente anticipato, l'algoritmo oggetto di questo lavoro è **A Novel Breadth-first Strategy Algorithm for Discovering Sequential Patterns from Spatio-temporal Data** di Piotr S. Maciąg e Robert Bembenik del *Instituite of Computer Science, Warsaw University of Tecnology, Nowowiejska 15/19, 00-665, Warsaw, Poland.*

Di seguito vi è la base teorica su cui si fonda l'algoritmo e le strutture dati utilizzate per la sua realizzazione.

### 2.2 Vicinato

Il problema che si considera è la scoperta di pattern da un certo dataset di istanze di eventi, i quali sono di una data tipologia.

Definiamo quindi:

$D \rightarrow$  dataset di istanze di eventi

$F \rightarrow$  insieme di tipologie di eventi

Ogni istanza  $e \in D$  ha:

- chiave di identificazione (unica)
- location spaziale (es. coordinate geografiche)
- istante temporale

- tipologia  $f \in F$

La sequenza di eventi (pattern) è così definita:

$$\vec{s} = f_{i_1} \rightarrow f_{i_2} \rightarrow \dots \rightarrow f_{i_n}, \text{ dove } f_{i_1}, f_{i_2}, \dots, f_{i_n} \in F$$

Quindi per ogni due tipi consecutivi di eventi in una sequenza  $f_{i_{j-1}} \rightarrow f_{i_j}$ , le istanze dell'evento di tipo  $j-1$  sono connesse con le istanze del tipo successivo  $j$  spazialmente e temporalmente.

L'insieme di eventi collegati in questo modo a una determinata istanza viene definito **neighborhood** o vicinato.

#### Esempio

Consideriamo una situazione come quella in Fig. 2.1, dove:

$$D = \{a1, a2, b1, b2, b3, b4, b5, b6, b7, b8, c1, c2, c3, c4, c5\}$$

$$F = \{A, B, C\}$$

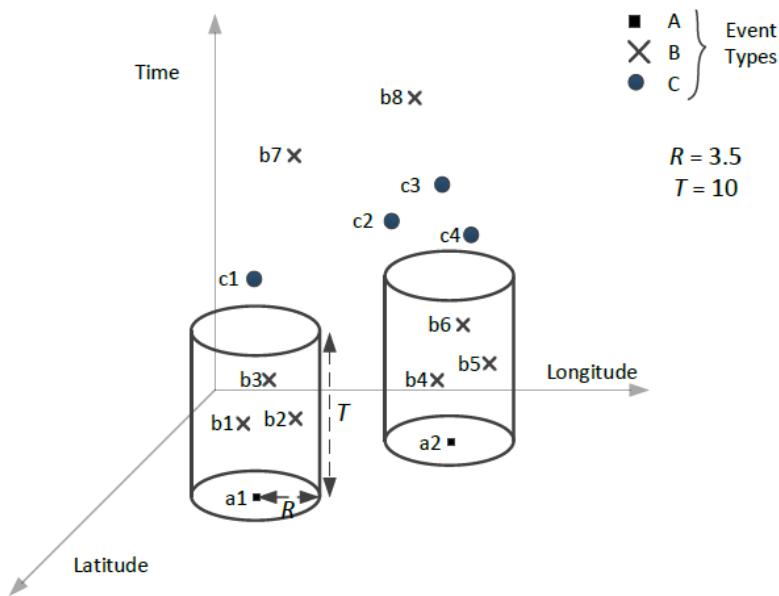


Figura 2.1: esempio di istanze con vicinato degli eventi di tipo A

Una sequenza significativa per esempio potrebbe essere  $\vec{s} = A \rightarrow B \rightarrow C$ . Per valutare la connessione  $A \rightarrow B$  bisogna considerare il *neighborhood* tra le loro istanze. Come si nota dalla figura per la dimensione del vicinato è stato scelto un raggio spaziale pari a  $R = 3.5$  e un intervallo temporale pari a  $T = 10$ .

## 2.3 Nozioni di base

Dopo aver inquadrato graficamente il problema, ora definiamo formalmente i concetti base usati per il calcolo di tutte le sequenze pattern impieghi nell'algoritmo.

**Spazio Neighborhood** Con  $V_{N(e)}$  denotiamo lo spazio di neighborhood (vicinato) dell'istanza  $e$ . Questo spazio si basa su tre dimensioni, che sono le due dimensioni spaziali - latitudine e longitudine - e la dimensione temporale. Graficamente ne risulta un cilindro, con i seguenti parametri:

- $R$ : il raggio spaziale
- $T$ : l'intervallo temporale

In Figura 2.2 vengono mostrati i due neighborhood tratti dall'*esempio* della Figura 2.1:  $V_{N(a1)}$  e  $V_{N(a2)}$ .



Figura 2.2:  $V_{N(a1)}$  e  $V_{N(a2)}$

**Neighborhood rispetto ad una tipologia di evento** Data una certa istanza  $e$ , il *neighborhood* di  $e$  è definito nel modo seguente:

$$\begin{aligned} N_f(e) = \{ & e | p \in D(f) \\ & \wedge \text{distance}(p.\text{location}, e.\text{location}) \leq R \\ & \wedge (p.\text{time} - e.\text{time}) \in [0, T] \} \end{aligned}$$

dove  $R$  e  $T$  sono i parametri dello spazio di vicinato  $V_{N(e)}$  e  $D(f)$  è l'insieme di istanze degli eventi di tipo  $f$  nel dataset  $D$ .

Nota si considerano solo gli eventi che si susseguono dal punto di vista temporale, in quanto è poco significativo nella ricerca di sequenze considerare gli eventi passati dell'istanza.

Riassumendo con  $N_f(e)$  denoto l'insieme di istanze di tipo  $f$  contenute all'interno dello spazio  $V_{N(e)}$ .

Nel nostro *esempio* della Figura 2.2:

$$N_B(a1) = \{b1, b2, b3\} \text{ e } N_B(a2) = \{b4, b5, b6\}$$

**Set di istanze** Per una sequenza di tipi di eventi  $\vec{s} = \vec{s}[1] \rightarrow \vec{s}[2] \rightarrow \dots \rightarrow \vec{s}[m]$  di lunghezza  $m$ , gli insiemi (set) di istanze  $I(\vec{s}[1]), I(\vec{s}[2]), \dots, I(\vec{s}[m])$  che sono inclusi nella sequenza  $\vec{s}$  sono definiti come segue:

1. Per un tipo di evento  $\vec{s}[1]$ , il set di istanze  $I(\vec{s}[1])$  è definito come:

$$I(\vec{s}[1]) = D(\vec{s}[1])$$

2. Per i tipi  $\vec{s}[2] \rightarrow \dots \rightarrow \vec{s}[m]$  con  $i = 2, 3, \dots, m$ , gli insiemi di istanze  $I(\vec{s}[i])$  sono definiti così:

$$I(\vec{s}[i]) = distinct(\bigcup_{e \in I(\vec{s}[i-1])} N_{\vec{s}[i]}(e))$$

In altre parole, per il primo tipo di evento (d'ora in poi nominato solo "tipo") che partecipa alla sequenza  $\vec{s}$ , il set di istanze  $I(\vec{s}[1])$  corrisponde al set di istanze di tipo  $\vec{s}[1]$  in  $D$ , ovvero  $D(\vec{s}[1])$ .

Per i tipi successivi di  $\vec{s}$ , i set  $I(\vec{s}[i])$  sono definiti come insiemi di istanze contenute nei vicinati di istanze a partire da  $I(\vec{s}[i-1])$ .

Seguendo questo meccanismo si valuta tutta la sequenza, tenendo in considerazione l'insieme di istanze calcolato al passaggio precedente.

### *Esempio*

Consideriamo la sequenza  $\vec{s} = A \rightarrow B$  dal dataset dell'*esempio* in Figura 2.1. In questo caso avremmo i seguenti set di istanze:

$$I(\vec{s}[1]) = \{a1, a2\}$$

$$I(\vec{s}[2]) = \{b1, b2, b3, b4, b5, b6\}$$

**Participation Ratio** Data una sequenza  $\vec{s} = \vec{s}[1] \rightarrow \vec{s}[2] \rightarrow \dots \rightarrow \vec{s}[m]$  il *participation rateo* tra due tipi consecutivi contenuti in  $\vec{s}$  è definito:

$$PR(\vec{s}[i-1] \rightarrow \vec{s}[i]) = \frac{|I(\vec{s}[i])|}{|D(\vec{s}[i])|}$$

questo valore corrisponde al numero di istanze distinte di tipo  $\vec{s}[i]$  contenute nei neighborhoods delle istanze di tipo  $\vec{s}[i - 1]$  diviso il numero di istanze di tipo  $\vec{s}[i]$  presenti nel dataset  $D$ .

Per ogni coppia di tipi consecutivi  $(\vec{s}[i - 1], \vec{s}[i])$  in una sequenza  $\vec{s}$  il *participation rateo* è definito come il rapporto tra  $|I(\vec{s}[i])|$  e  $|D(\vec{s}[i])|$  e il suo valore è compreso nel range  $[0, 1]$ .

**Participation Index** Data una sequenza lunga  $m$ :  $\vec{s} = \vec{s}[1] \rightarrow \vec{s}[2] \rightarrow \dots \rightarrow \vec{s}[m]$ , il *participation index* è così definito:

1. se  $m = 2$ :

$$PI(\vec{s}) = PR(\vec{s}[1] \rightarrow \vec{s}[2])$$

2. se  $m > 2$ :

$$PI(\vec{s}) = \min \begin{cases} PI(\vec{s}^*) \\ PR(\vec{s}[m - 1] \rightarrow \vec{s}[m]) \end{cases}$$

dove  $\vec{s}^* = \vec{s}[1] \rightarrow \vec{s}[2] \rightarrow \dots \rightarrow \vec{s}[m - 1]$

Il *participation index* corrisponde al minimo di tutti i *participation rateo* calcolati su due tipi consecutivi presenti in  $\vec{s}$ , ed è il nostro valore di output dell'algoritmo.

Intuitivamente lo possiamo pensare come il numero di istanze di eventi collegate alle istanze di tipi connessi precedentemente secondo un ordine stabilito dalla sequenza e questo ci dà la misura di quanto la sequenza sia correlata.

Consideriamo il nostro *esempio* della Figura 2.1, per la sequenza  $\vec{s} = A \rightarrow B$  il  $PI(\vec{s}) = 0.75$  ( $PI(A \rightarrow B) = PR(A \rightarrow B) = \frac{6}{8} = 0.75$ ) che è un buon risultato di correlazione.



# Capitolo 3

## Algoritmo

### 3.1 Algoritmo di base

Definiti i concetti teorici di base, ora verrà descritto l'algoritmo **Spatio-Temporal Breadth-First Miner** (STBFM) che permette di trovare tutte le sequenze pattern con *participation index* maggiori di una certa soglia **threshold**  $\theta$  data dall'utente.

La definizione di un  $\theta$  permette, da un lato, di considerare come risultato solo le sequenze più significative e, dall'altro, ignorare dalla computazione quelle che hanno un livello di correlazione troppo basso.

Naturalmente bisogna considerare che l'utente definisce il parametro  $\theta$  ma anche il raggio spaziale  $R$  e l'intervallo temporale  $T$  che sono imprescindibili per il calcolo del neighborhood.

**Algorithm 1** L'algoritmo che sta alla base di tutto è l'Algorithm 1. Inizialmente genera le sequenze candidato di lunghezza 2, unendo ciascun tipo con tutti gli altri in modo da coprire tutte le combinazioni possibili. Per tutte queste sequenze viene calcolato il *set di istanze* e basandosi su questo si calcolano i *participation index* (PI) attraverso la funzione VerifyCandidates. Nella fase che segue vengono generate le sequenze candidato del livello successivo ( $C_k$ ) attraverso la funzione CandidateGen (riga 5), con calcolo dei *set di istanze* annessi. Successivamente vengono calcolati i valori di PI associati alle sequenze appena generate ( $L_k$ ), sempre utilizzando la funzione VerifyCandidates (riga 6).

Queste operazioni avvengono per ogni insieme di sequenze di un certa lunghezza iterativamente finché l'insieme di sequenze di lunghezza  $k - 1$  non sia

vuoto. Tutti questi risultati vengono registrati nella famiglia di insiemi  $L$  che verrà poi ritornata.

---

**Algorithm 1:** Spatio-temporal breadth-first miner for discovering significant sequential patterns (STBFM)

---

**Data:** D - dataset delle istanze degli eventi, F - insieme di tipi di eventi, R - raggio spaziale, T - intervallo temporale,  $\theta$  - threshold

**Result:**  $Top$  - insieme delle  $N$  sequenze più significative

- 1  $C_2 :=$  generazione candidati di lunghezza 2
- 2  $L_2 := \text{VerifyCandidates}(C_2)$
- 3  $k = 3$
- 4 **while**  $L_{k-1} \neq \emptyset$  **do**
- 5      $C_k := \text{CandidateGen}(L_{k-1})$
- 6      $L_k := \text{VerifyCandidates}(C_k)$
- 7     Add  $L_k$  to  $L$
- 8      $k := k + 1$
- 9 **return**  $L$

---



---

**Algorithm 2:** VerifyCandidates

---

**Data:**  $C_m$  - insieme di candidati sequenze di tipi di lunghezza  $m$ ,  $\theta$  - threshold

**Result:**  $L_m$  - insieme di sequenze di tipi significativi di lunghezza  $m$

- 1  $C_m := \emptyset$
- 2 **foreach**  $\vec{s} \in C_m$  **do**
- 3      $\text{CalculatePI}(\vec{s})$
- 4     **if**  $PI(\vec{s}) \geq \theta$  **then**
- 5         Add  $\vec{s}$  to  $L_m$
- 6 **return**  $L_m$

---

**Algorithm 2** L'Algorithm 2 ha il compito di verificare le sequenze candidato. La verifica consiste nel calcolo del *participation index* della sequenza e se questo valore supera il threshold  $\theta$  allora la si accetta. Questa valutazione viene effettuata iterativamente per tutte le sequenze candidato.

Il calcolo del PI viene effettuato seguendo la definizione nella sua applicazione nell'**Algorithm 4**.

---

**Algorithm 3:** CandidateGen( $L_{m-1}$ )

---

**Data:**  $L_{m-1}$  - insieme di sequenze di tipi di lunghezza  $m-1$   
**Result:**  $C_m$  - insieme di candidati sequenze di tipi di lunghezza  $m$

```

1  $C_m := \emptyset$ 
2 foreach  $\vec{s}_i \in L_{m-1}$  do
3   foreach  $\vec{s}_j \in L_{m-1} \wedge \vec{s}_i \neq \vec{s}_j$  do
4     if  $\vec{s}_i[2] = \vec{s}_j[1] \wedge \vec{s}_i[3] = \vec{s}_j[2] \wedge \dots \wedge \vec{s}_i[m] = \vec{s}_j[m-1]$  then
5        $\vec{s} := \vec{s}_i[1] \rightarrow \vec{s}_i[2] \rightarrow \dots \rightarrow \vec{s}_i[m-1] \rightarrow \vec{s}_j[m-1]$ 
6        $I(\vec{s}[1]) := I(\vec{s}_i[1]) \wedge I(\vec{s}[2]) := I(\vec{s}_i[2]) \wedge \dots \wedge$ 
7        $I(\vec{s}[m-1]) := I(\vec{s}_i[m-1]) \wedge$ 
     CalculateNeighborhood( $I(\vec{s}_i[m-1]), I(\vec{s}_j[m-1])$ )
    Add  $\vec{s}$  to  $C_m$ 
8 return  $C_m$ 

```

---

**Algorithm 3** L'Algorithm 3 genera un insieme di sequenze candidato di lunghezza  $m$  basate sulle sequenze già analizzate di lunghezza  $m-1$ .

La generazione avviene secondo la seguente logica:

due sequenze di lunghezza  $m-1$  si uniscono per creare una nuova sequenza di lunghezza  $m$ , se contengono gli stessi tipi di eventi in certe posizioni: il secondo tipo della prima sequenza è uguale al primo tipo della seconda sequenza, il terzo tipo della prima sequenza è uguale al secondo tipo della seconda sequenza e così via finché l'ultimo tipo della prima sequenza non è uguale al penultimo tipo della seconda sequenza.

*Esempio:*

$$\begin{aligned} P_1 &= \{A \rightarrow B \rightarrow C \rightarrow D\} \text{ (sequenza lunga } m-1) \\ P_2 &= \{B \rightarrow C \rightarrow D \rightarrow E\} \text{ (sequenza lunga } m-1) \\ P_{new} &= \{A \rightarrow B \rightarrow C \rightarrow D \rightarrow E\} \text{ (sequenza lunga } m) \end{aligned}$$

Come si può notare, la nuova sequenza ha tutti i tipi della prima e come ultimo elemento l'ultimo tipo della seconda sequenza.

Per quanto riguarda il calcolo del *set di istanze* (riga 6) ci sono 2 possibili casi:

1. per i tipi di eventi dal primo al penultimo elemento della sequenza, il set di istanze è lo stesso di quello della prima sequenza quindi di

$I(\vec{s}_i[m - 1]);$

2. per l'ultimo tipo di eventi, il set di istanze è dato dal neighborhood che c'è tra il penultimo tipo (che corrisponde al set di istanze della prima sequenza) e l'ultimo tipo della seconda sequenza, ovvero  
 $\text{CalculateNeighborhood}(I(\vec{s}_i[m - 1]), I(\vec{s}_j[m - 1])).$

Per il calcolo del vicinato si utilizza la definizione applicata nell'**Algorithm 5**.

**Algorithm 4** L'Algorithm 4 definisce il calcolo del *participation index* come definito dalla formula:  $PI(\vec{s}) = \min \begin{cases} PI(\vec{s}^*) \\ PR(\vec{s}[m - 1] \rightarrow \vec{s}[m]) \end{cases}$  dove  $\vec{s}^* = \vec{s}[1] \rightarrow \vec{s}[2] \rightarrow \dots \rightarrow \vec{s}[m - 1].$

---

**Algorithm 4:** CalculatePI( $\vec{s}$ )

**Data:**  $\vec{s} = \vec{s}[1] \rightarrow \vec{s}[2] \rightarrow \dots \rightarrow \vec{s}[m - 1] \rightarrow \vec{s}[m]$  - una sequenza di tipi di eventi

**1 return**

$\min(PI(\vec{s}[1] \rightarrow \vec{s}[2] \rightarrow \dots \rightarrow \vec{s}[m - 1]), PR(\vec{s}[m - 1] \rightarrow \vec{s}[m]))$

---

**Algorithm 5** L'Algorithm 5 si occupa del calcolo del neighborhood. Esso si applica secondo la definizione data in precedenza, però non prendendo i dati direttamente dal dataset. Si considera  $I(\vec{s}_i[m - 1])$  come set di partenza e si considera il vicinato con l'ultimo elemento della seconda sequenza  $I(\vec{s}_j[m - 1]).$

**Algorithm 5:** CalculateNeighborhood( $I(\vec{s}_i[m - 1]), I(\vec{s}_j[m - 1])$ )

**Data:**  $I(\vec{s}_i[m - 1])$  - un insieme di istanze di tipo  $\vec{s}[m]$  della sequenza  $\vec{s}_i$ ,  $I(\vec{s}_j[m - 1])$  - un insieme di istanze di eventi di tipo  $\vec{s}_j[m]$  della sequenza  $\vec{s}_j$

**Result:**  $I(\vec{s}[m])$  - un insieme di istanze di eventi di tipo  $\vec{s}[m]$  della sequenza candidato  $\vec{s}$

**1 return**  $I(\vec{s}[i])$  as  $\text{distinct}(\bigcup_{e \in I(\vec{s}_i[m - 1])} N_{\vec{s}_j[m - 1]}(e))$

---

Come si può da subito notare il carico computazionale più pesante lo si ha nell’Algorithm 3, la *CandidateGen*. In quest’ultimo bisogna valutare tutte le possibili coppie di sequenze del livello precedente se sono compatibili tra loro per generare le nuove sequenze del livello successivo e i rispettivi set di istanze.

Se consideriamo anche solo un dataset di 10.000 record con 20 tipi di eventi, significa che l’algoritmo itera su tutti i 10.000 record migliaia di volte per ciascuna sequenza da generare.

Certamente un threshold  $\theta$  alto ci permette di ignorare diverse combinazioni, però è un parametro handmade. Pertanto se si sceglie un  $\theta$  troppo alto si rischia di avere pochi risultati con PI molto alto, se si sceglie un  $\theta$  troppo basso si rischia di considerare troppe sequenze che dilaterebbero troppo i tempi di computazione.

Per questi motivi viene definita una nuova struttura ad albero per evitare moltissime di queste computazioni (*SPTree*) e viene migliorato l’algoritmo in modo da rendere il parametro  $\theta$  variabile in base al numero di risultati significativi, come vediamo di seguito.

## 3.2 Struttura ad albero

A sostegno dell'algoritmo viene definita una struttura ad albero dedicata per ridurre drasticamente il numero delle computazioni, in particolare per quanto riguarda la generazione di sequenze candidato nuove.

### 3.2.1 Albero

Innanzitutto definiamo cos'è una struttura ad albero ordinato:

Un **albero ordinato** è un grafo connesso, aciclico ed ordinato. Esso è una struttura dati che si compone di due sottostrutture fondamentali:

- il **nodo** che solitamente contiene informazioni o altre strutture dati;
- l'**arco** che stabilisce un collegamento gerarchico fra due nodi.

Si definisce *nodo padre* il nodo che ha un arco orientato uscente che va in un *nodo figlio*.

La **radice** è il nodo che si distingue da tutti gli altri in quanto essa è l'unico nodo da cui vi sono solo archi uscenti.

I nodi **foglia** sono i nodi che hanno solo archi uscenti ma non entranti, corrispondono ai nodi dell'ultimo livello di albero.

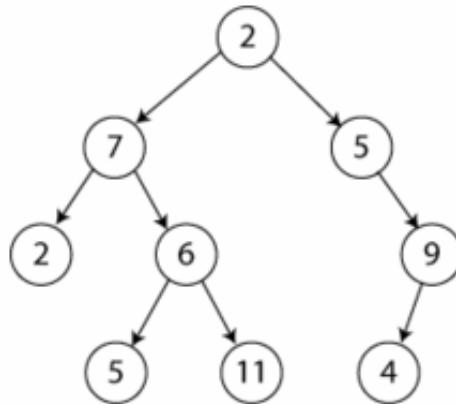


Figura 3.1: esempio di albero ordinato

### 3.2.2 Sequence Pattern Tree

L’albero utilizzato nel lavoro svolto è chiamato **Sequence Pattern Tree** (*SPTree*).

Consideriamo un insieme di tipi di eventi di questo tipo:  $F = \{A, B, C, D, E, F\}$  e un insieme  $L$  di sequenze pattern presentate in Tabella 3.1.

$F$	$A, B, C, D, E, F$
$L$	Sequenze pattern
$L_2$	$A \rightarrow B, B \rightarrow C, B \rightarrow D, C \rightarrow E, C \rightarrow F,$ $A \rightarrow B \rightarrow C, A \rightarrow B \rightarrow D,$
$L_3$	$B \rightarrow C \rightarrow E, B \rightarrow C \rightarrow F.$
$L_4$	$A \rightarrow B \rightarrow C \rightarrow E, A \rightarrow B \rightarrow C \rightarrow F.$

Tabella 3.1: esempio di sequenze pattern

Partendo dalla radice verranno inserite tutte le sequenze significative secondo i criteri che seguono.

La **radice** ha valore nullo e come figli tutti i tipi di eventi presenti in  $F$ . Assumiamo che le sequenze di lunghezza 2 ( $L_2$ ) siano state generate dai tipi di partenza effettuando tutte le combinazioni tra di essi, con le verifiche del *participation rateo* come presentato dal  $L_2$  dell’albero in Figura 3.1.

Per ogni sequenza, ovvero ogni nodo dell’albero, vengono impiegate tre strutture di dati: *firstParent*, *secondParent* e *children*.

Per la generazione dei **nodi** si procede per livello, ovvero ci si basa sul livello immediatamente precedente e si genera quello sottostante, seguendo queste procedure:

- se la sequenza  $\vec{s}$  è stata creata unendo i tipi di eventi  $f_{i_1}$  e  $f_{i_2}$  a  $\vec{s} = f_{i_1} \rightarrow f_{i_2}$ , allora:  
 $firstParent(\vec{s}) := f_{i_1}$   
 $secondParent(\vec{s}) := f_{i_2}$   
 $\vec{s}$  viene aggiunta ai *children*( $\vec{s}$ )
- se la sequenza  $\vec{s}$  è stata creata dalle sequenze  $\vec{s}_i$  e  $\vec{s}_j$ , allora:  
 $firstParent(\vec{s}) := \vec{s}_i$   
 $secondParent(\vec{s}) := \vec{s}_j$   
 $\vec{s}$  viene aggiunta ai *children*( $\vec{s}$ )

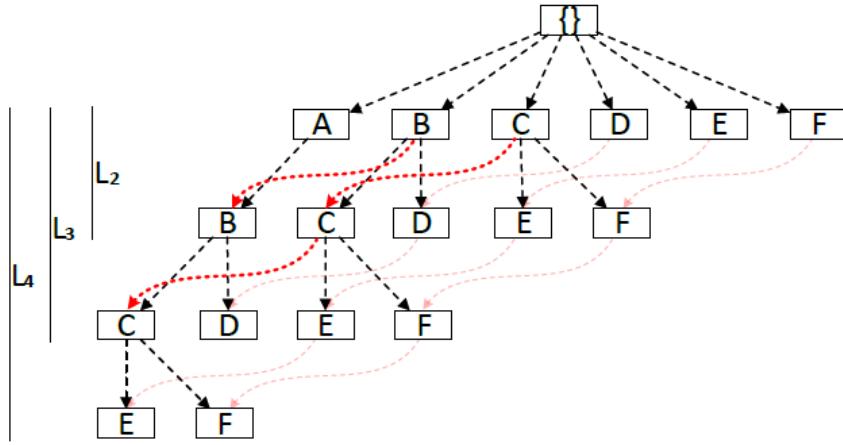


Figura 3.2: SPTree basato sulle sequenze della Tabella 3.1

### Esempio

Consideriamo la sequenza  $A \rightarrow B$  dalla Tabella 3.1 e quindi la sua memorizzazione nel SPTree come in Figura 2.3. Assumiamo che dobbiamo generare i candidati di lunghezza 3 quindi generare il terzo livello dell'albero.

In questo caso  $A \rightarrow B$  può essere estesa con due diversi tipi  $C$  o  $D$ , in quanto  $A \rightarrow B$  ha come *secondParent* il tipo  $B$  ed esso ha come sequenze figlio (*children*)  $B \rightarrow C$  e  $B \rightarrow D$ , pertanto le posso aggiungere alla sequenza  $A \rightarrow B$ . Seguendo questo processo viene generato l'SPTree.

### 3.3 STBFM

L'integrazione tra l'algoritmo precedentemente mostrato e l'albero *SPTree* porta numerosi e fondamentali vantaggi in termini di tempi computazionali. L'algoritmo risultante rispetta tutte le condizioni date e svolge la computazione in tempi molto più ristretti.

Questa versione dell'algoritmo è in grado di trovare un insieme di  $N$  sequenze pattern significative da un certo dataset, rispetto a ricavare *tutte* le sequenze pattern significative. Viene introdotta la seguente definizione:

**Top-N Sequential Pattern** Una sequenza pattern  $\vec{s}$  è l' $N$ -esima sequenza pattern, se esistono  $N - 1$  sequenze nel insieme dei *Top* con *participation index* maggiore o uguale al  $\text{PI}(\vec{s})$ .

Detto ciò si definisce un insieme *Top* formato da  $N$  sequenze pattern significative che corrisponde all'output dell'algoritmo stesso.

Partendo dagli algoritmi precedentemente definiti integriamo l'albero *SPTree*.

**Algorithm 6** Si parte dall'Algorithm 3 che si occupa della generazione delle possibili sequenze candidato. Presa una sequenza già valutata  $\vec{s}_i$  e il suo corrispondente nodo, si estrae una sequenza candidato  $\vec{s}_j$  nel insieme *children* del suo *secondParent* (riga 3).

Il ruolo del *secondParent* è quello di verificare la presenza di possibili prolungamenti della sequenza  $\vec{s}_i$  presa in esame. Come si può notare dall'albero di esempio in Figura 3.1 si può generare  $A \rightarrow B \rightarrow C$  ( $\vec{s}$ ) partendo da  $A \rightarrow B$  ( $\vec{s}_i$ ) aggiungendo la sequenza  $B \rightarrow C$  ( $\vec{s}_j$ ) presente nei *children*(*secondParent*( $\vec{s}_i$ )).

In concreto il *secondParent* collega un certo nodo al nodo di livello precedente con il suo stesso valore finale di sequenza, dai *children* di quest'ultimo si può verificare la presenza di possibili nuove sequenze.

Dai due nodi relativi alle due sequenze  $\vec{s}_i$ ,  $\vec{s}_j$  viene creato il nuovo nodo per la nuova sequenza  $\vec{s}$  che avrà come  $\text{firstParent}(\vec{s}) := \vec{s}_i$  e come  $\text{secondParent}(\vec{s}) := \vec{s}_j$  (riga 6), queste due proprietà garantiscono la possibilità della ricerca di un prolungamento ulteriore dell'albero. Infine si aggiunge ai *children* di  $\vec{s}_i$  il nodo creato (in quanto  $\vec{s}_i$  è *firstParent*) e si

completa l'inserimento (riga 7).

Queste istruzioni vengono svolte iterativamente per tutti i nodi dell'ultimo livello dell'albero ( $m - 1$ ) così da generare il livello successivo ( $m$ ). Il nuovo livello dell' $SPTree$ , come si può notare, si limita alle nuove sequenze candidato significative rispetto a quelle in precedenza generate.

---

**Algorithm 6:** CandidateGen( $L_{m-1}$ )

---

**Data:**  $L_{m-1}$  - insieme di sequenze di tipi di lunghezza  $m - 1$   
**Result:**  $C_m$  - insieme di candidati sequenze di tipi di lunghezza  $m$

- 1  $C_m := \emptyset$
- 2 **foreach**  $\vec{s}_i \in L_{m-1}$  **do**
- 3     **foreach**  $\vec{s}_j \in children(secondParent(\vec{s}_i))$  **do**
- 4          $\vec{s}' := \vec{s}_i[1] \rightarrow \vec{s}_i[2] \rightarrow \dots \rightarrow \vec{s}_i[m-1] \rightarrow \vec{s}_j[m-1]$
- 5          $I(\vec{s}'[1]) := I(\vec{s}_i[1]) \wedge I(\vec{s}'[2]) := I(\vec{s}_i[2]) \wedge \dots \wedge$
- 6          $I(\vec{s}'[m-1]) := I(\vec{s}_i[m-1]) \wedge$
- 7          $I(\vec{s}'[m]) :=$   
           CalculateNeighborhood( $I(\vec{s}_i[m-1]), I(\vec{s}_j[m-1])$ )
- 6          $firstParent(\vec{s}') := \vec{s}_i$ ,  $secondParent(\vec{s}') := \vec{s}_j$
- 7         Add  $\vec{s}'$  to  $children(\vec{s}_i)$ , Add  $\vec{s}'$  to  $C_m$
- 8 **return**  $C_m$

---



---

**Algorithm 7:** VerifyCandidates

---

**Data:**  $C_m$  - insieme di candidati sequenze di tipi di lunghezza  $m$ ,  $\theta$  - threshold  
**Result:**  $L_m$  - insieme di sequenze di tipi significativi di lunghezza  $m$

- 1  $C_m := \emptyset$
- 2 **foreach**  $\vec{s} \in C_m$  **do**
- 3     CalculatePI( $\vec{s}$ )
- 4     **if**  $PI(\vec{s}) \geq \theta$  **then**
- 5         Add  $\vec{s}$  to  $L_m$
- 6     **else**
- 7         Remove  $\vec{s}$  from  $children(firstParent(s))$
- 8 **return**  $L_m$

---

**Algorithm 7** L’Algorithm 7 parte dalla base dell’ Algorithm 2 e integra l’*SPTree*. Questo algoritmo si occupa del calcolo dei *participation index* delle nuove sequenze candidato e se essi superano il threshold  $\theta$  allora vengono inserite nell’insieme di sequenze significative (riga 5), altrimenti viene eliminata la rispettiva sequenza presente nell’albero.

Il *pruning* dell’albero avviene attraverso la cancellazione del collegamento che ha il nodo della sequenza  $\vec{s}$  con il nodo *firstParent* (riga 7).

Quest’ultimo algoritmo però non ricerca l’insieme di sequenze che rispettano la definizione di *Top-N sequential pattern*. Per questo viene ridefinito nell’Algorithm 8.

**Algorithm 8** L’Algorithm 8 ridefinisce l’Algorithm 7 in modo da creare solo un determinato numero  $N$  di sequenze significative memorizzate nell’insieme *Top*. Pertanto la cardinalità di *Top* ( $|Top|$ ) è uguale a  $N$ .

Calcolato il *participation index*, se esso è maggiore del threshold  $\theta$  vi sono tre scenari possibili:

1. se il numero di sequenze in *Top* è minore di  $N - 1$ , allora la sequenza candidato viene inserita in *Top* e  $L_m$  (riga 6);
2. se il numero di sequenze in *Top* è uguale a  $N - 1$ , allora la sequenza candidato viene inserita in *Top* e  $L_m$  e  $\theta$  viene aumentato al valore del PI dell’N-esima sequenza top, in quanto l’insieme *Top* è stato riempito di  $N$  sequenze (righe 8-9);
3. se il numero di sequenze in *Top* è uguale ad  $N$ , allora la sequenza candidato viene inserita in *Top* e  $L_m$  e se il suo PI è maggiore del threshold attuale  $\theta$ , allora  $\theta$  viene settato al valore del PI dell’N-esima sequenza top attuale e tutte le sequenze con PI minore del nuovo  $\theta$  vengono eliminate da *Top* e da  $L_m$ , ovviamente di tutte queste sequenze cancellate viene fatto il *pruning* nell’*SPTree* (righe 11-16).

---

**Algorithm 8:** Top-N-VerifyCandiatesSPTree( $C_m$ )

---

**Data:**  $C_m$  - un insieme di sequenze pattern candidato di lunghezza  $m$ ,  $\theta$  - threshold

```
1  $L_m := \emptyset$ 
2 foreach  $\vec{s} \in C_m$  do
3   CalculatePI( $\vec{s}$ )
4   if  $PI(\vec{s}) \geq \theta$  then
5     if  $|Top| < N - 1$  then
6       Add  $\vec{s}$  to  $L_m$ , add  $\vec{s}$  to  $Top$ 
7     else if  $|Top| = N - 1$  then
8       Add  $\vec{s}$  to  $L_m$ , add  $\vec{s}$  to  $Top$ 
9        $\theta := PI(Top(N))$ 
10    else
11      Add  $\vec{s}$  to  $L_m$ , add  $\vec{s}$  to  $Top$ 
12      if  $PI(\vec{s}) > \theta$  then
13         $\theta := PI(Top(N))$ 
14        Delete all  $\vec{s}' \in Top$  with  $PI(\vec{s}') < \theta$ 
15        Delete all  $\vec{s}' \in L_m$  with  $PI(\vec{s}') < \theta$ 
16        Delete  $\vec{s}'$  from  $children(firstParent(\vec{s}'))$ 
17    else
18      Remove  $\vec{s}'$  from  $children(firstParent(\vec{s}'))$ 
19 return  $L_m$ 
```

---

Come si può notare quest'ultimo algoritmo garantisce due premesse fondamentali per la risoluzione del problema: l'output di un numero fissato  $N$  di sequenze significative ed un  $\theta$  che, una volta inizializzato dall'utente, aumenta in base ai risultati ottenuti.

La combinazione di Algorithm 1, Algorithm 6 ed Algorithm 8 forma l'algoritmo **Spatio-Temporal Breadth-First Miner**.

### 3.4 Alternativa - Algoritmo apriori