Università degli Studi di Milano - Bicocca
Scuola di Scienze
Dipartimento di Informatica, Sistemistica e Comunicazione
Corso di laurea in Informatica

Identificazione di eventi spazio-temporali associati: studio e implementazione su dataset di crimini

Relatore: Prof. Fabio Antonio Stella

Correlatore: Dott. ssa Paola Chiesa

Relazione della prova finale di:
Federico Luzzi
Matricola 816753

Anno Accademico 2018-2019

Indice

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 creare valore e 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*).
Le tecniche di data mining utilizzate nel lavoro svolto sono quelle delle *regole di associazione*, ossia metodi per estrarre relazioni nascoste tra i dati.
Le condizioni che rendono possibile il raggiungimento di risultati significativi sono:

- un insieme di n etichette (item): $I = \{i_1, i_2, ..., i_n\}$;
- un insieme di transazioni (dataset): $D = \{t_1, t_2, ..., t_m\}$

Ciascuna transazione $t \in D$ possiede un codice identificativo e un sottoinsieme di oggetti contenuti in I. Una *regola* è definita come un'implicazione $X \to Y$ dove $X, Y \in I$ e $X \cap Y \neq \emptyset$.

L'applicazione pratica tipica di questi metodi è l'analisi degli acquisti in un

supermercato. Consideriamo la seguente tabella:

ID	latte	omogenizzato	pannolini	birra
1	1	1	0	0
2	0	0	1	1
3	0	0	1	1
4	1	1	1	1
5	1	0	0	0

Tabella 1.1: esempio supermercato

L'insieme di oggetti è $I = \{latte, omogenizzato, pannolini, birra\}$, il database è rappresentato dalla Tabella 1.1, dove 1 indica l'acquisto dell'oggetto in una transazione, 0 invece l'assenza del prodotto.
Come si può notare, vi è una forte presenza della coppia di oggetti $\{pannolini, birra\}$ per cui una possibile regola di associazione che possiamo dedurre è $\{pannolini\} \rightarrow \{birra\}$.
La regola di associazione attesta che un'alta percentuale di persone che acquistano pannolini, acquistano, solitamente, anche birra.
Da una prima analisi questa associazione sembra completamente controintuitiva. Eppure questo è forse il più iconico degli esempi di applicazioni di data mining. Infatti dall'analisi dei dati degli scontrini degli acquisti del fine settimana emergeva questa associazione, mettendo in luce il segmento delle coppie di giovani neo-genitori che invitavano gli amici a casa a cena nel weekend.
Dalla scoperta di questa informazione nascosta, il gestore può applicare dei miglioramenti al supermercato, quali avvicinare il reparto pannolini al reparto birra in modo da agevolare i clienti e possibilmente aumentare le vendite o proporre delle offerte dedicate.

Dal piccolo esempio appena esposto si possono intuire le potenzialità offerte da questi algoritmi, nonchè la loro applicazione in svariati ambiti, per citarne alcuni: l'amministrazione aziendale, la prevenzione di epidemie e la previsione del crimine, proprio su quest'ultima si concentra il lavoro svolto.

Essendo i campi di applicazione molto diversi e di complessità crescente, si ha la necessità di algoritmi sempre più sofisticati e scalabili. Nel seguente ela-

borato si studia un algoritmo di data mining che mira a trovare associazioni di eventi avvenuti in un istante temporale e in un luogo preciso.

1.2 Breve descrizione del lavoro

L'algoritmo preso in esame è lo *Spatio-Temporal Breath-first Miner (STBFM)* definito da Piotr S. Maciag and Robert Bembenik. Esso cerca di scoprire associazioni di tipologie di eventi connesse spazialmente e temporalmente.
Viene definito un vicinato basato su un raggio spaziale e un intervallo di tempo con il quale valutare se il singolo evento è "vicino" (o in relazione) ad altri. Le relazioni di vicinato vengono considerate per tutti gli eventi di uno stesso tipo rispetto a tutti gli eventi di un altro tipo. In base ai vicinati trovati si ricava un valore di *connessione* tra tipi compreso tra $[0, 1]$, più questo valore tende a 1 più i rispettivi tipi sono associati.
Diverse sequenze di tipi vengono considerate e ad ogni una si calcola il valore di associazione, in questo modo si è in grado di confrontare quali sequenze sono più signficative di altre.
es.

$A \rightarrow B \rightarrow C$ - 0.5
$B \rightarrow D$ - 0.6
$\{pannolini\} \rightarrow \{birra\}$ - 0.8

Tramite i risultati ottenuti siamo in grado di comprendere le relazioni tra gli eventi e, in base all'applicazione, si supportano oppure si ostacolano.
Si supportano nei casi in cui creano valore come l'esempio $\{pannolini\} \rightarrow \{birra\}$; si ostacolano nel caso di eventi negativi, ad esempio se si considera una sequenza di eventi criminali che ha una elevata correlazione come $borseggio \rightarrow$ *furto con scasso*, si rileva la prima e si cerca di prevenire la seconda.

Il caso pratico considerato 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 temporale 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 1.2: estratto del database del BPD

Come si può notare dall'estratto in Tabella 1.2, il database rispetta i vincoli di applicazione di questo algoritmo: vi è un gran numero di eventi etichettati per tipologia, localizzati spazialmente e temporalmente.

Il fine ultimo dell'applicazione è quindi la scoperta delle sequenze di eventi criminali meno evidenti rispetto a quelle già conosciute e monitorate dalle forze dell'ordine ma ugualmente significative, in modo da supportare la lotta alla criminalità.

1.3 Scopo e prospettive

Lo scopo di questo lavoro è implementare 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 risultati e 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 così strutturata:

- Nel **capitolo due** si parla della base teorica su cui si basa l'algoritmo, in particolare le definizioni di base
- Nel **capitolo tre** viene presentato l'algoritmo ricostruendo il suo percorso di sviluppo e la struttura dati di appoggio utilizzata
- Nel **capitolo quattro** si analizza in modo più approfondito il dataset utilizzato e l'implementazione dell'algoritmo
- 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 descritto in:

> **A Novel Breadth-first Strategy Algorithm for Discovering Sequential Patterns from Spatio-temporal Data** (Piotr S. Maciag e Robert Bembenik, 2019).

Di seguito vi è la base teorica su cui si fonda l'algoritmo.

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:

$\overrightarrow{s} = f_{i_1} \rightarrow f_{i_2} \rightarrow ... \rightarrow f_{i_n}$, dove $f_{i_1}, f_{i_2}, ..., f_{i_n} \in F$

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:

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

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

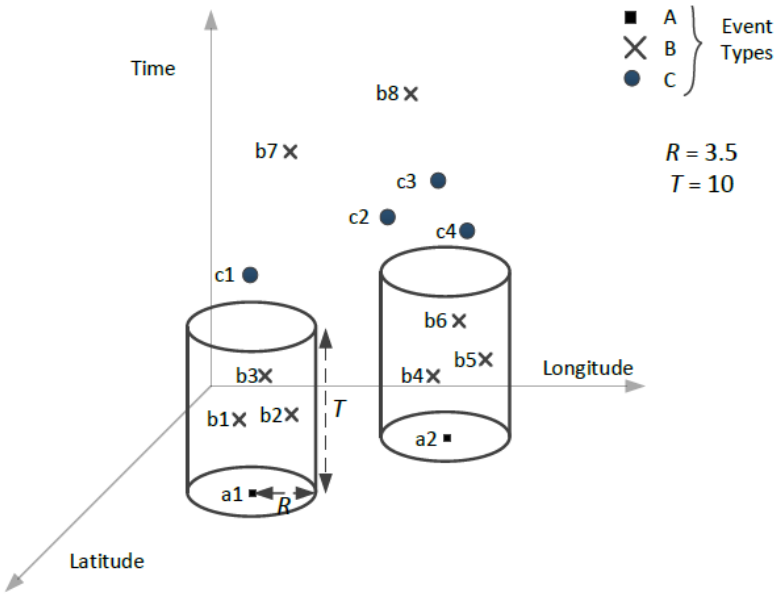

Figura 2.1: esempio di istanze con vicinato degli eventi di tipo A

Una sequenza significativa per esempio potrebbe essere $\overrightarrow{s} = A \rightarrow B \rightarrow C$. 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 impiegati nell'algoritmo.

2.3.1 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 $V_{N(a2)}$.

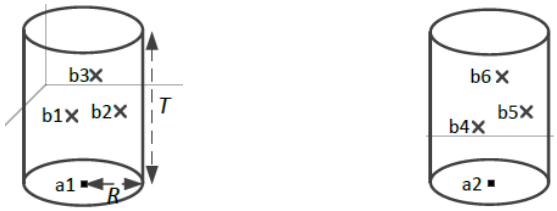

Figura 2.2: $V_{N(a1)}$ *e* $V_{N(a2)}$

2.3.2 Neighborhood rispetto ad una tipologia di evento

Data una certa istanza e, il *neighborhood* di e rispetto ad istanze di tipo f è definito nel modo seguente:

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

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.

Nel nostro *esempio* della Figura 2.2, cerco gli eventi di tipo B presenti in $V_{N(a1)}$ e $V_{N(a2)}$:
$N_B(a1) = \{b1, b2, b3\}$
$N_B(a2) = \{b4, b5, b6\}$

2.3.3 Set di istanze

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

1. Per un tipo di evento $\overrightarrow{s}[1]$, il set di istanze $I(\overrightarrow{s}[1])$ è definito come:
   $$I(\overrightarrow{s}[1]) = D(\overrightarrow{s}[1])$$
2. Per i tipi $\overrightarrow{s}[2] \rightarrow ... \rightarrow \overrightarrow{s}[m]$ con $i = 2, 3, ..., m$, gli insiemi di istanze $I(\overrightarrow{s}[i]$ sono definiti così:
   $$I(\overrightarrow{s}[i]) = distinct(\bigcup_{e \in I(\overrightarrow{s}[i-1])} N_{\overrightarrow{s}[i]}(e))$$

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

Per i tipi successivi di $\overrightarrow{s}$, i set $I(\overrightarrow{s}[i])$ sono definiti come insiemi di istanze contenute nei vicinati di istanze a partire da $I(\overrightarrow{s}[i-1])$.
Seguendo questo meccanismo si valuta tutta la sequenza, tenendo in considerazione l'insieme di istanze calcolato al passaggio precedente.

Esempio
Consideriamo la sequenza $\overrightarrow{s} = A \rightarrow B$ dal dataset dell'*esempio* in Figura 2.1. In questo caso avremmo i seguenti set di istanze:
$I(\overrightarrow{s}[1]) = \{a1, a2\}$
$I(\overrightarrow{s}[2]) = \{b1, b2, b3, b4, b5, b6\}$

2.3.4 Participation Ratio

Data una sequenza $\overrightarrow{s} = \overrightarrow{s}[1] \rightarrow \overrightarrow{s}[2] \rightarrow ... \rightarrow \overrightarrow{s}[m]$ il **participation ratio** tra due tipi consecutivi contenuti in $\overrightarrow{s}$ è definito:

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

il valore corrisponde al numero di istanze distinte di tipo $\overrightarrow{s}[i]$ contenute nei neighborhoods delle istanze di tipo $\overrightarrow{s}[i-1]$ diviso il numero di istanze di tipo $\overrightarrow{s}[i]$ presenti nel dataset D.
Per ogni coppia di tipi consecutivi $(\overrightarrow{s}[i-1], \overrightarrow{s}[i])$ in una sequenza $\overrightarrow{s}$ il *participation rateo* è definito come il rapporto tra $|I(\overrightarrow{s}[i])|$ e $|D(\overrightarrow{s}[i])|$ e il suo valore è compreso nel range $[0, 1]$.

2.3.5 Participation Index

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

1. se $m = 2$:
$$PI(\overrightarrow{s}) = PR(\overrightarrow{s}[1] \rightarrow \overrightarrow{s}[2])$$
2. se $m > 2$:
$$PI(\overrightarrow{s}) = min \begin{cases} PI(\overrightarrow{s}^*) \\ PR(\overrightarrow{s}[m-1] \rightarrow \overrightarrow{s}[m]) \end{cases}$$

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

Il **participation index** corrisponde al minimo di tutti i *participation ratio* calcolati su tutte le coppie di tipi consecutivi presenti in $\overrightarrow{s}$, 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 $\overrightarrow{s} = A \rightarrow B$ il $PI(\overrightarrow{s}) = 0.75$, infatti $PI(A \rightarrow B) = PR(A \rightarrow B) = \frac{6}{8} = 0.75$, 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* maggiore di una certa soglia **threshold** θ data dall'utente.
La definizione di un θ 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 θ 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 finchè 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, θ - threshold

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

```
1 C_2 := generazione candidati di lunghezza 2
2 L_2 := VerifyCandidates(C_2)
3 k = 3
4 while L_{k-1} ≠ ∅ do
5 |   C_k := CandidateGen(L_{k-1})
6 |   L_k := 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, θ - threshold

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

```
1 C_m := ∅
2 foreach s⃗ ∈ C_m do
3 |   CalculatePI(s⃗)
4 |   if PI(s⃗) ≥ θ then
5 |   |   Add s⃗ 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 θ allora la si accetta. Questa valutazione viene effettuata iterativamente per tutte le sequeze 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** $\overrightarrow{s_i} \in L_{m-1}$ **do**
3 **foreach** $\overrightarrow{s_j} \in L_{m-1} \wedge \overrightarrow{s_i} \neq \overrightarrow{s_j}$ **do**
4 **if** $\overrightarrow{s_i}[2] = \overrightarrow{s_j}[1] \wedge \overrightarrow{s_i}[3] = \overrightarrow{s_j}[2] \wedge ... \wedge \overrightarrow{s_i}[m] = \overrightarrow{s_j}[m-1]$ **then**
5 $\overrightarrow{s} := \overrightarrow{s_i}[1] \rightarrow \overrightarrow{s_i}[2] \rightarrow ... \rightarrow \overrightarrow{s_i}[m-1] \rightarrow \overrightarrow{s_j}[m-1]$
6 $I(\overrightarrow{s}[1]) := I(\overrightarrow{s_i}[1]) \wedge I(\overrightarrow{s}[2]) := I(\overrightarrow{s_i}[2]) \wedge ... \wedge$
 $I(\overrightarrow{s}[m-1]) := I(\overrightarrow{s_i}[m-1]) \wedge$
 $I(\overrightarrow{s}[m]) :=$
 CalculateNeighborhood($I(\overrightarrow{s_i}[m-1]), I(\overrightarrow{s_j}[m-1])$
7 Add $\overrightarrow{s}$ 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 finchè l'ultimo tipo della prima sequenza non è uguale al penultimo tipo della seconda sequenza.

Esempio:
$P_1 = \{A \rightarrow B \rightarrow C \rightarrow D\}$ (sequenza lunga $m-1$)
$P_2 = \{B \rightarrow C \rightarrow D \rightarrow E\}$ (sequenza lunga $m-1$)
$P_{new} = \{A \rightarrow B \rightarrow C \rightarrow D \rightarrow E\}$ (sequenza lunga m)

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 $I(\overrightarrow{s_i}[m-1])$;

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 CalculateNeighborhood$(I(\overrightarrow{s_i}[m-1]), I(\overrightarrow{s_j}[m-1]))$.

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

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

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

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

1 **return**
$min(PI(\overrightarrow{s}[1] \rightarrow \overrightarrow{s}[2] \rightarrow ... \rightarrow \overrightarrow{s}[m-1]), PR(\overrightarrow{s}[m-1] \rightarrow \overrightarrow{s}[m])$

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 $I(\overrightarrow{s_i}[m-1])$ come set di partenza e si considera il vicinato con l'ultimo elemento della seconda sequenza $I(\overrightarrow{s_j}[m-1])$.
Come si può da subito notare il carico coputazionale 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.

Algorithm 5: CalculateNeighborhood($I(\vec{s_i}[m-1]), I(\vec{s_j}[m-1])$)

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

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

1 **return** $I(\overrightarrow{s}[i])$ as $distinct(\bigcup_{e \in I(\overrightarrow{s_i}[m-1])} N_{\overrightarrow{s_j}[m-1]}(e))$

Certamente un threshold θ alto ci permette di ignorare diverse combinazioni, però è un parametro handmade. Pertanto se si sceglie un θ troppo alto si rischia di avere pochi risultati con PI molto alto, se si sceglie un θ troppo bassso 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 θ 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 orientato:
Un **albero orientato** è un grafo connesso, aciclico ed orientato. 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 entranti, quindi nessun arco uscente.

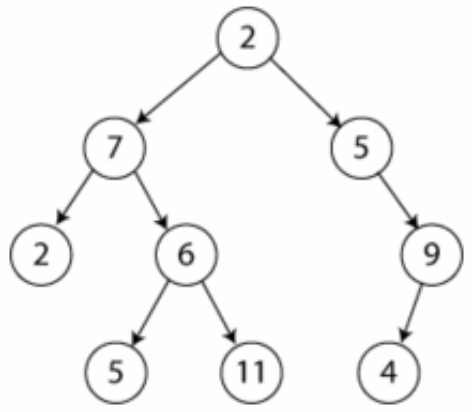

Figura 3.1: esempio di albero ordinato

3.2.2 Sequence Pattern Tree

L'albero utilizzato nel lavoro svolto è chiamato **Sequence Pattern Tree** ($SPTree$).
Si consideri 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	$A \to B$, $B \to C$, $B \to D$, $C \to E$, $C \to F$.
L_3	$A \to B \to C$, $A \to B \to D$, $B \to C \to E$, $B \to C \to F$.
L_4	$A \to B \to C \to E$, $A \to B \to C \to F$.

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 $\overrightarrow{s}$ è stata creata unendo i tipi di eventi f_{i_1} e f_{i_2} a $\overrightarrow{s} = f_{i_1} \to f_{i_2}$, allora:
  $firstParent(\overrightarrow{s}) := f_{i_1}$
  $secondParent(\overrightarrow{s}) := f_{i_2}$
  $\overrightarrow{s}$ viene aggiunta ai $children(\overrightarrow{s})$

- se la sequenza $\overrightarrow{s}$ è stata creata dalle sequenze $\overrightarrow{s}_i$ e $\overrightarrow{s}_j$, allora:
  $firstParent(\overrightarrow{s}) := \overrightarrow{s}_i$
  $secondParent(\overrightarrow{s}) := \overrightarrow{s}_j$
  $\overrightarrow{s}$ viene aggiunta ai $children(\overrightarrow{s})$

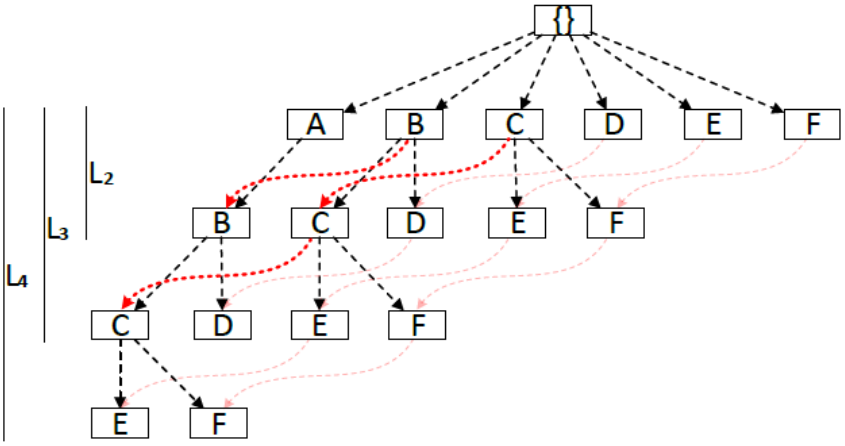

Figura 3.2: SPTree basato sulle sequenze della Tabella 3.1

Esempio
Consideriamo la sequenza $A \rightarrow B$ dalla Tabella 3.1 e quindi la sua memorizzazione nel *SPTree* come in Figura 3.2. 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 codizioni 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 $\overrightarrow{s}$ è l'N-esima sequenza pattern, se esistono $N-1$ sequenze nel insieme dei Top con *participation index* maggiore o uguale al PI($\overrightarrow{s}$).

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 $\overrightarrow{s_i}$ e il suo corrispondente nodo, si estrae una sequenza candidato $\overrightarrow{s_j}$ nel insieme *children* del suo *secondParent* (riga 3).
Il ruolo del *secondParent* è quello di verificare la presenza di possibili prolungamenti della sequenza $\overrightarrow{s_i}$ presa in esame. Come si può notare dall'albero di esempio in Figura 3.1 si può generare $A \rightarrow B \rightarrow C$ ($\overrightarrow{s}$) partendo da $A \rightarrow B$ ($\overrightarrow{s_i}$) aggiungendo la sequenza $B \rightarrow C$ ($\overrightarrow{s_j}$) presente nei $children(secondParent(\overrightarrow{s_i}))$.
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 $\overrightarrow{s_i}$, $\overrightarrow{s_j}$ viene creato il nuovo nodo per la nuova sequenza $\overrightarrow{s}$ che avrà come $firstParent(\overrightarrow{s}) := \overrightarrow{s_i}$ e come $secondParent(\overrightarrow{s}) := \overrightarrow{s_j}$ (riga 6), queste due proprietà garantiscono la possibilità della ricerca di un prolungamento ulteriore dell'albero. Infine si aggiunge ai *children* di $\overrightarrow{s_i}$ il nodo creato (in quanto $\overrightarrow{s_i}$ 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** $\overrightarrow{s_i} \in L_{m-1}$ **do**
3 **foreach** $\overrightarrow{s_j} \in children(secondParent(\overrightarrow{s_i}))$ **do**
4 $\overrightarrow{s} := \overrightarrow{s_i}[1] \rightarrow \overrightarrow{s_i}[2] \rightarrow ... \rightarrow \overrightarrow{s_i}[m-1] \rightarrow \overrightarrow{s_j}[m-1]$
5 $I(\overrightarrow{s}[1]) := I(\overrightarrow{s_i}[1]) \wedge I(\overrightarrow{s}[2]) := I(\overrightarrow{s_i}[2]) \wedge ... \wedge$
 $I(\overrightarrow{s}[m-1]) := I(\overrightarrow{s_i}[m-1]) \wedge$
 $I(\overrightarrow{s}[m]) :=$
 CalculateNeighborhood($I(\overrightarrow{s_i}[m-1]), I(\overrightarrow{s_j}[m-1])$
6 $firstParent(\overrightarrow{s}) := \overrightarrow{s_i}, secondParent(\overrightarrow{s}) := \overrightarrow{s_j}$
7 Add $\overrightarrow{s}$ to $children(\overrightarrow{s_i})$, Add $\overrightarrow{s}$ to C_m
8 **return** C_m

Algorithm 7: VerifyCandidates

Data: C_m - insieme di candidati sequenze di tipi di lunghezza m, θ - threshold
Result: L_m - insieme di sequenze di tipi significativi di lunghezza m

1 $C_m := \emptyset$
2 **foreach** $\overrightarrow{s} \in C_m$ **do**
3 CalculatePI($\overrightarrow{s}$)
4 **if** $PI(\overrightarrow{s}) \geq \theta$ **then**
5 Add $\overrightarrow{s}$ to L_m
6 **else**
7 Remove $\overrightarrow{s}$ 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 θ 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 $\overrightarrow{s}$ 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 θ 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 θ 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 θ, allora θ viene settato al valore del PI dell'N-esima sequenza top attuale e tutte le sequenze con PI minore del nuovo θ 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, θ - threshold

```
1  L_m := ∅
2  foreach s⃗ ∈ C_m do
3      CalculatePI(s⃗)
4      if PI(s⃗) ≥ θ) then
5          if |Top| < N − 1 then
6              Add s⃗ to L_m, add s⃗ to Top
7          else if |Top| = N − 1 then
8              Add s⃗ to L_m, add s⃗ to Top
9              θ := PI(Top(N))
10         else
11             Add s⃗ to L_m, add s⃗ to Top
12             if PI(s⃗) > θ) then
13                 θ := PI(Top(N))
14                 Delete all s⃗ ∈ Top with PI(s⃗) < θ
15                 Delete all s⃗ ∈ L_m with PI(s⃗) < θ
16                 Delete s⃗ from children(firstParent(s⃗))
17     else
18         Remove s⃗ from children(firstParent(s⃗))
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 θ 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

L'algoritmo proposto, non è l'unica alternativa allo studio di regole di associazioni. Un altro algoritmo di ricerca di associazioni molto valido è l'*algoritmo apriori*.

L'algoritmo apriori è utilizzato per la generazione di *itemset* frequenti a partire da itemset con un solo elemento. Concretamente parte da una serie di itemset frequenti e vi aggiunge elementi verificando quanto questa frequenza venga mantenuta.
L'algoritmo si basa sulla considerazione che se un insieme di oggetti (itemset) è frequente, allora anche tutti i suoi sottoinsiemi sono frequenti, viceversa se un itemset non è frequente, allora neanche gli insiemi che lo contengono sono frequenti, esso è definito **principio di anti-monotonicità**.

Viene applicato soprattutto per il *problema del carrello della spesa*, ovvero lo studio delle correlazioni tra prodotti acquistati al supermercato. In questo tipo di situazioni la generazione del gruppo di candidati viene svolta aggiungendo un item per volta come quando si mettono i prodotti della spesa nel carrello. I gruppi di candidati sono successivamente valutati e l'algoritmo termina quando non ci sono ulteriori estensioni possibili, similmente a come avviene per lo *STBFM*.

La generazione di itemset candidati consiste nella costruzione di un insieme ordinato di k elementi, a partire da un itemset di grandezza $k-1$.
Esempio
itemset di partenza:

$A \to B \to C$

$A \to B \to D$

itemset generati:

$A \to B \to C \to D$

$A \to B \to D \to C$

Ogni itemset così creato viene valutato all'interno del dataset, concetamente contando il numero delle istanze delle quali fa parte. Se il numero delle istanze supera un threshold ϵ, l'itemset viene considerato per la generazione di itemset di lunghezza maggiore. Il processo è ripetuto finchè non è

più possibile generare itemset significativi, rispettando il *principio di anti-monotonicità*.
Nel dettaglio vedi **Algorithm 9**.

Algorithm 9: Algoritmo apriori

Data: T - dataset delle transazioni, ϵ threshold di significatività

$L_1 := \{$ large $1 - itemset\}$
$k := 2$
while $L_{k-1} \neq \emptyset$ **do**
    $C_k :=$ Generate(L_{k-1})
    foreach $t \in T$ **do**
        $C_t :=$ Subset(C_k, t)
        foreach $c \in C_t$ **do**
            $count[c] := count[c] + 1$
    $L_k := \{c \in C_k | count[c] \geq \epsilon\}$
    $k := k + 1$
return $\bigcup_k L_k$

Considerazioni Come possiamo notare questo algoritmo, con alcune dovute modifiche, è applicabile al nostro problema. Va notato, però, che non vi è alcuna ottimizzazione per quanto riguarda la generazione di candidati, inoltre il threshold utilizzato (ϵ) è fisso.
Per questi motivi, per il corrente caso di studio, si preferisce utilizzare lo $STBFM$.

Capitolo 4

Dataset e Implementazione

La sezione seguente descrive il dataset del Boston Police Department (BPD) e l'implementazione dello Spatio-Temporal Breadth-First Miner applicato ad esso.

4.1 Dataset

Il dataset utilizzato si basa sulla raccolta dati del Department of Innovation and Technology del Boston Police Department. In questo database sono registrati tutti i reati rilevati sul campo dalla polizia di Boston dal giugno 2015 fino ad oggi. È un servizio che viene quotidianamente aggiornato con le nuove segnalazioni.
Il dataset si compone di 17 colonne descritte singolarmente in Tabella 4.1.

Per l'algoritmo oggetto del lavoro non sono necessari tutti questi dati, le informazioni fondamentali sono:

- chiave di intentificazione unica
- location spaziale
- istante temporale
- tipologia di evento

N	Campo	Tipo di dato	Descrizione
1	INCIDENT_NUMBER	varchar(20)	numero di report interno del BPD nonchè **chiave primaria**
2	OFFENSE_CODE	varchar(25)	numero che codifica la descrizione dell'evento
3	OFFENSE_CODE_GROUP	varchar(80)	categoria di evento criminale (**tipologia**)
4	OFFENSE_DESCRIPTION	varchar(80)	descrizione sommaria dell'evento
5	DISTRICT	varchar(10)	distretto cittadino
6	REPORTING_AREA	varchar(10)	area dalla quale è stato segnalato l'evento
7	SHOOTING	char(1)	indica se vi è stata una sparatoria
8	OCCURRED_ON_DATE	datetime(7)	data e ora in cui l'evento ha preso luogo
9	YEAR	integer(8)	anno in cui avviene l'evento
10	MONTH	integer(8)	mese in cui avviene l'evento
11	DAY_OF_WEEK	varchar(25)	giorno della settimana in cui avviene l'evento
12	HOUR	integer(8)	ora in cui avviene l'evento
13	UCR_PART	varchar(25)	Universal Crime Reporting part, codice: 1,2,3
14	STREET	varchar(50)	nome della via in cui l'evento è avvenuto
15	LATITUDE	varchar(20)	latitudine del luogo dell'evento
16	LONGITUDE	varchar(20)	longitudine del luogo dell'evento
17	LOCATION	varchar(50)	latitudine e longitudine del luogo dell'evento

Tabella 4.1: attributi database BPD

La chiave di identificazione del evento si può ricavare in INCIDENT_NUMBER, in quanto corrisponde alla chiave primaria del database e garantisce l'univocità di valore di un evento rispetto a tutti gli altri.
La location spaziale si acquisisce dalle colonne LATITUDE e LONGITUDE. Le coordinate sono presenti nel formato digitale.
L'istante temporale viene ottenuto dall'attributo OCCURRED_ON_DATE. Il dato è registrato nel seguente formato: yyyy-MM-dd hh:mm.
Ad esempio: 2018-09-01 21:28:00
La tipologia di evento si ricava dalla colonna OFFENCE_CODE_GROUP. L'attributo definisce le diverse tipologie di crimini, pertanto è l'informazione in questo campo. Non è stato scelto l'attributo OFFENCE_CODE in quanto le descrizioni sono troppo specifiche e poco generiche come dovrebbero essere le tipologie di eventi.
Inoltre si è scelto di analizzare i crimini di particolare gravità per cercare di avere dei risultati più utili e visibili possibili. Nel database i crimini sono suddivisi in tre macro categorie denominate UCR part (Uniform Crime Reporting), seguendo uno standard americano di raccolta dati criminali. Queste categorie sono ordinate per gravità: la terza categoria comprende i reati lievi quali liti e frodi, la seconda gli illeciti più gravi come vandalismo e spaccio, la prima i reati molto gravi come furto con scasso, aggressione grave e omicidio.

Nei test effettuati si utilizzano le tipologie di crimini con UCR Part = 1, e sono i seguenti 9:

- Aggravated Assault - aggressione aggravata
- Auto Theft - furto d'auto
- Commercial Burglary - furto con scasso commerciale
- Homicide - omicidio
- Robbery - rapina
- Larceny - furto
- Residential Burglary - furto con scasso residenziale
- Larceny From Motor Vehicle - furto da autoveicolo
- Other Burglary - altro furto con scasso

Preprocessing Prima di procedere all'applicazione dell'algoritmo sul database è stata effettuata una fase di *data preprocessing* con lo scopo di eliminare i dati inconsistenti e lavorare solo su quelli significativi e coerenti.
Sono stati eliminati i record con le informazioni di LATITUDINE e LONGITUDINE vuote o non valide. Sono stati eliminati valori di INCIDENT_NUMBER duplicati.
Il periodo di tempo degli eventi scelto è l'anno 2018, perchè degli anni precedenti mancano riferimenti ad alcuni mesi. Questa mancanza di dati avrebbe potuto condizionare gli esiti dei test, in quanto è risaputo che la frequenza di certe tipologie di eventi criminali è soggetta a variazioni in base alla stagione e al periodo dell'anno.

Dopo tutte queste assunzioni i record cosiderati per il database del 2018 sono: 72 135.
Di questi si è deciso di considerare, per significatività e compattezza nei risultati, i tipi con UCR part = 1, ovvero i reati di più alta gravità, concretamente **15 648 record** di eventi criminali.
Come chiaro che sia, i tipi di eventi non sono ugualmente frequenti, si può notare in Figura 4.1 (di seguito) l'evento più rilevato è *larceny* (furto), mentre quello di intesità minore è *homicide* (omicidio).

Le colonne della versione finale del database sono quelle di
INCIDENT_NUMBER, OFFENCE_CODE_GROUP, OCCURRED_ON_DATE, LATITUDE, LONGITUDE.
In Tabella 4.2 un esempio di alcuni record del dataset:

incident_number	offence_code_group	occurred_on_date	latitude	longitude
I192014076	Larceny	20/12/2018 00:01	4.232.386.050	-7.107.600.196
I192003189	Aggravated Assault	30/12/2018 12:00	4.233.596.847	-7.108.129.856
I192013746	Larceny	16/11/2018 12:00	4.232.809.966	-7.106.321.676
I192013689	Auto Theft	28/12/2018 10:07	4.235.988.372	-7.106.016.189

Tabella 4.2: esempio di record nel dataset dopo preprocessing

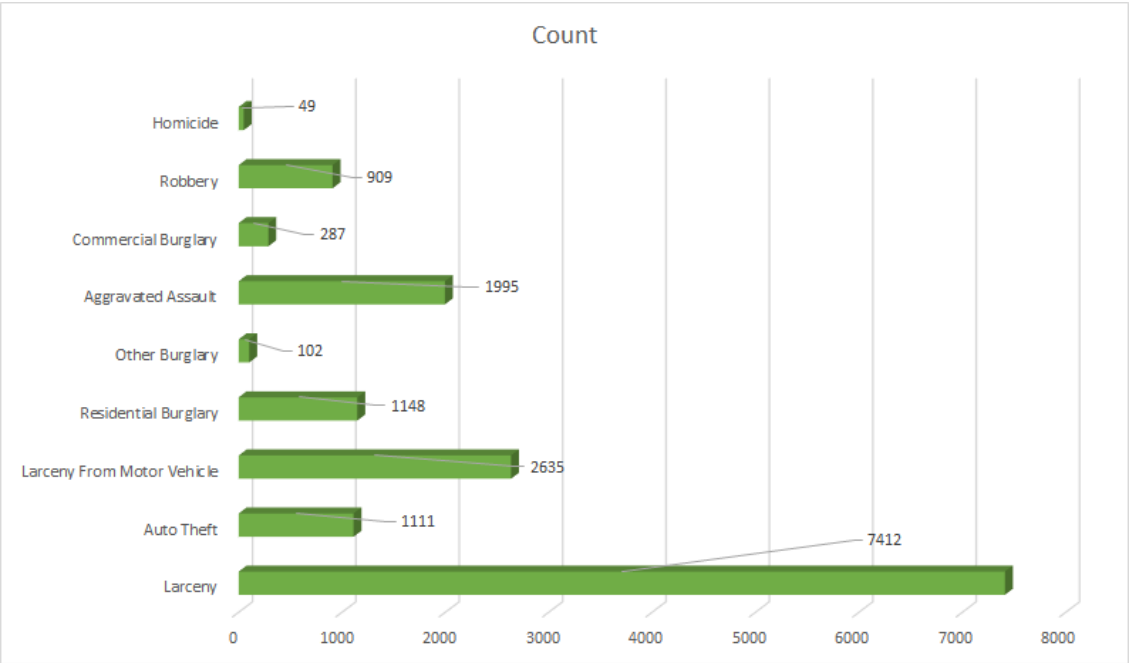

Figura 4.1: frequenze eventi per tipologia nel 2018

4.2 Implementazione

L'implementazione dell'algoritmo Spatio-Temporal Breadth-First Miner è stata svolta in linguaggio Python. Sono state sperimentate diverse soluzioni, di seguito vi è la spiegazione del funzionamento della versione migliore nonchè più affine all'algoritmo presentato precedentemente.

Il software è diviso in due file *SPTree.py* ed *STBFM.py*. Il primo definisce le classi *Node* ed *SPTree*, responsabili della generazione e gestione della struttura ad albero. Il secondo contiene tutti i metodi e funzioni che permettono la corretta computazione dell'algoritmo.

4.2.1 Node

La classe *Node* definisce la struttura dati che viene utilizzata per definire i nodi del SPTree.
La struttura è la seguente:

parent1	parent2	value	set (dizionario)	children (lista)

Il **parent1** e il **parent2** sono reference ad oggetti Node connessi al nodo attuale corrispondenti rispettivamente al $firstParent$ ed al $secondParent$ definiti nell'albero.

Il **value** corrisponde al tipo di evento del nodo corrente, che varia in base alle sequenze generate e alla posizione dello stesso. Indicativamente il value corrisponde all'ultimo valore della sequenza fin qui generata. *esempio* in una sequenza totalmente inserita $A \rightarrow B \rightarrow C$ il nodo più profondo avrà come $value = C$.

Il **set** si riferisce al *set di istanze* calcolato fino a quel punto nella sequenza, viene salvato nel nodo in modo da non perdere l'informazione quando si dovranno analizzare le sequenze di lunghezza maggiore, vedi riga 5 dell'Algorithm 6. L'insieme è memorizzato in un dizionario in Python, che è la struttura dati perfetta per questo genere di informazione, perchè ogni istanza evento inserita viene indicizzata attraverso l'*incident_number*. In questo modo ogni tupla inserita risulta univoca e non duplicata nel set. Inoltre le

memorizza direttamente in RAM rendendo necessario l'accesso a memoria secondaria (estremamente costoso) solo a una prima lettura dei dati e non ad ogni iterazione dell'algoritmo.

Il campo **children** contiene una lista di reference ad oggetti Node che sono considerati "figli" del nodo corrente. La struttura della lista è stata scelta perchè dinamica e comoda nell'aggiunta ed eliminazione di elementi.

4.2.2 SPTree

La classe SPTree si occupa di modellare il Sequence Pattern Tree. Come struttura del nodo di base utilizza la classe *Node*.
Seguendo il modello di struttura dati originale, l'SPTree ha un solo attributo di base, la **root**. Essa è un *Node* con tutti i parametri inizialmente settati a **None**. Con l'inserimento di ciascun nodo di primo livello di albero, viene riempito il campo children della root con i reference al nodo stesso.
L'**inserimento** viene gestito dal metodo *insertNode()* che, data una sequenza in input e il suo set di istanze, genera il nuovo nodo. Il nuovo nodo corrisponde all'ultimo elemento della sequenza in input, e viene posizionato nell'albero seguendo il percorso della stessa.
L'**eliminazione** di un nodo viene gestita dal metodo *deleteNode()* che, data in input una sequenza, elimina l'ultimo nodo corrispondente, permettendo il *pruning* di tutto il sottoalbero presente nei children di quel nodo.
Inoltre è stato definito un **metodo ausiliario** *searchNode()* che, data una sequenza in input, restituisce il nodo corrispondete all'ultima posizione della stessa.

4.2.3 STBFM

Il corpo dell'algoritmo vero e proprio è contenuto nel file *STBFM.py*.
Il software rispecchia il più possibile l'algoritmo originale definito nel Capitolo 3 utilizzando la classe *SPTree*. Vanno fatte comunque delle premesse.

La distanza spaziale, avendo in input coordinate GPS, non può considerarsi perfetta in quanto la superficie terrestre non è regolare e il nostro pianeta non può essere considerato una sfera. Il metodo di calcolo della distanza

scelto è quello della *legge sferica dei coseni*:

date (φ_1, λ_1) e (φ_2, λ_2) due coordinate in radianti (latitudine, longitudine) la distanza tra di esse in km è:

$$d = \arccos(\sin(\varphi_1) \cdot \sin(\varphi_2) + \cos(\varphi_1) \cdot \cos(\varphi_2) \cdot \cos(\Delta\lambda)) \cdot R$$

dove $R = 6371[km]$ è il raggio della Terra e $\Delta\lambda$ la differenza tra λ_1 e λ_2.
L'applicazione di altri metodi di calcolo per la distanza probabilmente modifica i risultati anche se di lieve intensità.

Un'altra premessa degna di nota è la generazione delle sequenze candidato di lunghezza 2. Il software genera tutte le combinazioni possibili di sequenze *randomicamente*. Questo implica che la generazione dell'albero non avviene sempre nello stesso ordine e quindi valida ulteriormente i risultati ottenuti, in particolare per quanto riguarda i tempi di computazione che restano indipendenti dall'ordine delle sequenze candidato scelto.

In generale, il resto del software rispetta tutti i passaggi visti nell'Algorithm 1, 6 e 8, adattandoli al linguaggio di programmazione Python.

L'**input** al software avviene per riga di comando, i parametri da inserire sono: threshold θ, numero di risultati nel Top, raggio spaziale ed intervallo temporale (in ore).
esempio:

- > python3 STBFM.py 0.25 50 3 168

in questo caso richiedo una computazione con $\theta = 0.25$, num$Top = 50$, raggio $R = 3km$, intervallo tempo $T = 168$ ore (7 giorni).

Nota Bene è possibile, dati i parametri, eseguire la computazione in più dataset. I dataset utilizzati sono dei file in formato .csv e sono dichiarati esplicitamente all'interno del codice.

L'**output** è di due tipi:

- output da terminale che, man mano nella computazione, mostra i risultati intermedi e lo svolgimento delle operazioni, nonchè eventuali errori;

- output su file.csv dove al termine della computazione di un dataset stampa l'insieme Top e i tempi di computazione per livello.

L'output in csv è stampato nel seguente formato:
nomeFile, R, T, numTop, θ finale, $[tempo2, tempo3, ..., tempoN]$, $tempoTot$
dove $[tempo2, tempo3, ..., tempoN]$ è una lista che contiene i tempo di calcolo divisi nei vari livelli di albero.
esempio

dataset18_5216.csv, 2,168,0.57,[230,450,345,123],1148

Il codice del software con relativa guida all'utilizzo sono presenti sulla piattaforma di condivisione del codice GitHub al seguente indirizzo:
https://github.com/fedelux3/BostonCrime.

Capitolo 5

Analisi dei risultati

La performance del software è stata valutata tramite diversi test in cui vengono utilizzati database con diverso numero di record e diverse combinazioni di parametri.

Il database di partenza utilizzato è quello relativo al Boston Crime Department del 2018 descritto nel capitolo precedente con 15 648 record. Non è stato utilizzato però direttamente in quanto il numero di record è molto elevato. Con un alto numero di record il tempo di elaborazione aumenta eccessivamente, pertanto si è preferito testare il software su suoi sottoinsiemi.
Per la generazione dei dataset sottoinsiemi viene copiato un record ogni N, in questo modo si mantengono le proporzioni delle frequenze tra i diversi tipi di eventi.
Sono stati generati 4 dataset sottoinsieme ed hanno i seguenti numeri di record:

1. 1043 - 1 record ogni 15
2. 2235 - 1 record ogni 7
3. 3129 - 1 record ogni 5
4. 5216 - 1 record ogni 3

I parametri utilizzati sono stati: θ, numero di sequenze top (Num), R, T.

Il threshold θ è stato fissato a 0.25, in quanto è il valore consigliato dal paper. Essendo un valore basso, l'algoritmo esclude a priori le sequenze

poco significative e si adatta, incrementandolo, alle sequenze considerate nell'insieme Top nel caso in cui il loro numero superi il massimo consentito.

Il numero di sequenze top Num da considerare nel insieme Top di output è stato fissato a 50, ovvero come output si considerano solo le prime cinquanta sequenze di tipi per *participation index*. Il valore è frutto del compromesso tra numero di sequenze significative da visualizzare e tempi di computazione. Se si scegliesse un valore minore non si avrebbero sequenze di tipi significativi probabilmente utili. Se si segliesse un valore maggiore i tempi di computazione lieviterebbero esponenzialmente, inoltre le sequenze ricavate oltre la cinquantesima posizione avranno, con alta probabilità, un PI basso, quindi poco rilevanti.

I parametri di **raggio spaziale** R e **intervallo temporale** T sono variabili che vengono fissante in base al test da effettuare.
Sono state scelte 10 combinazioni diverse di questi due parametri, con R che varia da 1 a 3 km e T che varia da 72 a 168 ore (da 3 giorni a 7 giorni).

5.1 Tempi di computazione

La prima parte dell'analisi fatta riguarda i tempi di computazione e in particolare lo studio di come scala all'aumentare dei record il tempo di elaborazione. Sono state scelte 12 combinazioni dei parametri R e T, applicate a tutti i quattro dataset. In concreto vi sono $12 \times 4 = 48$ test, 12 combinazioni per 4 dataset.
In Tabella 5.1 vengono mostrati i test effettuati con i tempi di computazione, suddivisi in base a: raggio spaziale, intervallo temporale e numero di record.

test	**raggio spaziale**	**intervallo temporale**	**tempo per numero di record**			
			1043	**2235**	**3129**	**5216**
1	2	168	10	87	198	693
2	1.5	168	7	50	158	538
3	1.5	120	7	42	116	475
4	3	120	13	105	247	815
5	1	120	7	30	71	234
6	1	72	7	30	60	182
7	2	120	8	68	181	624
8	3	72	9	81	204	668
9	1.5	72	7	33	74	307
10	3	168	20	117	267	818
11	2	72	7	43	115	492
12	1	168	7	32	85	362

Tabella 5.1: tempi in secondi dei test effettuati

Per comprendere come il software scali per numero di record vengono fissati i parametri di raggio spaziale R e intervallo temporale T, in questo modo è possibile valutare l'andamento del tempo di computazione legato alla sola variazione di numero di record. Seguendo la procedura per tutte le combinazioni di raggio e tempo utilizzate ne risulta il grafico in Figura 5.1.
Una linea del grafico rappresenta una serie di test effettuati sui tutti i quattro dataset fissando una coppia di parametri R e T. Le serie di test vengono effettuate per tutte le 12 combinazioni e i risultati rappresentati in figura.

Come era prevedibile i test che impiegano più tempo ad essere elaborati sono quelli con *neighborhood* più ampio sia per quanto riguarda il raggio spaziale sia quello temporale.
L'altro aspetto lampate è l'andamento **non lineare** del tempo all'aumentare

del numero di record. Al contrario, con un numero sufficientemente alto di record, si può pensare che il software tenda ad un tempo esponenziale. Lo si evince soprattutto quando il numero di record sale da 3129 a 5216.
Da notare inoltre che per un *vicinato* più ristretto, l'incremento dei tempi legato al numero di record è meno marcato, ad esempio per $R = 1 - T = 72$.

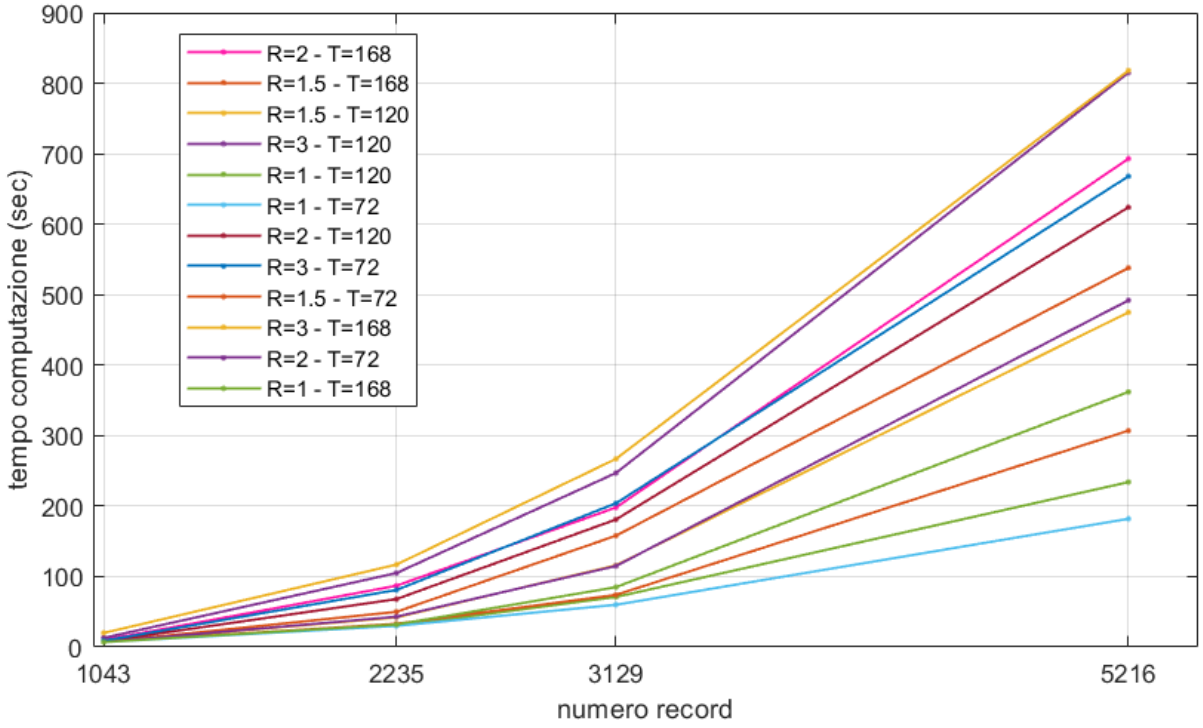

Figura 5.1: tempi di computazione in relazione al numero di record

Bibliografia

[1] AnkurJain. Kaggle - crimes in boston. https://www.kaggle.com/ankkur13/boston-crime-data.

[2] Department, B. P. Crime incident reports (august 2015 - to date). https://data.boston.gov/dataset/crime-incident-reports-august-2015-to-date-source-new-system/resource/12cb3883-56f5-47de-afa5-3b1cf61b257b.

[3] FBI. Ucr - uniform crime reporting. https://www.bjs.gov/ucrdata/offenses.cfm.

[4] IlSole24ore. Big data, tutti i numeri che contano. https://www.infodata.ilsole24ore.com/2015/11/02/big-data-tutti-i-numeri-che-contano/, novembre 2015. pagine 12-13.

[5] Luzzi, F. Bostoncrime. https://github.com/fedelux3/BostonCrime.

[6] Maciag., P. S., and Bembenik., R. A novel breadth-first strategy algorithm for discovering sequential patterns from spatio-temporal data. In *Proceedings of the 8th International Conference on Pattern Recognition Applications and Methods - Volume 1: ICPRAM,* (2019), INSTICC, SciTePress, pp. 459–466.

[7] Scripts, M. T. Calculate distance, bearing and more between latitude/longitude points. http://www.movable-type.co.uk/scripts/latlong.html.

[8] Thomas H. Cormen, Charles E. Leiserson, R. L. R. C. S. *Introduzione agli algoritmi e strutture dati - 3a edizione*. 2010. Appendice B.5 - pag. 974-981.

[9] Wikipedia. Regole di associazione. //it.wikipedia.org/w/index.php?title=Regole_di_associazione&oldid=99576184, 2018.

[10] Wikipedia. Albero (informatica). //it.wikipedia.org/w/index.php?title=Albero_(informatica)&oldid=105897217, 2019.

[11] Wikipedia. Algoritmo apriori. //it.wikipedia.org/w/index.php?title=Algoritmo_apriori&oldid=104408711, 2019.

[12] Wikipedia. Data mining. http://it.wikipedia.org/w/index.php?title=Data_mining&oldid=105051611, 2019.