UNIVERSITÀ
POLITECNICA
DELLE MARCHE

Facoltà di Ingegneria
Corso di Laurea Magistrale in Ingegneria Informatica e dell'Automazione

---

A Spark implementation of the Building Instance Graph algorithm for the analysing of Big event logs

Candidato:
Jacopo Iezzi

Relatore:
Prof. Domenico Potena

Anno Accademico 2021-2022

UNIVERSITÀ
POLITECNICA
DELLE MARCHE

Facoltà di Ingegneria
Corso di Laurea Magistrale in Ingegneria Informatica e dell'Automazione

---

Implementazione in Spark dell'algoritmo Building Instance Graph per l'analisi di Big event log

Candidato:
Jacopo Iezzi

Relatore:
Prof. Domenico Potena

Anno Accademico 2021-2022

Una promessa è un impegno, è il mettersi ancora in corsa, è il non sedersi su quel che si è fatto.
Dà nuove responsabilità, obbliga a cercare, a trovare nuove energie.
Gino Strada

Abstract

Al giorno d'oggi, molte aziende hanno implementato dei sistemi informativi per tracciare e facilitare l'esecuzione dei vari processi aziendali. Questi sistemi, grazie al progresso tecnologico e all'aumentare delle applicazioni, producono una notevole quantità di dati che vanno a rappresentare come vengono eseguiti i vari processi. Le esecuzioni di questi ultimi sono descritte all'interno dei file come un insieme di tracce che contengono a loro volta gli eventi. Le tracce rappresentano una singola istanza del processo mentre gli eventi rappresentano le attività svolte per ciascuna traccia.
Il process mining è una tecnica di process management, che permette l'analisi dei processi di business basati su event log. Attraverso l'uso di specifici algoritmi applicati ai log degli eventi è possibile scoprire molte informazioni riguardanti un sistema informativo.
L'obiettivo del process mining, infatti, è di migliorare quest'ultimo, fornendo tecniche e strumenti per la scoperta di processi, di di organizzazioni e strutture sociali a partire dai log.
Uno di questi algoritmi è il Building Instance Graph (BIG), che permette di estrarre un modello di processo relativo ad una sua variante. L'implementazione attuale dell'algoritmo è stato realizzato in python grazie all'utilizzo di varie librerie. Questo script elabora il file di log sequenzialmente, cioè prendendo una singola traccia alla volta, applicando l'algoritmo per ciascuna di esse. Questa scelta risulta essere poco efficiente nel caso in cui andassimo ad analizzare dei file che abbiano al loro interno un numero elevato di tracce.
L'obiettivo è di potenziare le performance andando ad eseguire l'algoritmo per ciascuna traccia non più sequenzialmente ma parallelamente. Per far ciò si è utilizzato Apache SPARK, framework open source per il calcolo distribuito, sia per il caricamento del file di log, sia per la sua elaborazione.
Grazie all'utilizzo del framework è stato possibile partizionare il file di log per singola traccia e successivamente eseguire l'algoritmo per ciascuna di esse. Il seguente documento è suddiviso in 4 capitoli:

- Process Mining: In questo capitolo verrà spiegato cos'è il process mining, andando ad specializzarci sull'algoritmo BIG, analizzando i suoi passaggi. Verrà poi illustrato il codice dell'algoritmo analizzando le sue criticità:

- BIG for Big Event Log: In questo capitolo verrà esposta la soluzione creata partendo dalle criticità dell'applicativo del capitolo precedente, illustrando le varie tecncologie usat:.

- Testing: In questo capitolo verranno esposti le varie tipologie di test eseguiti, illustrando i risultati ottenuti:
- Conclusioni: Andremo a commentare i risultati precedenti

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

Process Mining

Oggigiorno molte aziende utilizzano i sistemi informativi, ad esempio WMS, CRM o ERP, per gestire i loro processi. Questi sistemi producono delle informazioni (log), che descrivono le singole attività dei singoli processi e che permettono ad ogni processo di essere rappresentato come una sequenza di attività o di eventi. Questi event_log,o semplicemente dataset, sono composti da un insieme di tracce che rappresentano le varie istanze del processo, costituiti al loro interno da eventi che indicano le varie attività che costituiscono il processo stesso. La sequenza di attività definite in fase di progettazione del processo rappresenta il modello del processo stesso. Grazie all'utilizzo intensivo di questi sistemi, la disponibilità e le dimensioni dei file è aumentato a dismisura animando la ricerca scientifica nel trovare le migliori strategie per trovare il valore aggiunto nell'analisi di questi log.
Da questo spunto è nato il **Process Mining (PM),** una nuova disciplina che ha come scopo di migliorare le tecniche di analisi di questi file. Nelle organizzazioni, gli utenti che sono coinvolti nei vari processi, possono eseguire le varie attività anche non rispettando la sequenza definita dal modello, andandone a variare la sequenza ottenendo come risultato una variazione dallo schema formato dalle best practice aziendali. Nonostante le migliorie che questi potrebbero apportare, la loro analisi risulta essere complessa non permettendo di comprendere come il processo viene effettivamente eseguito.
Attualmente, gli approcci proposti dal PM risultano avere importanti svantaggi quando bisogna affrontare processi altamente variabili, poiché introducendo un alto grado di variabilità nell'esecuzione del processo, implicherà che il dataset corrispondente coinvolgerà comportamenti eterogenei. Questi log, nei processi fortemente variabili, porteranno a generare dei modelli caotici e molto complessi, chiamati "spaghetti-like". Negli ultimi anni molti ricercatori hanno lavorato per sviluppare tecniche di analisi per queste tipologie di modelli, creando diverse approcci.
Una di queste tipologie consiste nell'applicare tecniche di pattern discovery (PD), per estrarre le porzioni più rilevanti dei comportamenti di processo. Il pattern discovery sono delle tecniche che mirano alla ricerca di schemi o modelli all'interno di un set di dati. Questa ricerca è diventata uno dei task più importanti del data mining e può essere applicata in diversi contesti. Nel nostro caso vogliamo trovare i sotto-processi più rilevanti, piuttosto che ricercare modelli di processi completi. L'idea alla base è

che anche in contesti dove gli attori hanno un alto grado di libertà e le esecuzioni di processi sono molto variabili, è possibile che esistano alcune tendenze o regolarità nel processo, che si esprimono come modelli ricorrenti. Quest'ultima è l'idea che si pone alla base del Building Instance Graph definiti nella tesi di dottorato della professoressa Laura Genga [11], o BIG come verrà menzionato successivamente nel documento. Big è una procedura per la costruzione di grafi partendo da un dataset che rappresenta i processi che si vuole porre in esame con l'aggiunta di un meccanismo di riparazione che migliora la qualità dei modelli costruiti. Nei prossimi paragrafi andremo ad analizzare i vari passaggi dell'algoritmo su cui si basa la procedura.

1.1 The IG Building Algorithm

L'algoritmo di costruzione degli instance Graph si pone come obbietivo di creare per ciascuna traccia presente nel file di log, una rappresentazione tramite grafo della traccia basandosi sul modello del processo preso in oggetto. L'algoritmo è strutturato in diversi passi come si può osservare dall'immagine 1.1. Il primo punto

Algorithm 1 Instance Graphs Building Algorithm.

Let $L = (E, C, act, pi, \preceq)$ be an event log

1: M=ApplyProcessDiscovery(L, PD);
2: CR= ExtractCausalRelation(M);
3: **for** $k = 1$ to $|C|$ **do**
4: IG_k=ExtractInstanceGraph(σ_k,CR);
5: $[D, I]$=CheckTraceConformance(σ_k,M);
6: **if** $D \cup I \neq \emptyset$ **then**
7: IG_k=IrregularGraphRepairing(IG_k,σ_k, D,I, CR);

Figura 1.1: Algoritmo del Building Instance Graph

dell'algoritmo è quello di estrarre il modello (M) del processo applicando una funzione di Process Discovery (PD) al nostro dataset (L). Ottenuto il modello, si procede con l'estrazione delle Casual Relation (CR) da esso. Queste relazioni casuali servono per collegare le varie attività contenute all'interno delle tracce ed esprimono le relazioni che intercorrono tra esse. Successivamente per ogni traccia contenuta all'interno del dataset, si costruisce l'Instance Graph (IG). L'instance Graph è la rappresentazione su grafo della traccia presente nel log, dove gli eventi della traccia sono i nodi e le CR estratte precedentemente servono per la costruzione degli archi. Il secondo passo del loop è di estrapolare due liste:

- **Inserted I**: Lista di attività contenute nel log ma non presenti nel modello
- **Deleted D**: Lista di attività non contenute nel log ma presenti nel modello

Se una di queste due liste contiene un elemento, implica che la traccia contenuta nel dataset ha dei cambiamenti rispetto al modello del processo stesso e l'IG creato nei

passi precedenti dovrà essere modificato andando ad inserire e/o eliminare i nodi contenuti all'interno delle liste. Questa attività viene effettuata con un procedura di riparazione chiamata IrregolareGraphRepairing.
L'algoritmo appena descritto può essere quindi diviso in due fasi:

- **IGinitialize**: in cui andremo a creare un'insieme di Instance Graph per ciascuna traccia presente all'interno del dataset;
- **Repair**: per ogni IG precedentemente creato, andremo ad applicare l'algoritmo di repair.

Vediamo nel dettaglio le varie fasi.

1.1.1 IGinitialize

In questa prima fase, per ogni traccia contenuta all'interno del nostro dataset andremo a costruire il grafo iniziale, cioè la rappresentazione del processo definito nel modello e la mappatura delle attività contenute all'interno della traccia rispetto a quelle dichiarate all'interno del suo modello.
L'algoritmo della prima fase è costituito da vari step che andremo ad analizzare

Algorithm 1 IG initialize

Require: $L, [P, AT]$
1: $IG = \emptyset; M = \emptyset$
2: **for each** $t \in L$ **do**
3: at=pick_aligned_trace($\pi_{id}(t)$)
4: m=get_mapping_(t, at)
5: at=generate_compliant_trace(at)
6: ig=build_ig(at)
7: $IG.add(ig); M.add(m)$
8: **end for**

Figura 1.2: Algoritmo di IGInitialize

passo passo:

- **Pick_alignment_trace (AT)**: in questo primo step, si vuole andare a mostrare quelle che sono le differenze tra le attività presenti all'interno del modello e quelle presenti all'interno della traccia del dataset. Per ogni evento presente sulla traccia ma non presente sul modello e viceversa, andremo ad inserire il simbolo "»", come si può notare dall'immagine 1.3

$$\eta_1 = \begin{array}{|c|c|c|c|c|c|c|c|} \hline \mathsf{a} & \gg & \mathsf{b} & \mathsf{c} & \mathsf{d} & \mathsf{e} & \gg & \mathsf{f} \\ \hline \mathsf{a} & \mathsf{k} & \mathsf{b} & \mathsf{c} & \gg & \mathsf{e} & \mathsf{m} & \mathsf{f} \\ \hline \end{array}$$

Figura 1.3: Esempio di un Alignment_trace

- **Mapping**: Partendo dall'AT, si vuole creare una mappatura delle posizioni dei vari eventi contenuti all'interno della traccia. Per fare ciò si è creata una tabella dove verranno riportate il nome dell'attività, la posizione all'interno della traccia del log (Pos Trace) e la posizione rispetto al modello (Pos Compl Trace). Per spiegare meglio questo step, prendiamo come esempio l'AT descritto dalla figura 1.3. Possiamo osservare che la traccia presa in esame presenti differenze rispetto al modello, come le attività 'k' ed 'm' che non hanno corrispondenza poichè sono nuove, mentre nella traccia non è presente l'attività 'd' che nel modello invece esiste. Per questo motivo, le attività 'k' ed 'm' verranno inserite in coda, per indicare che rappresentano delle aggiunte, come si può vedere dall'immagine 1.4

Actvity	**Pos Trace**	**Pos Compl Trace**
a	1	1
k	2	7
b	3	2
c	4	3
d		4
e	5	5
m	6	8
f	7	6

Figura 1.4: Esempio di un mapping basato sul AT precedenete

- **Generate_compliant_trace**: la compliant_trace è l'insieme di tracce e di relazioni estrapolate dal modello che servirà successivamente per la costruzione del grafo di partenza;

- **Build_ig**: si crea l'instance graph sulla base delle attività e delle relazioni estrapolate dal modello nello step precedente.

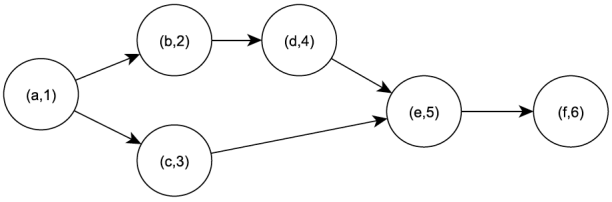

Figura 1.5: Esempio di IG

Terminati questi step, possiamo procedere con la fase di riparazione dei grafi per quelle tracce che presentano delle modifiche rispetto al modello.

1.1.2 Repair Algorithm

In questa fase, l'obiettivo è di andare a ricostruire il grafo precedentemente creato aggiungendo e rimuovendo le attività trovate all'interno delle tracce del dataset. Il primo passo sarà quello di creare le due liste di attività da aggiungere e da rimuovere, utilizzando l'instance graph e l'aligment set creati nella fase precedente. Note le due liste, si procederà con la riparazione del grafo aggiungendo e successivamente rimuovendo le attività. Esaminiamo nello specifico le varie fasi:

Algorithm 2 IG repair

Require: L, IG, AT

1: $IG_r = \emptyset$
2: **for each** $ig \in IG$ **do**
3: INS=pick_insertion_anomalies(ig, AT)
4: DEL=pick_deletion_anomalies(ig, AT)
5: **for each** $ins \in INS$ **do**
6: ig_r=repair_insertion(ig, ins)
7: **end for**
8: **for each** $del \in DEL$ **do**
9: ig_r=repair_deletion(ig, ins, m)
10: **end for**
11: ig_r =map_nodes(ig_r)
12: ig_r =trans_reduction(ig_r)
13: $IG_r.add(ig_r)$;
14: **end for**

Figura 1.6: IG Repair Algorithm

- **pick_insertion_anomalies**: partendo dall'AT, vogliamo estrarre tutte le anomalie da inserire. In output avremo una struttura di questo tipo (activity_name,pos_in, pos_fin), dove per activity_name intendiamo il nome dell'attività sulla traccia, pos_in la posizione all'interno della traccia e pos_fin la posizione all'interno della compliant trace;
- **pick_deletion_anomalies**: funzionamento simile al punto precedente, ma in questo caso, si andranno a considerare solo le attività da rimuovere;
- **Repair insertion**: In figura 1.7 possiamo esaminare nel dettaglio le varie fasi di questa procedura.

  Lo scopo di questa procedura consiste nell'aggiornare la tabella del mapping con le varie posizioni. Un esempio: immaginiamo di avere un inserimento (k,2,7). Il primo passo è di cercare, se esistono, all'interno del grafo dei nodi in quella posizione. Nel caso ci fossero, dobbiamo modificare le varie posizioni,cioè aggiornando quelli che sono gli archi collegati al quel nodo.
- **Repair Deletion**: In figura 1.8 possiamo vedere le varie fasi di questa procedura. Lo scopo è simile al repair inserction, andando ad aggiornare gli archi

Algorithm 4 Repair insertion

Require: ig, ins, m

1: $E_{rem_p} = PRED = SUCC = \emptyset$
2: **for each** $i \in [ins.pos_{in}, ins.pos_{fin}]$ **do**
3: $\quad ig.N = ig.N \cup \{i\}$
4: **end for**
5: $pos_t = m(ig, ins.pos_{in})$
6: **if** $\exists \langle (v_{pos_t-1}, v_x), \dots (v_y, v_{pos_t}) \rangle$ **then** ▷ C_7
7: $\quad E_{rem_p} = \{(v_x, (pos_t)) \mid (v_x, pos_t \in ig.E\}$
8: $\quad PRED$=get_predecessors(E_{rem_p})
9: $\quad$ **for each** $p \in PRED$ **do** ▷ C_8
10: $\quad\quad E_{rem_p} = \{(p, v_x) \mid (p, v_x) \in ig.E\}$
11: $\quad$ **end for**
12: $\quad PRED$=get_predecessors(E_{rem_p})
13: **else**
14: $\quad PRED = PRED \cup \{pos_t - 1\}$
15: $\quad E_{rem_p} = \{((pos_t - 1), v_x) \mid ((pos_t - 1), v_x) \in ig.E\}$
16: **end if**
17: $SUCC$ =get_successors(E_{rem_p})
18: $ig.E = ig.E \setminus E_{rem_p}$
19: **for each** $p \in PRED$ **do**
20: $\quad ig.E = ig.E \cup \{(p, ins.pos_{in})\}$
21: **end for**
22: **for each** $i \in [ins.pos_{in}, ins.pos_{fin} - 1)$ **do**
23: $\quad ig.E = ig.E \cup \{(i, (i + 1))\}$
24: **end for**
25: **for each** $s \in SUCC$ **do**
26: $\quad ig.E = ig.E \cup \{(ins.pos_{fin}, s)\}$
27: **end for**
28: **Return** ig

Figura 1.7: Repair Insertion Algorithm

delle attività da rimuovere.

Algorithm 3 Repair deletion

Require: ig, del

1: $E_{rem_p} = E_{rem_s} = PRED = SUCC = \emptyset$
2: **for each** $i \in [del.pos_{in}, del.pos_{fin}]$ **do**
3: $\quad E_{rem_p} = \{(v_x, i) \mid v_x, i \in ig.E\}$
4: $\quad E_{rem_s} = \{(i, v_x) \mid i, v_x \in ig.E\}$
5: $\quad PRED$=get_predecessors(E_{rem_p})
6: $\quad SUCC$=get_successors(E_{rem_s})
7: $\quad ig.E = ig.E \setminus (E_{rem_p} \cup E_{rem_s})$
8: $\quad ig.N = ig.N \setminus \{i\}$
9: $\quad$ **for each** $p \in PRED$ **do**
10: $\quad\quad$ **for each** $s \in SUCC$ **do**
11: $\quad\quad\quad ig.E = ig.E \cup \{(p, s)\}$
12: $\quad\quad$ **end for**
13: $\quad$ **end for**
14: **end for**
15: **Return** ig

Figura 1.8: Repair Deletion Algorithm

Dopo aver effettuato gli inserimenti e le rimozioni, possiamo procedere con la creazione del nuovo IG, che avrà al suo interno le nuove attività. Nelle figure 1.9, 1.10, 1.11 e 1.12 è mostrato come il grafo viene modificato.

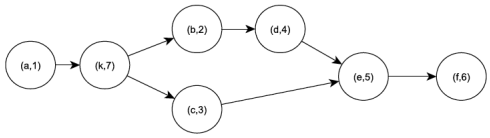

Figura 1.9: Ig dopo il primo inserimento

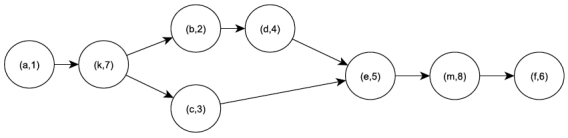

Figura 1.10: Ig dopo il secondo inserimento

1.2 Implementazione di IG Building Algorithm

L'algoritmo è stato implementato in Python ed è costituito da una funzione main() che richiama a sua volta le varie procedure e funzioni che rappresentano i vari passaggi dell'algoritmo descritto nella sezione precedenete. Il main richiede in input due parametri, il path del dataset che vogliamo analizzare e il path della rete di petri che rappresenta il modello delle tracce.

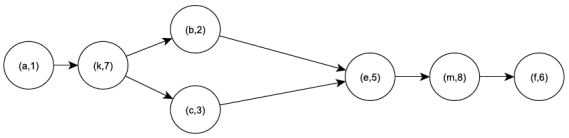

Figura 1.11: IG riparato dopo la prima rimozione

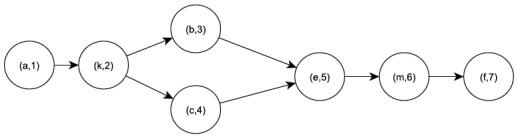

Figura 1.12: IG finale

1.2.1 Analisi del codice di BIG

Come detto precedentemente, tutto lo script è regolato dalla procedura di main chiamata BIG, che prende in input i parametri elencati precedentemente. Nella prima parte del codice, viene caricato il dataset tramite l'utilizzo della libreria PM4PY. Nella riga successiva, viene caricata anche la rete di petri allo stesso modo. Importati i due file, si procede nel core dello script, andando a definire un for-loop all'interno del quale vengono passate le singole tracce contenute nel dataset. Il primo step è quello di estrapolare le AlignedTrace che viene caricato anch'esso grazie all'utilizzo della libreria libreria PM4PY, attraverso l'utilizzo della funzione pick_alignment_trace. Tutti i vari step dell'algoritmo sono stati implementati attraverso funzioni ad hoc, che verranno richiamati man mano all'interno del main. Conclusa la prima fase, si passa alla riparazione dei grafi implementate da funzioni create appositamente per eseguire le operazioni di inserimento ed eliminazione . Una volta eseguite queste due operazioni, si creeranno i nuovi IG e verranno salvati sotto formato .txt.

1.2.2 Criticità

Lo script è stato creato grazie alla collaborazione di diversi studenti, con l'unico scopo di creare uno strumento capace di creare l'output definito dall'algoritmo ma non è stato progettato per affrontare dei test di performance per capire quali fossero i limiti di dimensione che riesce a maneggiare. I limiti sono imposti principalmente dalla libreria PM4PY che viene utilizzata sia per il caricamento della rete di petri che per il dataset. Se per il primo file la dimensione può rimanere nell'ordine di qualche decina di Megabyte, il secondo può avere dimensioni dell'ordine delle centinaia di Megabyte se non dei Gigabyte, poichè il dataset può descrivere l'esecuzioni di processi di archi temporali molto lunghi mentre la rete di petri descrive la struttura di un singolo processo. Anche se la libreria PM4PY risulta essere capace di gestire dataset con molte tracce, rimane il problema delle performance. Il core dell'applicazione si basa su un ciclo for che applica l'algoritmo di BIG una traccia alla volta. Il

meccanismo risulta essere poco performante all'aumentare del numero di tracce da esaminare.
Questi sono i principali problemi che la nostra soluzione vuole andare a risolvere, cercando di migliorare sia la capacità che le performance dell'import del dataset, che l'esecuzione dell'algoritmo stesso.

Capitolo 2

BIG for Big Event Log: BIG^2

Note le criticità della versione originale, in questo capitolo verrà spiegata la soluzione creata. La prima differenza che troviamo con la verione originale è la divisione dello script in due applicativi differenti:

- Il primo per la fase di ingestion, è stato scritto con il linguaggio Java e permette di caricare all'interno di un dataframe il contenuto di un file di qualsiasi dimensione;
- Il secondo è stato scritto in python dove risiede il cuore del nostro applicativo in cui viene applicato l'algoritmo ad ogni traccia presente nel log.

Per entrambi gli script, è stato utilizzato sparkSQL.

2.1 Apache Spark

Apache Spark [2] è un framework open source per il calcolo distribuito sviluppato dall'AMPlab della California university e successivamente donato alla Apache Software Foundation.
Apache Spark è un motore di analisi unificato per l'elaborazione di dati su vasta scala con moduli integrati per SQL, flussi di dati, machine learning ed elaborazione di grafici. Spark può essere eseguito su Apache Hadoop, Apache Mesos, Kubernetes, in modo indipendente, nel cloud e su diverse origini dati. Ricorrente è la domanda: quando si usa Apache Spark e quando invece Apache Hadoop?
Entrambi figurano tra i sistemi distribuiti più importanti oggi sul mercato. Sono entrambi progetti Apache simili di primo livello che spesso vengono utilizzati insieme. Hadoop è usato principalmente per operazioni a uso intensivo di dischi con il paradigma MapReduce.
Hadoop MapReduce è un modello di programmazione per l'elaborazione di grandi dataset con un algoritmo distribuito parallelo. Gli sviluppatori possono scrivere operatori altamente parallelizzati, senza doversi preoccupare della distribuzione del lavoro e della fault tollerance. Tuttavia, una sfida per MapReduce è il processo sequenziale in più fasi necessario per eseguire un lavoro. Ad ogni passaggio, MapReduce legge i dati dal cluster, esegue operazioni e riscrive i risultati in HDFS. Poiché

ogni passaggio richiede una lettura e una scrittura del disco, i lavori MapReduce sono più lenti a causa della latenza dell'I/O del disco. Per risolvere i limiti di Map Reduce è stato creato Spark il quale esegue l'elaborazione in memoria, riducendo il numero di passaggi in un processo e riutilizzando i dati su più operazioni parallele. Con Spark, è necessario un solo passaggio in cui i dati vengono letti in memoria, le operazioni eseguite e i risultati riscritti, con un'esecuzione molto più rapida. Spark riutilizza anche i dati utilizzando una cache in memoria per velocizzare notevolmente gli algoritmi di apprendimento automatico che chiamano ripetutamente una funzione sullo stesso set di dati. Il riutilizzo dei dati viene ottenuto attraverso la creazione di DataFrames, un'astrazione su Resilient Distributed Dataset (RDD), che è una raccolta di oggetti memorizzati nella cache in memoria e riutilizzati in più operazioni Spark. Ciò riduce drasticamente la latenza, rendendo Spark più volte più veloce di MapReduce, soprattutto quando si esegue l'apprendimento automatico e l'analisi interattiva. La comprensione delle diverse funzionalità di ciascuno permetterà di capire quale implementare nelle diverse situazioni. L'ecosistema Spark comprende cinque componenti chiave:

- **Spark Core** è un motore di elaborazione dati distribuito e per uso generico. Comprende librerie per SQL, l'elaborazione dei flussi, il machine learning e il calcolo dei grafici: tali sono tutti elementi che possono essere utilizzati insieme in un'applicazione. Spark Core è la base di un intero progetto, che fornisce l'invio, la programmazione e le funzionalità I/O di base di attività distribuite;

- **Spark SQL** è il modulo Spark per lavorare con dati strutturati che supporta un modo comune per accedere a una varietà di origini dati;

- **Spark Streaming** semplifica la creazione di soluzioni per flussi di dati scalabili e a tolleranza di errore. Utilizza l'API integrata nel linguaggio di Spark per l'elaborazione dei flussi, in modo da poter scrivere job in flussi allo stesso modo dei job in batch. Spark Streaming supporta Java, Scala e Python e dispone di una semantica "exactly-once" di tipo stateful;

- **MLlib** è la libreria Spark scalabile per il machine learning con strumenti che rendono il ML pratico scalabile e facile da usare. MLlib contiene molti algoritmi di apprendimento comuni, come la classificazione, la regressione, i suggerimenti e il clustering. Contiene inoltre il flusso di lavoro e altre utilità, comprese le trasformazioni delle funzionalità, la costruzione di pipeline ML, la valutazione dei modelli, l'algebra lineare distribuita e le statistiche;

- **GraphX** è l'API Spark per i grafici e il calcolo grafico parallelo. È flessibile e funziona perfettamente sia con i grafici che con le raccolte; unifica estrazione, trasformazione, caricamento, analisi esplorativa e calcolo iterativo dei grafici all'interno di un unico sistema. Oltre a essere un'API altamente flessibile, GraphX fornisce diversi algoritmi grafici. In termini di prestazioni, compete

con i sistemi di grafici più rapidi pur mantenendo la flessibilità, la tolleranza di errore e la facilità d'uso di Spark;

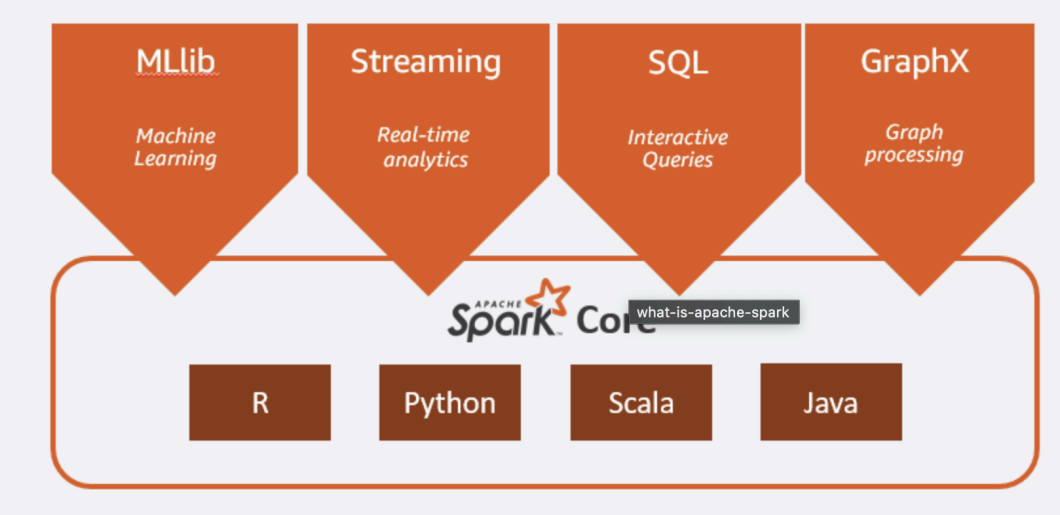

Figura 2.1: Moduli di Spark

Nel nostro caso d'uso, andremo ad utilizzare molto il modulo di SparkSql.

2.1.1 SparkSQL

Spark SQL [3] è un query engine distribuita che fornisce query interactive a bassa latenza fino a 100 volte più veloci di MapReduce. Include un ottimizzatore basato sui costi,la generazione di codice per query rapide, scalando fino a migliaia di nodi. Gli analisti aziendali possono utilizzare SQL standard o Hive Query Language per eseguire query sui dati. Gli sviluppatori possono utilizzare le API, disponibili in Scala, Java, Python e R. Supporta varie origini dati predefinite tra cui JDBC, ODBC, JSON, HDFS, Hive, ORC e Parquet.
In particolare andremo ad utilizzare la libreria SparkSql per i suoi dataframe.
Ma cosa sono i dataframe?
Per capirlo meglio, dobbiamo introdurre il concetto di dataset. Il dataset è una collezione distribuita di dati che ha diverse prorpietà, ad esempio la possibilità di usare le lambda function e l'uso del optimized execution engine. I dataset possono essere costruiti attraverso gli oggetti della JVM e possono essere manipolati con funzioni di filtraggio, mapping ecc.
Il dataframe è un dataset organizzato in colonne e può essere visto come l'equivalente di una tabella relazionale. Abbiamo diversi modi per creare un dataframe, possiamo partire da file strutturati, database esterni o RDD preesistenti. Grazie a queste proprietà, abbiamo potuto implementare il nostro file parser, come vedremo nel paragrafo successivo.

2.2 XMLParser

Come descritto nel capitolo precedente, il codice implementato presenta alcune criticità per il caricamento del file di log. La libreria PM4PY [4] rappresenta un collo di bottiglia per lo scale up dell'applicazione, poichè all'aumentare della dimensione del file diminuiscono le prestazioni. Per questo motivo, si è creato uno script per il caricamento dei file di log attraverso l'utilizzo di Spark SQL.
Il codice prende come parametro il path del file che si vuole leggere, che verrà utilizzato successivamente come argomento della funzione di *read* della spark session. Per il caricamento del dataset, si è utilizzata una libreria esterna, XML Data Source for Apache Spark [5]. Questa libreria è stata sviluppata per il caricamento di sorgenti di tipo XML all'interno dei Dataframe di Spark. La scelta di utilizzare questa libreria è dettata dall'impossibilità di caricare file di tipo XML attraverso le funzioni definite all'interno del dataframe.

```
public static void main(String[] args) throws AnalysisException {

    String pathfile_parquet = "random_noise";
    long start_time = System.currentTimeMillis();

    SparkSession spark = SparkSession
            .builder()
            .master("local[2]")
            .appName("Parser")
            .config("spark.driver.bindAddress", "127.0.0.1")
            .getOrCreate();

    spark.sql("set spark.sql.caseSensitive=true");

    long spark_session_time= System.currentTimeMillis();

    String sql =  "CREATE TEMPORARY TABLE filexes USING
        com.databricks.spark.xml OPTIONS (path
        '/Users/jozie/Desktop/NewBigOriginale/BIG
        Datasets/testBank2000NoRandomNoise/testBank2000NoRandomNoise.xes'
        ,rowTag 'trace', value_tag 'event')";
    spark.sql(sql);
```

Creato il dataframe, verranno eseguite delle query SQL per semplificare il tracciato del dataset, per ridurlo ad una struttura più semplice e tralasciando i campi che non sono di nostro interesse.

```
    String explode_string_col =  "select explode(string) as
        trace_attribute, event from filexes";
    String sql_explode_event = "select trace_event._value as
        trace_id,explode(event) as element_event from trace_event";
    String explode_element_event = "select trace_id,
        explode(element_event.string) as string_event,
```

```
    element_event.date._key as date_key,
    element_event.date._value as  date_value from second_explode";
String explain_string_struct = "select trace_id,
    string_event._key as string_key, string_event._value as
    string_value, date_key, date_value from thrid_explode";
```

Man mano che le query vengono eseguite, andremo a creare dei dataframe, fino a creare la versione finale.

```
Dataset explode_trace_attribute = spark.sql(explode_string_col);

Dataset creazione_colonne =  explode_trace_attribute
.select("trace_attribute._key","trace_attribute._value","event");

Dataset trace_event = creazione_colonne.
select("_value", "event")
.where("_key = 'concept:name'");
trace_event.createTempView("trace_event");

Dataset primo_explode_data = spark.sql(sql_explode_event);
primo_explode_data.createTempView("second_explode");

Dataset second_explode_event = spark.sql(explode_element_event);
second_explode_event.createTempView("thrid_explode");

Dataset data_explosure = spark.sql(explain_string_struct);
data_explosure.createTempView("data_explosure");

Dataset trace_activity = spark.sql("Select trace_id,string_value
    as activity_name from data_explosure where  string_key =
    'concept:name'");
trace_activity.createTempView("trace_activity");
```

Una volta ottenuto il dataframe, dobbiamo salvare il nostro dataset. Si hanno diverse opzioni per memorizzare un dataframe. Ad esempio, se si ha una connessione jdbc ad un server sql, è possibile salvarlo all'interno di un database relazionale, oppure è possibile salvarlo come un file .csv. Entrambe queste soluzioni sono state scartate perchè non risolvono il problema per cui è stato creato questo script, dato che andremmo a cambiare il formato del file del log, senza aggiungere nessun tipo di miglioramento.
Esiste una terza opzione per memorizzare il risultato delle query, ossia quello di salvarlo come file .parquet. Apache Parquet è un formato open source e column-oriented progettato per il data storage, che fornisce un'efficiente compressione dei dati e performance migliori per la gestione di strutture dati complessi.Inoltre ha diversi benefici:

- Possibilità di salvare dati di qualsiasi tipo, come tabelle, immagini, video e documenti;

- Risparmiare spazio di archiviazione usando un'efficiente compressione delle colonne e un flessibile schema di codifica per le colonne di diverso tipo;
- Incremento del throughput e delle performance usando diverse tecniche, come il data skipping, che permette di recuperare valori all'interno delle colonne senza dover leggere l'intera riga.

Inoltre è possibile ripartire il salvataggio del dataframe, indicando la colonna di partizionamento. Nel nostro caso, questa soluzione si rivelerà fondamentale per migliorare le performance. Partizionando il nostro dataframe in base ai valori della colonna trace_id, ci permetterà successivamente di poter applicare una specifica funzione per ciascuna ripartizione. Come verrà mostrato successivamente, questo aspetto ci permetterà di codificare una funzione BIG e di passarla ad ogni singola partizione del nostro dataframe.

```
trace_activity.write()
.partitionBy("trace_id").
format("parquet")
.save(pathfile_parquet+".parquet");
```

2.3 BIG²

In questa sezione andremo a spiegare la soluzione creata. La struttura del nuovo applicativo è molto semplice e si articola su tre livelli:

- Nel primo livello avremo il main, nel quale andremo a creare la variabile che conterrà il path del file parquet creato da XMLParser, e che passerà come argomento al secondo livello;
- Nel secondo livello si ha la funzione BIG, che prenderà in input il path del file parquet e andrà a creare il dataframe, inizializzando una Spark Session. Subito dopo andrà ad eseguire la funzione big_partitioned per ciascuna partizione presente all'interno del dataframe, grazie all'utilizzo del metodo foreach_partition() del Dataframe API;
- Il terzo e ultimo livello è la funzione big_partitioned() che rappresenterà l'algoritmo BIG.

Partendo dalla funzione main, essa viene richiamata passando come argomento il path del file parquet creato da XMLParser; inoltre viene creata una variabile che verrà valorizzata con il path del file della rete di petri.

```
def main(fileparquet):
  netfile = "/Users/jozie/Desktop/NewBigOriginale/BIG Datasets
  /testBank2000NoRandomNoise/testBank2000NoRandomNoise_petriNet.pnml"
  BIG(netfile,fileparquet)
```

```
if __name__ == '__main__':
  fileparquet = sys.argv.__getitem__(1)
  main(fileparquet)
```

Successivamente viene richiamata la funzione BIG, che come nello script iniziale, rappresenta il core di tutta la nostra procedura.

```
def main(fileparquet):
  netfile = "/Users/jozie/Desktop/NewBigOriginale/BIG Datasets
  /testBank2000NoRandomNoise/testBank2000NoRandomNoise_petriNet.pnml"
  BIG(netfile,fileparquet)


if __name__ == '__main__':
  fileparquet = sys.argv.__getitem__(1)
  main(fileparquet)
```

Il primo cambiamento è che non verrà caricato più l'interno file di log, ma verrà creata una Spark Session che sarà successivamente utilizzata per la creazione del dataframe, partendo dal file parquet passato come argomento. Il secondo è che la funzione di BIG, che precedentemente era il main, è stata incapsulata all'interno di una funzione chiamata big_partitioned(). Questa funzione verrà passata come argomento al metodo foreach_partition() disponibile all'interno dei dataframe API. Questo metodo viene utilizzato principalmente quando si ha bisogno di applicare una funzione per ciascuna partizione del dataframe.

```
def BIG(net_path, file_parquet):

  start_time_total = time.time()

  spark =
      SparkSession.builder.appName("BigSpark").config("spark.driver.bindAddress",
      "127.0.0.1").master("local[4]").getOrCreate()

  intermediate_temp = time.time()

  net, initial_marking, final_marking = pnml_importer.apply(net_path)
  trace_event_activity = spark.read.parquet(file_parquet)


  gviz = pn_visualizer.apply(net, initial_marking, final_marking)
  gviz.render(filename="petri")

  trace_event_activity.foreachPartition(big_partitioned)
  end_time=time.time()
  esecution_time= end_time-start_time_total
  avvioSparkSession= intermediate_temp-start_time_total
```

```
  print("il tempo di esecuzione totale    "+str(esecution_time))
  print("Il tempo di avvio della SparkSession
      "+str(avvioSparkSession))
```

Questa funzionalità ci permetterà di aumentare le prestazioni del nostro applicativo. Avendo partizionato precedentemente il nostro dataset tramite l'utilizzo del XML-PARSER, la funzione di BIG verrà eseguita parallelamente per ciascuna traccia e non più sequenzialmente, come nello script originario.

```
def big_partitioned(f):

  print(f)

  view = False

  netfile = "/Users/jozie/Desktop/NewBigOriginale/BIG
      Datasets/testBank2000NoRandomNoise/testBank2000NoRandomNoise_petriNet.pnml"
  log_file = "/Users/jozie/Desktop/NewBigOriginale/BIG
      Datasets/testBank2000NoRandomNoise/testBank2000NoRandomNoise.xes"

  splits = logfile.split('/')
  name = splits[-1].split(".")[0]
  sort_labels = False

  net, initial_marking, final_marking = pnml_importer.apply(netfile)

  cr = findCausalRelationships(net, initial_marking, final_marking)
  if view:
    print(cr)

  file_xes_by_trace_id = pd.DataFrame(f, columns=['concept:name',
      'trace_id'])
  parameters =
      {log_converter.Variants.TO_EVENT_LOG.value.Parameters.CASE_ID_KEY:
      'trace_id'}
  event_log = log_converter.apply(file_xes_by_trace_id,
      parameters=parameters, variant=log_converter.Variants.TO_EVENT_LOG)

  for trace in event_log:
      Aligned, A = pick_aligned_trace(trace, net, initial_marking,
          final_marking)
      Align = Aligned[0]
      A1 = A[0]
      print('Aligned to model')
      print(Align)
      # print(L1)
      print('with invisible moves')
      print(A1)
      map, ins = mapping(Align, A1)

      # compliant mi serve per generare l'ig in base al modello (rimuove
```

```
    dalla traccia allineata i move)

compliant = compliant_trace(Align)
effettiva = compliant_trace(A1)

print(compliant)
print('Effettiva: ', effettiva)

print("map: ", map)
print("ins: ", ins)

d = []

trace_start_time = time.time()
num = trace.attributes.get('concept:name')
# estrazione dell' IG su cui poi devo fare riparazione
V, W = ExtractInstanceGraph(compliant, cr)
print('V')
print(V)
print('W')
print(W)
if view:
  print("\n\n-----------------------------------\nUnrepaired
      Instance Graph")

V_n = node_number(V)
# print('V_n = ', V_n )
W_n = edge_number(W)
# print('W_n = ', W_n)

for element in map:  # crea le liste Eremp e Erems
  if element[1] == 0:  # da cancellare
    d.append(element)

graph = viewInstanceGraph(V, W)

Vpos = []
for node in V:
  Vpos.append(node)

print('Vpos1: ', Vpos)

for el in map:
  if el[1] == 0:
    Vpos.remove((el[2], el[0]))

print('Vpos = ', Vpos)

print('INSERTION')

for insertion in ins:
```

```
        V, W = ins_repair(V, W, map, insertion, V_n, ins, Vpos)

    print('W repaired: ', W)

    graph = viewInstanceGraph(V, W)

    print('DELETION')
    print(d)

    for deletion in d:
      V, W = del_repair(V, W, map, deletion)

    if len(ins) > len(d) + 3:
      print('TANTEEEEEEE')

    # V,W = repair_insertion(V,W,map,ins)
    print('W aggiornata: ', W)
    print("V aggiornata: ", V)

    minimo = min(V_n)

    W1 = []

    W1 = riordina(W, minimo, W1)

    # print('W sorted: ', W1)

    W1, V1 = aggiorna_label(W, map, V)

    print('W1 label: ', W1)

    # W_sorted = sorting(W1,V)

    print('W ordinata: ', W)

    graph = viewInstanceGraph(V1, W1)

    V1.sort()
    W1.sort()

    nome_file=str(trace[0]['trace_id'])

    saveGFile(V1, W1,
        "/Users/jozie/Desktop/BigSpark/result/"+nome_file, time.time()
        - trace_start_time, sort_labels)
    saveGfinal(V1, W1, "{0}_instance_graphs.g".format(name),
        sort_labels)
```

In una prima fase, andiamo a ricreare la rete di petri e il file di log attraverso l'uso della libreria PM4PY con l'utilizzo del metodo log_converter.apply(). Questa funzione ci permette di ricreare un file di log, partendo da un dataframe, che nel

nostro caso, è una singola traccia riuscendo quindi ad avere un caricamento veloce. Avendo il file di log e la petrinet creati, possiamo iniziare con tutte le operazioni definite all'interno dell'algoritmo BIG.

Capitolo 3

Testing

Creato l'applicativo, si andrà a testarlo per capire se l'obiettivo posto all'inizio del documento sia stato raggiunto. Il test del software è il processo di valutazione e verifica del corretto funzionamento di un'applicazione o di un prodotto software rispetto alle aspettative. I vantaggi del test includono la prevenzione dei bug, la riduzione dei costi di sviluppo e il miglioramento delle prestazioni.
Perchè il test del software è importante?
Pochi possono contestare la necessità del controllo della qualità nello sviluppo del software. I ritardi nella consegna o i difetti del software possono danneggiare la reputazione di un marchio, creando frustrazione nei clienti e determinando il loro abbandono. In casi estremi, un bug o un difetto possono degradare sistemi interconnessi o causare gravi malfunzionamenti.
Esistono molti tipi diversi di test del software, ciascuno con obiettivi e strategie specifici:

- **Test di accettazione:** verifica se l'intero sistema funziona come previsto;
- **Test di integrazione:** garantisce che i componenti o le funzioni del software funzionino insieme;
- **Test dell'unità:** conferma che ogni unità software funzioni come previsto. Un'unità è la componente testabile più piccola di un'applicazione;
- **Test funzionale:** verifica le funzioni emulando gli scenari dell'azienda, in base ai requisiti funzionali. Il test black-box è un modo comune per verificare le funzioni;
- **Test delle prestazioni:** verifica le prestazioni del software quando è sottoposto a diversi carichi di lavoro. Il test di carico, ad esempio, viene utilizzato per valutare le prestazioni in condizioni di carico reali;
- **Test di regressione:** verifica se le nuove funzioni interrompono o peggiorano la funzionalità. Il test di integrità può essere utilizzato per verificare menu, funzioni e comandi a livello superficiale, quando non c'è tempo per un test di regressione completo ;

- **Test di sforzo:** verifica lo sforzo che può sostenere il sistema prima di riportare un errore. Viene considerato un tipo di test non funzionale;
- **Test di fruibilità:** valuta la facilità di utilizzo, da parte di un cliente, di un sistema o un'applicazione web per completare un'attività.

Per il nostro applicativo, andremo a svolgere prevalentemente test di accettazione e di performance.

3.1 Test di accettazione

I test di accettazione vengono eseguiti immediatamente prima del rilascio del prodotto, quindi dopo aver testato l'intero sistema e dopo il conseguente "bug fixing", ovvero la correzione dei difetti riscontrati.
Sono tipicamente di approccio black box testing, quindi orientati alla verifica dal punto di vista strettamente funzionale dell'intero sistema e vengono chiamati anche test di qualità, test finali o accettazione di rilascio. Proprio dal nome di tali test si evince che abbiano più lo scopo di dare fiducia sull'adeguato funzionamento del sistema, piuttosto che quello di trovare difetti.
Il test di accettazione proposto per il nostro applicativo riguarda la bontà dell'output prodotto, cioè, produrre lo stesso risultato della versione originale.

3.1.1 Struttura del test

Il nostro test ha due obiettivi principali:

- Il numero dei file prodotti sia lo stesso del numero di tracce contenute all'interno del file di log;
- Il contenuto dei file prodotti dalle due versioni deve essere uguale per singola traccia.

Per eseguire questi test, abbiamo considerato tre dataset differenti:

- testBank2000SCCUpdatedCopia 00.07.27.xes
- testBank2000NoRandomNoise.xes
- bpi2012decompositionExpr.xes

Per eseguire il test, si è lanciata prima la versione originale e successivamente la soluzione proposta dalla tesi.
Nella tabella verranno riassunti il numero di file prodotti da ciascuna esecuzione è avràla seguente struttura: Nella prima colonna verrà mostrato il dataset; nella seconda il numero di tracce contenute all'interno del log; nella terza il numero di file prodotti nella versione originale e nell'ultima colonna il numero di file prodotti nella

versione proposta.

Dataset	Tracce file	versione originale	Big[2]
testBank2000SCCUpdatedCopia	1.995	1.995	1.995
bpi2012decompositionExpr	7.455	7.455	7.455
testBank2000NoRandomNoise	1500	1500	1500

Per il secondo obiettivo del test , abbiamo considerato per ciascun dataset, tre tracce casuali contenute al suo interno, e abbiamo verificato che i file prodotti, che rappresetano gli IG riparati, siano uguali. Nella prossima tabella è vengono mostrati gli id delle tracce considerate per ciascun dataset, e se i file prodotti dalle due versioni coincidono.

Dataset	trace_id	Uguali?
bpi2012decompositionExpr.xes	173688	Si
bpi2012decompositionExpr.xes	196924	Si
bpi2012decompositionExpr.xes	196921	Si
testBank2000NoRandomNoise.xes	trace_0	Si
testBank2000NoRandomNoise.xes	trace_1999	Si
testBank2000NoRandomNoise.xes	trace_609	Si
testBank2000SCCUpdatedCopia 00.07.27.xes	trace_0	Si
testBank2000SCCUpdatedCopia 00.07.27.xes	trace_1999	Si
testBank2000SCCUpdatedCopia 00.07.27.xes	trace_609	Si

In appendice verranno inseriti sia le tracce che l'output presi in esame per il test.

3.2 Test di performance

Il Performance Testing determina la corretta esecuzione di un sistema informatico misurandone i tempi di risposta. Il suo obiettivo è quello di fornire metriche sulla velocità dell'applicazione. Il test delle prestazioni risponde così ad un'esigenza di velocità degli utenti e delle aziende.
I test sono stati eseguiti in locale, utilizzando il Macbook Air 13 2021 con chip M1 che presenta le seguenti caratteristiche tecniche:

- Chip Apple M1: CPU 8-core con 4 performance core e 4 efficiency core
- 8 Gb di RAM
- macOS Monterey 12.4

3.2.1 Struttura del test

Il test si pone come obiettivo quello di misurare come la soluzione proposta lavori all'aumentare della dimensione del dataset passatogli in input e paragonare le sue

performance con la versione originale, mostrando eventualmente delle differenze tra i tempi di esecuzione. Per eseguire i test sono stati presi gli stessi dataset del test precedente, ma per ottenere un andamento dei tempi di esecuzione all'aumentare della dimensione, abbiamo applicato un fattore moltiplicativo al numero delle tracce (x1,5 ,x2, x3, x5, x10, x20).
Per il calcolo del tempo di elaborazione, abbiamo utilizzato due funzioni: Time.time() per gli script python e currentTimemills() per Java.
Il test avviene eseguendo i vari script singolarmente,per 5 esecuzioni, riavviando la macchina ad ogni esecuzione così da non avere processiin background che influenzano le prestazioni della macchina.
I risultati verranno esposti nella tabella sottostante e presenterà le seguenti colonne:

- **Dataset**: Nome del dataset;
- **N°tracce**: Numero di tracce presenti nel dataset;
- **N°eventi**: Numero di eventi presenti nel dataset;
- **XML Parser**: Media dei tempi di esecuzione espressa in secondi dello script Java per il caricamento del dataset;
- **BIG**^{2}: Media dei tempi di esecuzione espressa in secondi della versione modificata di BIG;
- **BIG**: Media dei tempi di esecuzione espressa in secondi della versione originale di BIG;
- **Delta**: Rappresenta quanto la somma dei tempi di esecuzione della soluzione proposta si discosti dal tempo della versione originaria. la formula applicata è la seguente:[(BIG^2+XMLParser)-BIG]/BIG

Dataset	N°tracce	N Eventi	XMLParser	BIG^2	BIG	Delta
banking	1995	38963	18,802	36	38,339	43,28%
decomposition	7455	85426	45,834	133	151,558	17,74%
random_noise	1500	30228	15,743	42	35,399	63,82%
banking	3990	99750	34,736	100	182,736	-26,30%
decomposition	14910	208740	73,895	255	521,475	-37,02%
random_noise	3000	78000	25,345	106	220,979	-40,40%
banking	5985	149625	44,160	100	273,153	-47,25%
decomposition	22365	313110	108,117	377	768,243	-36,82%
random_noise	4500	117000	34,981	154	330,651	-42,98%
banking	9975	249375	49,074	224	454,440	-39,88%
decomposition	37275	521850	182,664	649	1410,300	-41,06%
random_noise	7500	195000	43,877	256	552	-45,63%
banking	19950	498750	89,576	331	797,547	-47,30%
decomposition	74550	1043700	342,394	1618	5180,45	-62,17%
random_noise	15000	390000	78,893	488	1262,030	-55,06%
banking	39900	997500	187,065	1401	5854,11	-72,87%
decomposition	149100	2087400	622355,5	7238	18256,741	-56,94%
random_noise	30000	780000	147,540	1440	3320,868	-52,19%
banking	2992	74812	31,109	71	136,336	-25,28%
decomposition	11182	156555	74,480	214	120,750	138,50%
random_noise	2250	58500	26,569	82,395	167,106	-34,79%

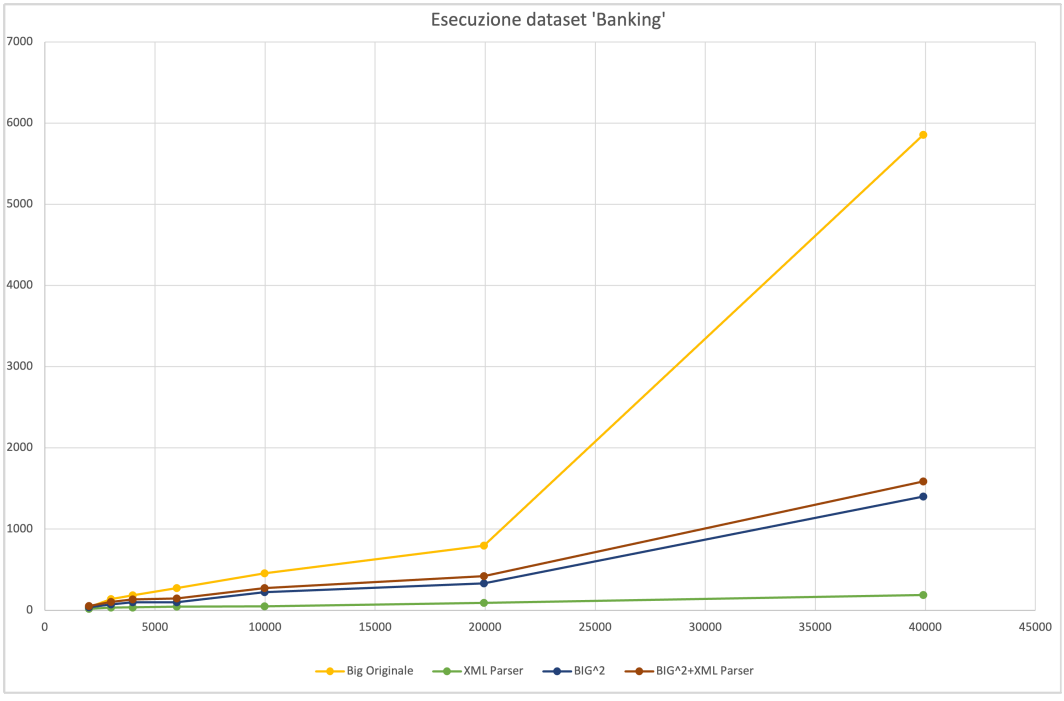

Figura 3.1: Andamento tempi di esecuzione del dataset Banking

Nella figura 3.1 viene rappresentato l'andamento dei tempi di esecuzione dei vari script per il dataset banking al variare del numero di tracce.La curva di crescita dei tempi di esecuzione risulta avere una bassa pendenza per un numero di tracce minore di 20000, per poi aumentare.

In figura 3.2, si cerca di trovare la funzione che approssimi meglio l'andamento

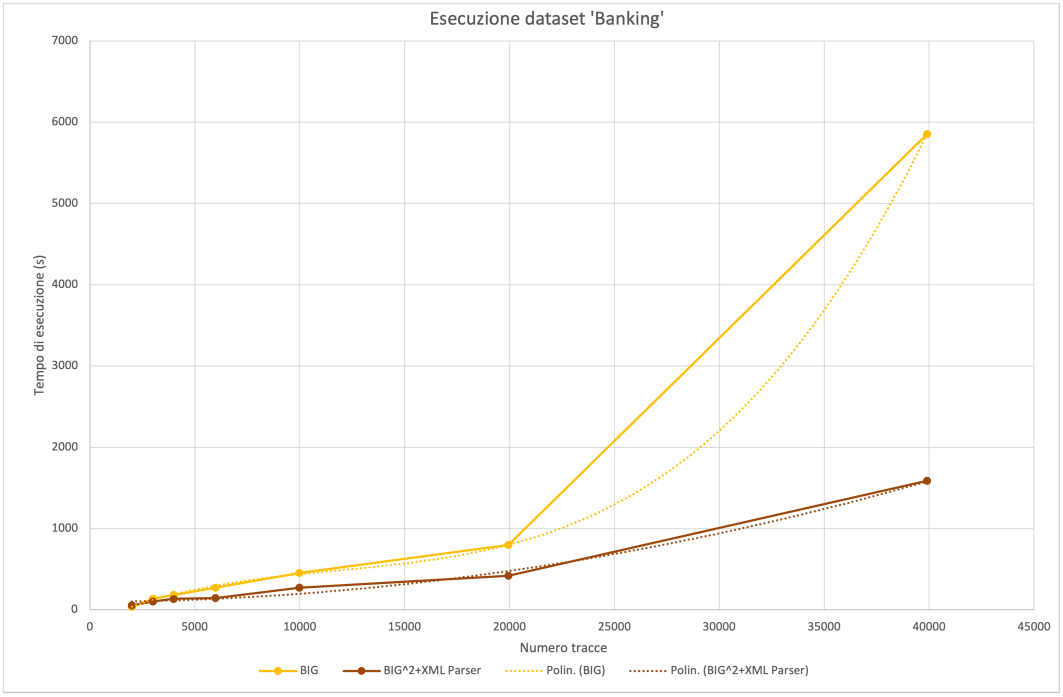

Figura 3.2: Linee di tendenza dei tempi di esecuzione delle due versioni di BIG per il dataset Banking

dei tempi di esecuzione. Nel caso del dataset banking, dopo diversi tentativi, si è notato come l'andamento venga approssimato bene con una funzione polinomiale di terzo grado fino al raggiungimento delle 20000 tracce. All'aumentare delle tracce, si discosta assumendo una pendenza maggiore della linea di tendenza, aumentando maggiormente il tempo di esecuzione al salire del numero di tracce. L'andamento della soluzione proposta si approssima molto bene con una polinomiale di secondo grado e non presenta pendenze eccessive come nel caso precedente.

In figura 3.3 viene mostrato il guadagno che si ottiene dalla nuova versione di BIG rispetto all'originale. Notiamo come al salire del numero di tracce, il guadagno aumenta andando a raggiungere il 72%. Questa percentuale si può osservare anche nella figura 3.2. In questo grafico, il guadagno è il segmento verticale che unisce i punti della retta rossa, che rappresenta l'esecuzione della nostra soluzione e la retta gialla, che rappresenta l'esecuzione della versione originale di BIG.

Il secondo dataset preso in esame "random_noise" mostra dei risultati che indicano come la versione originale mantenga la sua tendenza polinomiale anche all'aumentare

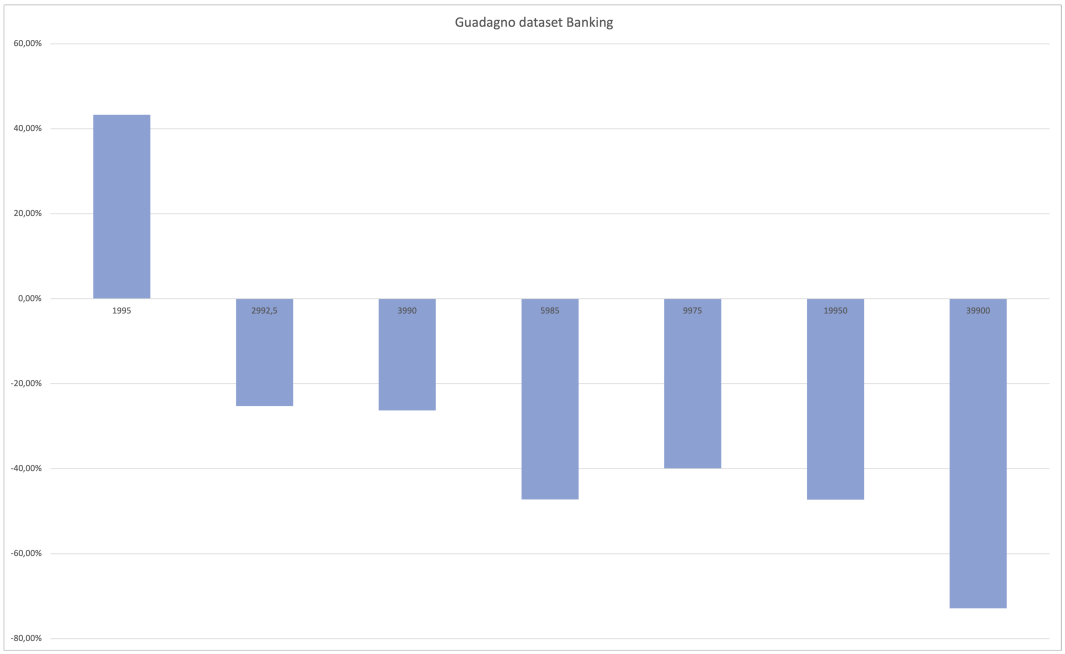

Figura 3.3: Delta dataset Banking

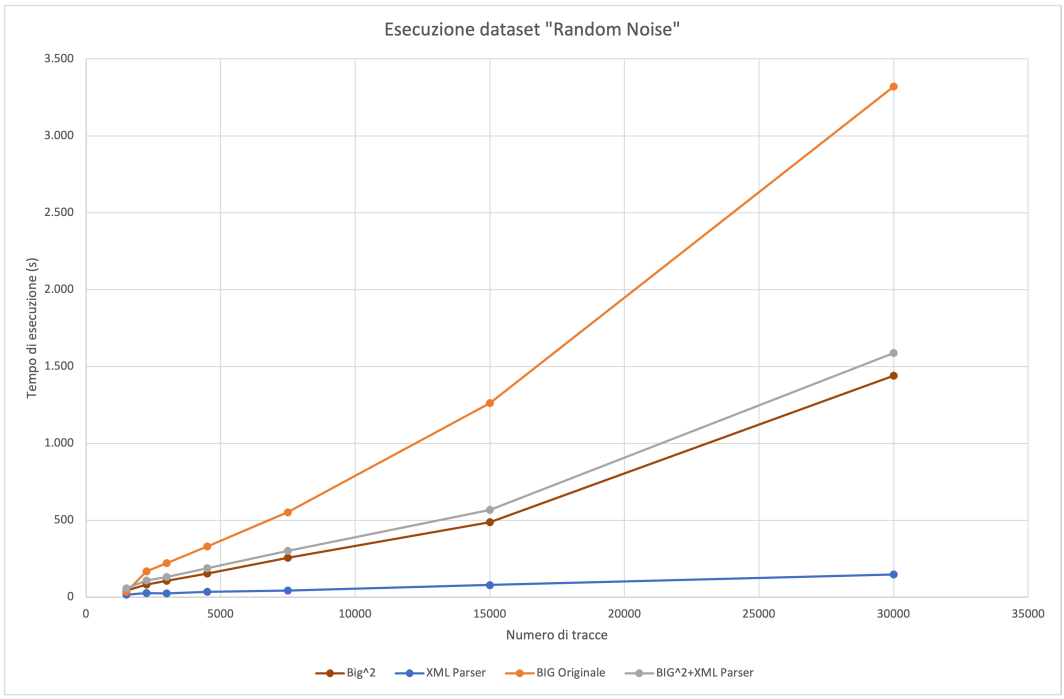

Figura 3.4: Andamento delle esecuzioni del dataset Random_noise

del numero di tracce, che viene rappresentata in figura 3.5. Nella 3.4 vengono mostrati gli andamenti dei tempi di esecuzione dei diversi script utilizzati, evidenziando in da subito che la versione originale abbia delle tempistiche di esecuzione maggiori rispetto a BIG$\hat{2}$.

Il guadagno per questo dataset si ottiene già da numeri bassi di tracce, come viene

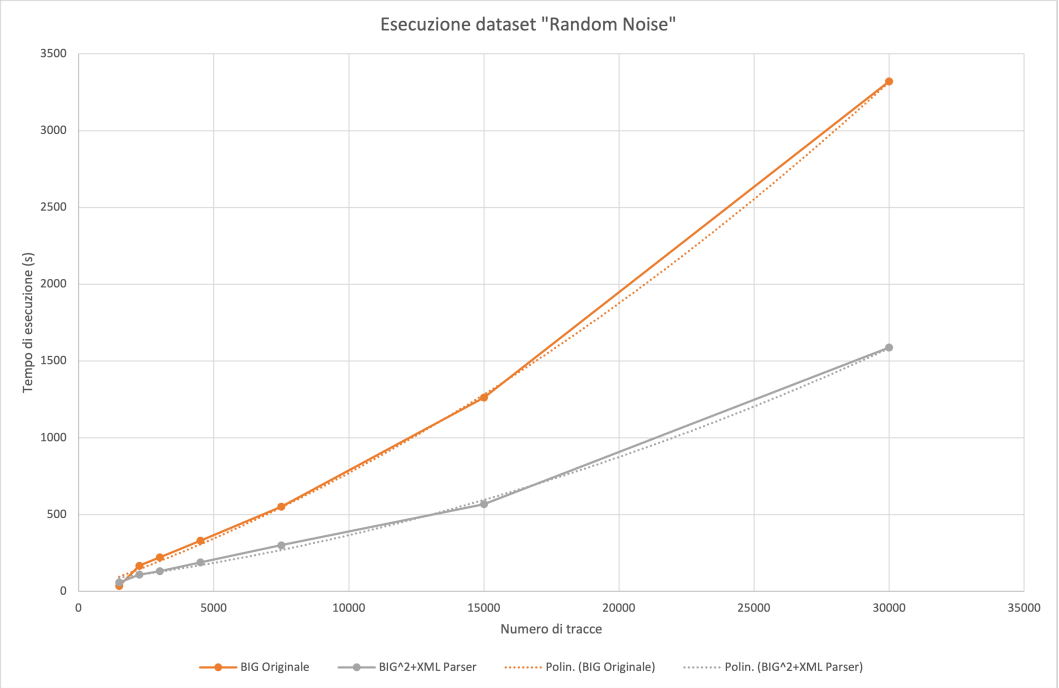

Figura 3.5: Linea di tendenza per l'andamento delle due versioni di BIG per il dataset random_noise

mostrato nella figura 3.6.

Per il terzo ed ultimo dataset, l'andamento dei tempi di esecuzione (figura 3.7) è simile ai precedenti. BIG presenta un aumento di pendenza maggiore rispetto a BIG$\hat{2}$ a 37000 tracce per poi aumentare notevolmente anche a 74500 tracce.

BIG$\hat{2}$ invece aumenta la propria pendenza ma si mantiene al di sotto della versione originaria mantenendo un notevole distacco. In figura 3.8 è rappresentata tale differenza all'aumentare del numero di tracce.

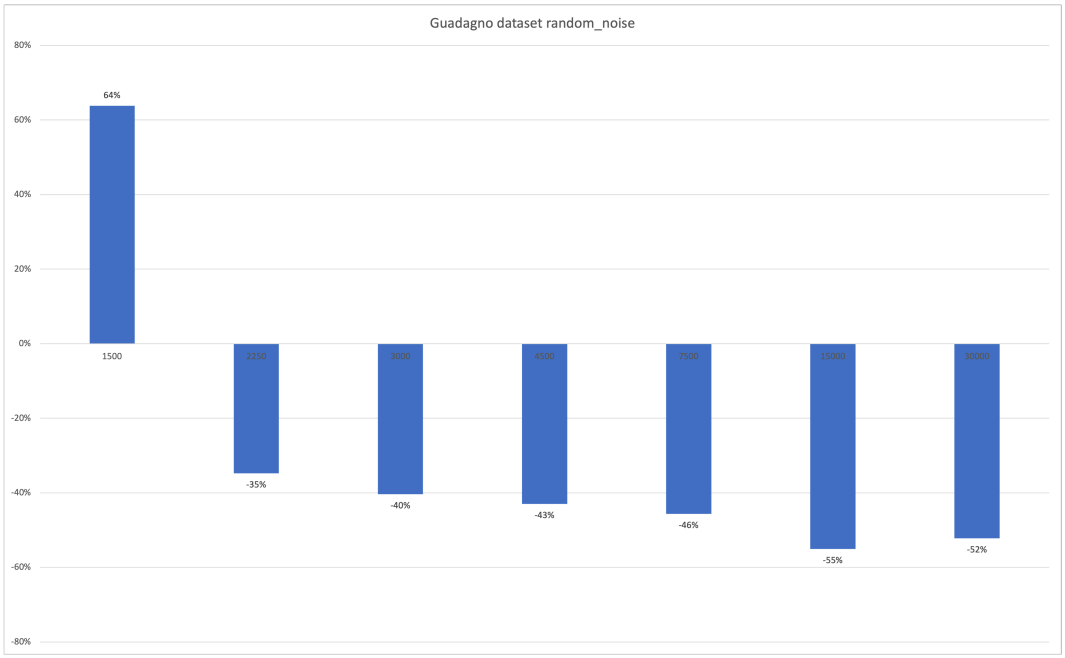

Figura 3.6: Delta dataset random_noise

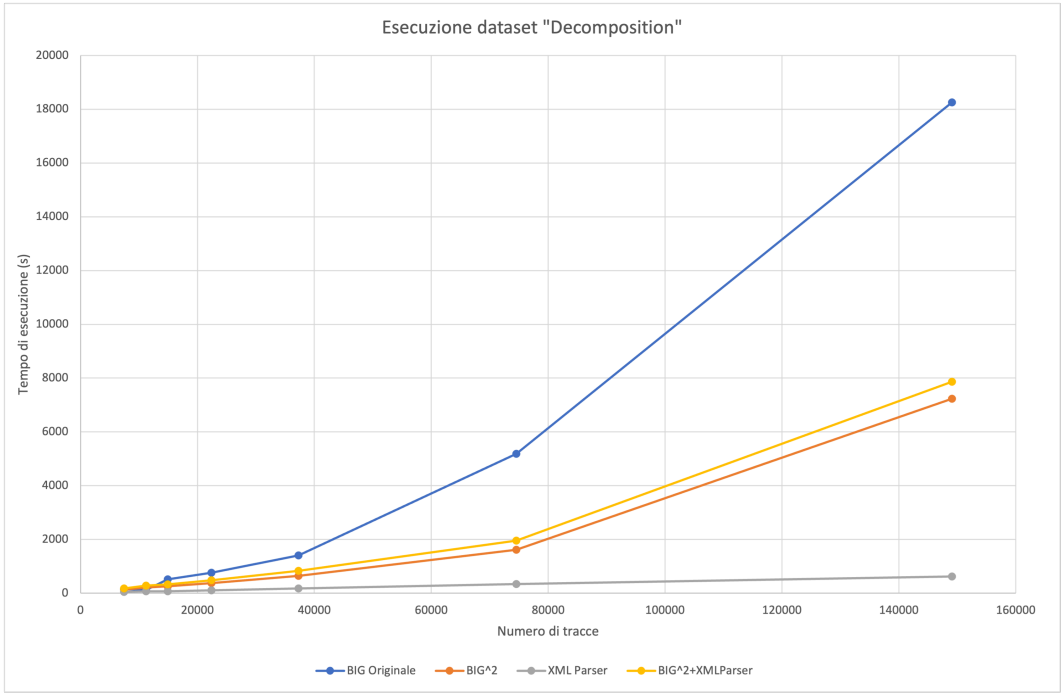

Figura 3.7: Andamento tempi di esecuzione del dataset deomposition

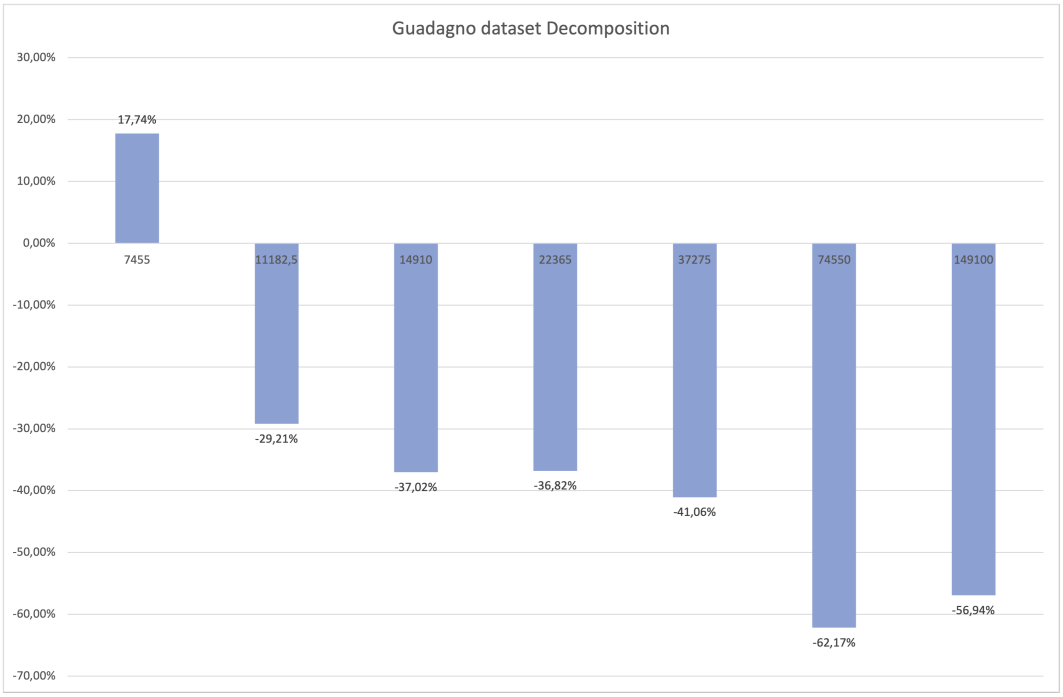

Figura 3.8: Delta dataset decomposition

Capitolo 4

Conclusioni

Dai risultati ottenuti, possiamo affermare di aver raggiunto gli obiettivi dichiarati in questo documento.Le varie modifiche dell'applicativo dichiarate nei capitoli precendenti hanno portato ai risultati prefissati.
In particolar modo la scelta di non caricare l'intero dataset tramite la libreria PM4PY ma di spezzare il caricamento e l'elaborazione di esso ha mostrato che le performance non siano peggiorate bensì abbiano portato l'applicativo ad un miglioramento dei tempi di esecuzione rispetto alla versione originale.
Inoltre il partizionamento del dataframe in fase di ingestion e l'esecuzione dell'algoritmo per ciascuna partizione permette di effettuare il performance tuning sulla spark session affinchè si possano migliorare ancor più i tempi di esecuzione in macchine specifiche per questo tipo di calcolo.

Codice BIG

```
def findCausalRelationships(net, im, fm):
  fp_net = footprints_discovery.apply(net, im, fm)
  return list(fp_net.get('sequence'))

def pick_aligned_trace(trace, net, initial_marking, final_marking):
  aligned_traces = alignments.apply_trace(trace, net, initial_marking,
      final_marking)
  temp = []
  id = 0
  al = []
  temp1 = []
  id1 = 0
  fin = []

  for edge in aligned_traces['alignment']:
    id += 1
    temp.append((id, edge[1]))
  al.append(temp)
  for edge in aligned_traces['alignment']:
    id1 += 1
    temp1.append((id1, edge[0]))
  fin.append(temp1)

  return al, fin
def traccia_effettiva(tracce):
  t = []
  L = []
  i = 0
  for event in tracce:
    i += 1
    nome = event.get('concept:name')
    t.append((i, nome))
  L.append(t)
  return L


#
def checkTraceConformance(trace, net, initial_marking, final_marking):
  aligned_traces = alignments.apply_trace(trace, net, initial_marking,
      final_marking)
  D = []
  I = []
```

```
  id = 0
  temp_d = []
  temp_i = []
  prev_d = False
  curr_d = False
  prev_i = False
  curr_i = False
  del_count = 1
  for edge in aligned_traces['alignment']:
    id += 1
    if edge[1] is None:
      id -= 1
      continue
    if edge[0] == '>>':
      temp_d.append((id, edge[1]))
      curr_d = True
      id -= 1
    if edge[1] == '>>':
      temp_i.append((id, edge[0]))
      curr_i = True

    if (prev_i and not curr_i):
      if len(temp_i) > 0:
        I.append(temp_i)
      temp_i = []
    prev_i = curr_i
    curr_i = False
    if (prev_d and not curr_d):
      if len(temp_d) > 0:
        D.append(temp_d)
      temp_d = []

    prev_d = curr_d
    curr_d = False
  if len(temp_i) > 0:
    I.append(temp_i)
  if len(temp_d) > 0:
    D.append(temp_d)
  return D, I

#TODO MAPPING
def mapping(L1, L2):  # crea il mapping, ritorna il map e la lista ins
    (lista degli inserimenti)

  map = [0] * len(L1)
  id1 = 0
  id2 = 0
  ins = []
  for i in range(len(L1)):
    e1 = L1[i]
    e2 = L2[i]
```

```
    if e1[1] == e2[1]:
      id1 += 1
      id2 += 1
      map[i] = (e1[1], id1, id2)
    elif e1[1] == '>>':  # insertion
      id1 += 1
      map[i] = (e2[1], id1, 0)
    elif e2[1] == '>>':  # deletion
      id2 += 1
      map[i] = (e1[1], 0, id2)

  for j in range(len(L1)):
    e1 = L1[j]
    # e2 = L2[j]
    e3 = map[j]
    if e1[1] == '>>':
      id2 += 1
      map[j] = (e3[0], e3[1], id2)
      ins.append((e3[0], e3[1], id2))

  return map, ins

def ExtractInstanceGraph(trace, cr):
  V = []
  W = []
  id = 1
  for event in trace:
    V.append(event)
    id += 1
  for i in range(len(V)):
    for k in range(i + 1, len(V)):
      e1 = V[i]
      e2 = V[k]
      if (e1[1], e2[1]) in cr:
        flag_e1 = True
        for s in range(i + 1, k):
          e3 = V[s]
          if (e1[1], e3[1]) in cr:
            flag_e1 = False
            break
        flag_e2 = True
        for s in range(i + 1, k):
          e3 = V[s]
          if (e3[1], e2[1]) in cr:
            flag_e2 = False
            break
        if flag_e1 or flag_e2:
          W.append((e1, e2))
  return V, W

#serve per printare il grafico
```

```
def viewInstanceGraph(V, W, view=True, title="Instance Graph"):
  # Conversion to string indexes
  V2 = []
  W2 = []
  for node in V:
    V2.append((str(node[0]), "{0} = {1}".format(node[0], node[1])))
  for edge in W:
    W2.append(((str(edge[0][0]), "{0} = {1}".format(edge[0][0],
        edge[0][1])),
               (str(edge[1][0]), "{0} = {1}".format(edge[1][0],
                   edge[1][1]))))

  print('V2: ', V2)
  print('W2: ', W2)

  dot = Digraph(comment=title, node_attr={'shape': 'circle'})
  for e in V2:
    dot.node(e[0], e[1])
  for w in W2:
    dot.edge(w[0][0], w[1][0])
  if view:
    display.display(dot)
  return dot

def compliant_trace(trace):
  t = []
  id = 0
  for event in trace:
    if event[1] == '>>':
      continue
    else:
      id += 1
      t.append((id, event[1]))

  return t

# funzione per la deletion repair

def del_repair(V, W, map, deletion):
  Eremp = []
  Erems = []
  Pred = []
  Succ = []
  W1 = []
  V1 = []
  d = []
  W2 = []

  to_del = (deletion[2], deletion[0])
  print('Da cancellare = ', to_del)
  for i in range(len(W)):
```

```
    e1 = W[i]
    e2 = e1[1]
    e3 = e1[0]
    if e2 == to_del:
      Eremp.append((e3, to_del))
    if e3 == to_del:
      Erems.append((to_del, e2))

  for a in Eremp:  # crea liste Pred e Succ
    Pred.append(a[0])
  for b in Erems:
    Succ.append(b[1])

  for ep in Eremp:
    W.remove(ep)
  for es in Erems:
    W.remove(es)

  V.remove(to_del)

  for p in Pred:
    for s in Succ:
      W.append((p, s))

  return V, W


# crea una lista dove gli archi sono riportati solo come coppia di numeri

def edge_number(W):
  W_number = []

  for i in range(len(W)):
    arc = W[i]
    a0 = arc[0]
    a1 = arc[1]
    W_number.append((a0[0], a1[0]))

  return W_number


# funzione che, data in input una lista di coppie (numero, label)
    corrispendenti ad ogni nodo del grafo,
# restituisce una lista con solo i numeri di ogni nodo

def node_number(V):
  V_number = []
  for i in range(len(V)):
    nod = V[i]
    V_number.append(nod[0])
```

```
  return V_number
def ins_list_num(ins_list):
  list = []
  for i in range(len(ins_list)):
    p1 = ins_list[i]
    list.append(p1[1])

  return list
def consecutive_insertion(lista, p, pos_t, insertion):
  p -= 1
  q = p - 1
  print('p: ', p, 'q: ', q)
  if p in lista and q not in lista:
    pos_t.remove(insertion[1])
    pos_t.append(p)
  elif p in lista and q in lista:
    print('Qua ci sono arrivato')
    post_t = consecutive_insertion(lista, p, pos_t, insertion)

  return pos_t, p

# funzione di f

def ins_repair(V, W, map, insertion, V_n, ins_list, Vpos):
  Eremp = []
  Pred = []
  Succ = []
  pos_t = []
  W_num = []
  V1 = []
  ins_num = []

  V.insert(insertion[1] - 1, (insertion[2], insertion[0]))
  Vpos.insert(insertion[1] - 1, (insertion[2], insertion[0]))
  pos_t.append(insertion[1])

  W_num = edge_number(W)
  V_num = node_number(V)
  ins_num = ins_list_num(ins_list)

  print('ins num: ', ins_num)

  print('Vpos agg: ', Vpos)


  for p in pos_t:  # numero dell'insertion
    print('P=', p)
    print('Len Vpos: ', len(Vpos))
    # print('posizione V[p]: ', V[p])
    if p < len(Vpos):
      position = Vpos[p]
```

```
      print('Position = ', position)
      pos = position[0]
    else:  # inserimento a ultimo posto
      position = V[-1]
      pos = position[0]
      print('ULtimo elemento ', Vpos[-2])
      print(pos)
    # linee 6-12 pseudocodice
    print('pos: ', pos)
    p_pred = Vpos[p - 2]
    pos_pred = p_pred[0]
    print('P pred: ', p_pred)
    if is_path(pos_pred, pos, W_num, V):  # se c'   un cammino tra p-1 e
        p
      print(is_path(pos_pred, pos, W_num, V))
      for i in range(len(W)):
        arc = W[i]
        a0 = arc[0]
        a1 = arc[1]
        if pos == a1[0] and (
        a0, a1) not in Eremp:  # se esiste un arco nel grafo che entra
            in posizione p e non    considerato in Eremp
          Eremp.append((a0, a1))
          Pred.append(a0)
      for n in Pred:
        for k in range(len(W)):
          e = W[k]
          e0 = e[0]
          e1 = e[1]
          if e0 == n and (e0, e1) not in Eremp:
            Eremp.append((e0, e1))
    else:  # linee 14-15
      print('Condizione verificata')
      for m in range(len(W)):  # pred.append(pos_t-1)
        edge = W[m]
        edge0 = edge[0]
        edge1 = edge[1]
        if (pos_pred) == edge0[0] and (edge0, edge1) not in Eremp:
          print('ELSE')
          Eremp.append((edge0, edge1))
          Pred.append(edge0)
        elif (pos_pred) == edge1[0] and (pos_pred) == V_n[-1]:  #
            insertion all'ultimo nodo del grafo
          print('ECCO IL CASO')
          Pred.append(edge1)
        elif isolato(pos_pred, W_num):
          print('NODO ISOLATO')

for erem in range(len(Eremp)):
  suc = Eremp[erem]
  suc1 = suc[1]
```

```
    if suc1 not in Succ:
      Succ.append(suc1)

  print('Pred = ', Pred)
  print('Succ = ', Succ)
  print('Eremp = ', Eremp)

  for el in Eremp:
    if el in W:
      W.remove(el)

  for i in Pred:
    if (i, (insertion[2], insertion[0])) not in W:
      W.append((i, (insertion[2], insertion[0])))

  for s in Succ:
    if ((insertion[2], insertion[0]), s) not in W:
      W.append(((insertion[2], insertion[0]), s))

  W_num = edge_number(W)
  V_num = node_number(V)

  print('V: ', V)
  print('VPOS Finale: ', Vpos)

  print('+++++++++++++++')

  return V, W

# funzione booleana per verificare se tra due nodi dati in input esiste
    un cammino che li collega

def is_path(a, b, W, V):
  flag = False
  if (a, b) in W:
    flag = True
    return flag
  else:
    for c in range(len(V)):
      e = V[c]
      if (a, e[0]) in W:
        flag = is_path(e[0], b, W, V)
      else:
        continue

  return flag

def riordina(W, minimo, W1):
  for i in range(len(W)):
    arc = W[i]
    e0 = arc[0]
```

```
    e1 = arc[1]
    if e0[0] == minimo and arc not in W1:
      W1.append(arc)

  for j in range(len(W)):
    arco = W[j]
    a0 = arco[0]
    a1 = arco[1]
    if a0[0] == minimo:
      W1 = riordina(W, a1[0], W1)

  return W1


# aggiorna le label dei nodi in base al mapping

def aggiorna_label(W, map, V):
  W1 = []
  V1 = []

  for i in range(len(W)):
    arc = W[i]
    a0 = arc[0]
    a1 = arc[1]
    for j in range(len(map)):
      e = map[j]
      if a0 == (e[2], e[0]):
        for k in range(len(map)):
          f = map[k]
          if a1 == (f[2], f[0]):
            W1.append(((e[1], e[0]), (f[1], f[0])))

  for i1 in range(len(V)):
    node = V[i1]
    for j1 in range(len(map)):
      e = map[j1]
      if node == (e[2], e[0]):
        V1.append((e[1], e[0]))

  return W1, V1


# effettua il sorting di W in base al numero non usato

def sorting(W, V):
  n = 0
  W_sorted = []

  while n < len(V):
    n += 1
    for i in range(len(W)):
```

```
      arc = W[i]
      a0 = arc[0]
      if a0[0] == n:
        W_sorted.append(arc)

  return W_sorted


"""##SAVE FILE

"""


def saveGFile(V, W, path, time, sort_labels):
  with open(path, 'w') as f:
    f.write("# Execution Time: {0:.3f} s\n".format(time))
    # f.write("# Deleted Activities: {0}\n".format(D))
    # f.write("# Inserted Activities: {0}\n".format(I))
    for n in V:
      f.write("v {0} {1}\n".format(n[0], n[1]))
    f.write("\n")
    if (sort_labels):
      W.sort()
    for e in W:
      f.write("e {0} {1} {2}__{3}\n".format(e[0][0], e[1][0], e[0][1],
          e[1][1]))

  print(V)
  print(W)


def saveGfinal(V, W, path, sort_labels):
  with open(path, 'a') as f:
    f.write("XP \n")
    for n in V:
      f.write("v {0} {1}\n".format(n[0], n[1]))
    if (sort_labels):
      W.sort()
    for e in W:
      f.write("e {0} {1} {2}__{3}\n".format(e[0][0], e[1][0], e[0][1],
          e[1][1]))
    f.write("\n")
    f.close()


def isolato(node, W):
  flag = True
  for i in range(len(W)):
    arc = W[i]
    if arc[0] == node or arc[1] == node:
      flag = False
```

```
  return flag


def saveCSV(path, aligned, model_moves, num):
  with open(path, 'a') as f:
    writer = csv.writer(f, delimiter=";")
    writer.writerow([num, aligned, model_moves])
    f.close()


"""##MAIN"""


def BIG(net_path, log_path, tr_start=0, tr_end=None, view=False,
    sort_labels=False):

  start_time= time.time()

  splits = log_path.split('/')
  name = splits[-1].split(".")[0]

  streaming_ev_object = xes_importer.apply(log_path,
      variant=xes_importer.Variants.XES_TRACE_STREAM)  # file xes
  net, initial_marking, final_marking = pnml_importer.apply(net_path)

  gviz = pn_visualizer.apply(net, initial_marking, final_marking)
  gviz.render(filename="petri")

  cr = findCausalRelationships(net, initial_marking, final_marking)
  if view:
    print(cr)

  n = 0

  start_time_total = time.time()

  with open("{0}_instance_graphs.csv".format(name), 'w') as f:
    writer = csv.writer(f, delimiter=";")
    writer.writerow(['n', 'aligned to model', 'with invisible moves'])
    f.close()

  for trace in streaming_ev_object:
    n += 1
    print(n)
    Aligned, A = pick_aligned_trace(trace, net, initial_marking,
        final_marking)
    Align = Aligned[0]
    # L1 = L[0]
    A1 = A[0]
    print('Aligned to model')
```

```
print(Align)
print('with invisible moves')
print(A1)
map, ins = mapping(Align, A1)

saveCSV("{0}_instance_graphs.csv".format(name), Align, A1, n)


compliant = compliant_trace(Align)
effettiva = compliant_trace(A1)

print(compliant)
print('Effettiva: ', effettiva)

print("map: ", map)
print("ins: ", ins)

d = []

trace_start_time = time.time()
num = trace.attributes.get('concept:name')
id = trace.attributes.get('variant-index')

V, W = ExtractInstanceGraph(compliant, cr)
print('V')
print(V)
print('W')
print(W)
if view:
  print("\n\n-----------------------------------\nUnrepaired
      Instance Graph")

V_n = node_number(V)
W_n = edge_number(W)

for element in map:  # crea le liste Eremp e Erems
  if element[1] == 0:
    d.append(element)

if n == 991:
  graph = viewInstanceGraph(V, W)

Vpos = []
for node in V:
  Vpos.append(node)

print('Vpos1: ', Vpos)

for el in map:
  if el[1] == 0:
    Vpos.remove((el[2], el[0]))
```

```
print('Vpos = ', Vpos)

print('INSERTION')

for insertion in ins:
  V, W = ins_repair(V, W, map, insertion, V_n, ins, Vpos)

print('W repaired: ', W)

if n == 991:
  graph = viewInstanceGraph(V, W)

print('DELETION')
print(d)

for deletion in d:
  V, W = del_repair(V, W, map, deletion)


print('W aggiornata: ', W)
print("V aggiornata: ", V)

minimo = min(V_n)

W1 = []

W1 = riordina(W, minimo, W1)

W1, V1 = aggiorna_label(W, map, V)

print('W1 label: ', W1)

print('W ordinata: ', W)

if n == 991:
  graph = viewInstanceGraph(V1, W1)

elapsed = time.time() - start_time_total

V1.sort()
W1.sort()

print(trace)
saveGFile(V1, W1,
    "/Users/jozie/Desktop/NewBigOriginale/results/"+str(n),
    time.time() - trace_start_time, sort_labels)
saveGfinal(V1, W1, "{0}_instance_graphs.g".format(name), sort_labels)

print('----------------------------------------------------------------')
```

```
end_time=time.time()
tempo_di_esecuzione= end_time-start_time
print("Il tempo di esecuzione     "+str(tempo_di_esecuzione))
```

Testing

```
<trace>
        <string key="variant" value="Variant 199"/>
        <int key="variant-index" value="199"/>
        <string key="creator" value="Fluxicon Disco"/>
        <string key="concept:name" value="trace_0"/>
        <event>
            <string key="org:resource" value="NONE"/>
            <string key="lifecycle:transition" value="complete"/>
            <string key="concept:name" value="SRP"/>
            <date key="time:timestamp"
                value="2013-12-01T08:01:24.772+01:00"/>
        </event>
        <event>
            <string key="org:resource" value="NONE"/>
            <string key="lifecycle:transition" value="complete"/>
            <string key="concept:name" value="SRPP"/>
            <date key="time:timestamp"
                value="2013-12-01T09:01:24.772+01:00"/>
        </event>
        <event>
            <string key="org:resource" value="NONE"/>
            <string key="lifecycle:transition" value="complete"/>
            <string key="concept:name" value="RBPC"/>
            <date key="time:timestamp"
                value="2013-12-01T10:01:24.772+01:00"/>
        </event>
        <event>
            <string key="org:resource" value="NONE"/>
            <string key="lifecycle:transition" value="complete"/>
            <string key="concept:name" value="RIBPC"/>
            <date key="time:timestamp"
                value="2013-12-01T11:01:24.772+01:00"/>
        </event>
        <event>
            <string key="org:resource" value="NONE"/>
            <string key="lifecycle:transition" value="complete"/>
            <string key="concept:name" value="REPC"/>
            <date key="time:timestamp"
                value="2013-12-01T12:01:24.772+01:00"/>
        </event>
        <event>
            <string key="org:resource" value="NONE"/>
```

```
        <string key="lifecycle:transition" value="complete"/>
        <string key="concept:name" value="FRPP"/>
        <date key="time:timestamp"
            value="2013-12-01T13:01:24.772+01:00"/>
    </event>
    <event>
        <string key="org:resource" value="NONE"/>
        <string key="lifecycle:transition" value="complete"/>
        <string key="concept:name" value="REPP"/>
        <date key="time:timestamp"
            value="2013-12-01T14:01:24.772+01:00"/>
    </event>
    <event>
        <string key="org:resource" value="NONE"/>
        <string key="lifecycle:transition" value="complete"/>
        <string key="concept:name" value="REPP"/>
        <date key="time:timestamp"
            value="2013-12-01T14:31:24.772+01:00"/>
    </event>
    <event>
        <string key="org:resource" value="NONE"/>
        <string key="lifecycle:transition" value="complete"/>
        <string key="concept:name" value="SRPP"/>
        <date key="time:timestamp"
            value="2013-12-01T15:01:24.772+01:00"/>
    </event>
    <event>
        <string key="org:resource" value="NONE"/>
        <string key="lifecycle:transition" value="complete"/>
        <string key="concept:name" value="RIBPC"/>
        <date key="time:timestamp"
            value="2013-12-01T16:01:24.772+01:00"/>
    </event>
    <event>
        <string key="org:resource" value="NONE"/>
        <string key="lifecycle:transition" value="complete"/>
        <string key="concept:name" value="REPC"/>
        <date key="time:timestamp"
            value="2013-12-01T17:01:24.772+01:00"/>
    </event>
    <event>
        <string key="org:resource" value="NONE"/>
        <string key="lifecycle:transition" value="complete"/>
        <string key="concept:name" value="RBPC"/>
        <date key="time:timestamp"
            value="2013-12-01T18:01:24.772+01:00"/>
    </event>
    <event>
        <string key="org:resource" value="NONE"/>
        <string key="lifecycle:transition" value="complete"/>
        <string key="concept:name" value="FRPP"/>
```

```
    <date key="time:timestamp"
        value="2013-12-01T19:01:24.772+01:00"/>
</event>
<event>
    <string key="org:resource" value="NONE"/>
    <string key="lifecycle:transition" value="complete"/>
    <string key="concept:name" value="REPP"/>
    <date key="time:timestamp"
        value="2013-12-01T20:01:24.772+01:00"/>
</event>
<event>
    <string key="org:resource" value="NONE"/>
    <string key="lifecycle:transition" value="complete"/>
    <string key="concept:name" value="FRPP"/>
    <date key="time:timestamp"
        value="2013-12-01T21:01:24.772+01:00"/>
</event>
<event>
    <string key="org:resource" value="NONE"/>
    <string key="lifecycle:transition" value="complete"/>
    <string key="concept:name" value="RBPC"/>
    <date key="time:timestamp"
        value="2013-12-01T22:01:24.772+01:00"/>
</event>
<event>
    <string key="org:resource" value="NONE"/>
    <string key="lifecycle:transition" value="complete"/>
    <string key="concept:name" value="RIBPC"/>
    <date key="time:timestamp"
        value="2013-12-01T23:01:24.772+01:00"/>
</event>
<event>
    <string key="org:resource" value="NONE"/>
    <string key="lifecycle:transition" value="complete"/>
    <string key="concept:name" value="REPC"/>
    <date key="time:timestamp"
        value="2013-12-02T00:01:24.772+01:00"/>
</event>
<event>
    <string key="org:resource" value="NONE"/>
    <string key="lifecycle:transition" value="complete"/>
    <string key="concept:name" value="SRPP"/>
    <date key="time:timestamp"
        value="2013-12-02T01:01:24.772+01:00"/>
</event>
<event>
    <string key="org:resource" value="NONE"/>
    <string key="lifecycle:transition" value="complete"/>
    <string key="concept:name" value="EPP"/>
    <date key="time:timestamp"
        value="2013-12-02T02:01:24.772+01:00"/>
```

```
        </event>
        <event>
            <string key="org:resource" value="NONE"/>
            <string key="lifecycle:transition" value="complete"/>
            <string key="concept:name" value="SLRRP"/>
            <date key="time:timestamp"
                value="2013-12-02T03:01:24.772+01:00"/>
        </event>
        <event>
            <string key="org:resource" value="NONE"/>
            <string key="lifecycle:transition" value="complete"/>
            <string key="concept:name" value="LRERV"/>
            <date key="time:timestamp"
                value="2013-12-02T04:01:24.772+01:00"/>
        </event>
        <event>
            <string key="org:resource" value="NONE"/>
            <string key="lifecycle:transition" value="complete"/>
            <string key="concept:name" value="LRIRV"/>
            <date key="time:timestamp"
                value="2013-12-02T05:01:24.772+01:00"/>
        </event>
        <event>
            <string key="org:resource" value="NONE"/>
            <string key="lifecycle:transition" value="complete"/>
            <string key="concept:name" value="FLRRP"/>
            <date key="time:timestamp"
                value="2013-12-02T06:01:24.772+01:00"/>
        </event>
        <event>
            <string key="org:resource" value="NONE"/>
            <string key="lifecycle:transition" value="complete"/>
            <string key="concept:name" value="LRRR"/>
            <date key="time:timestamp"
                value="2013-12-02T07:01:24.772+01:00"/>
        </event>
        <event>
            <string key="org:resource" value="NONE"/>
            <string key="lifecycle:transition" value="complete"/>
            <string key="concept:name" value="FRP"/>
            <date key="time:timestamp"
                value="2013-12-02T08:01:24.772+01:00"/>
        </event>
    </trace>
```

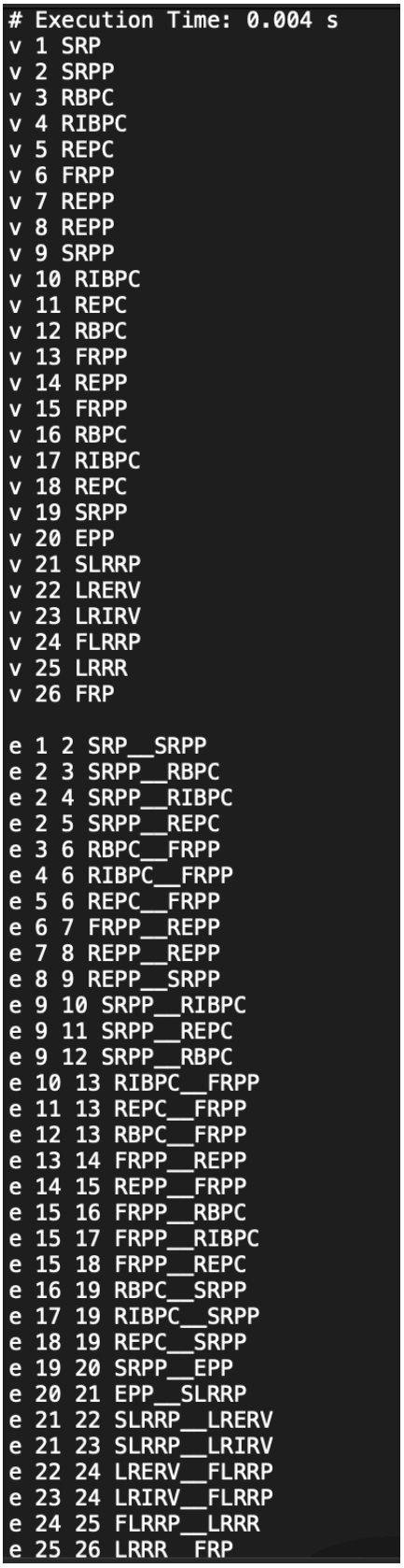

Figura 1: Output esecuzione BIG

```
# Execution Time: 0.006 s
v 1 SRP
v 2 SRPP
v 3 RBPC
v 4 RIBPC
v 5 REPC
v 6 FRPP
v 7 REPP
v 8 REPP
v 9 SRPP
v 10 RIBPC
v 11 REPC
v 12 RBPC
v 13 FRPP
v 14 REPP
v 15 FRPP
v 16 RBPC
v 17 RIBPC
v 18 REPC
v 19 SRPP
v 20 EPP
v 21 SLRRP
v 22 LRERV
v 23 LRIRV
v 24 FLRRP
v 25 LRRR
v 26 FRP

e 1 2 SRP__SRPP
e 2 3 SRPP__RBPC
e 2 4 SRPP__RIBPC
e 2 5 SRPP__REPC
e 3 6 RBPC__FRPP
e 4 6 RIBPC__FRPP
e 5 6 REPC__FRPP
e 6 7 FRPP__REPP
e 7 8 REPP__REPP
e 8 9 REPP__SRPP
e 9 10 SRPP__RIBPC
e 9 11 SRPP__REPC
e 9 12 SRPP__RBPC
e 10 13 RIBPC__FRPP
e 11 13 REPC__FRPP
e 12 13 RBPC__FRPP
e 13 14 FRPP__REPP
e 14 15 REPP__FRPP
e 15 16 FRPP__RBPC
e 15 17 FRPP__RIBPC
e 15 18 FRPP__REPC
e 16 19 RBPC__SRPP
e 17 19 RIBPC__SRPP
e 18 19 REPC__SRPP
e 19 20 SRPP__EPP
e 20 21 EPP__SLRRP
e 21 22 SLRRP__LRERV
e 21 23 SLRRP__LRIRV
e 22 24 LRERV__FLRRP
e 23 24 LRIRV__FLRRP
e 24 25 FLRRP__LRRR
e 25 26 LRRR__FRP
```

Figura 2: Output esecuzione BIG[2]

Bibliografia

[1] Laura Genga. From event logs to subprocesses:supporting the analysis of unstructured processes. 2016. https://research.tue.nl/en/persons/laura-genga.

[2] Apache Foundation. Apache spark. https://spark.apache.org.

[3] Apache Foundation. Spark sql and dataframe. https://spark.apache.org/sql/.

[4] Fraunhofer Institute for Applied Information Technology (FIT). Pm4py. https://pm4py.fit.fraunhofer.de.

[5] Xml data source for apache spark. https://github.com/databricks/spark-xml.

Ringraziamenti

Volevo scrivere un ringraziamento con una buona dose di pessimismo ma ho promesso di essere un pò più ottimista quindi non lo farò.
Durante questi anni ho vissuto molto. Prima della pandemia sono entrato in Croce Gialla, dove ho conosciuto tanta persone fantastiche, come Mariano, MariaFrancesca, Francesca, Davide, Leo, con cui ho condiviso molto e a cui ho lasciato un pezzo di cuore.
Durante la pandemia, sono tornato a casa e ho potuto trascorre dopo quasi 6 anni di latitanza, del tempo con mia sorella scoprendo che ragazza spettacolare sia diventata e che ringrazio per il supporto che mi ha dato durante questi anni di ritorno al nido.
Ringrazio mamma e papa per il continuo supporto che mi danno.
Vorrei ringraziare anche i miei amici di Pescara per i bei momenti trascorsi insieme.
Infine vorrei ringraziare Chiara, ragazza che mi sta sopportando e supportando ormai da un anno, per il suo aiuto in questo periodo folle tra lavoro e tesi. GRAZIE.

Ancona, Luglio 2022

Jacopo Iezzi