Caso di studio in Artificial Intelligence

1 Studenti

Nome Cognome, matricola, mail
Federico Canistro, 775605, f.canistro@studenti.uniba.it
Ivan De Cosmis, 787066, i.decosmis@studenti.uniba.it

2 Confronto con sperimentazioni precedenti

Nel caso di studio precedente, si é utilizzata la colonna *timestamp* nel file CSV come identificatore del caso durante la conversione del data frame in un event log. Ció significa che ogni riga nel file CSV veniva trattata come un caso separato con un singolo evento. Di conseguenza, la durata di ogni caso é sempre 0.0 perché é presente solo un evento in ogni caso.
Per calcolare correttamente la durata del caso, é stato necessario specificare una colonna nel file CSV che identifica univocamente ogni caso quando si chiama la funzione *pm4py.format_dataframe*.
Nella nuova versione del codice, stiamo utilizzando la colonna *id_exec* del file CSV come identificatore del caso quando convertiamo il data frame in un event log. Ció significa che le righe nel file CSV che hanno lo stesso valore per la colonna *id_exec*, verranno raggruppate nella stessa traccia dentro l' event log.
La differenza rispetto all' event log precedente é che ora le tracce nell'event log possono contenere piú di un evento. Ció consente di calcolare statistiche come la durata del caso, che misura il tempo trascorso tra il primo e l'ultimo evento in ogni traccia.
Ogni output degli algoritmi utilizzati viene salvato in un file pnml e anche in formato png per poterne osservare la sua visualizzazione grafica.

2.1 Funzioni secondarie

get_all_duration_case: controlla la durata di tutti i case.
compare_pnml_files: Confronta il numero di 'place' e 'transition' nei file.

2.2 Euristic Miner

L'Heuristics Miner é un algoritmo di scoperta di processi piú avanzato che puó gestire meglio le caratteristiche comuni degli event log. Di conseguenza, l'Heuristics Miner é stato in grado di produrre un modello di rete di Petri piú accurato rispetto gli altri algoritmi per lo stesso event log.

2.3 Alpha Miner

L'algoritmo Alpha Miner é un algoritmo di scoperta di processi che puó essere utilizzato per scoprire un modello di rete di Petri da un event log. Tuttavia, l'Alpha Miner ha alcune limitazioni e non é stato in grado di gestire correttamente alcune caratteristiche comuni degli event log, come le attivitá in parallelo e le dipendenze a lungo termine tra le attivitá. In questo caso, l'Alpha Miner ha prodotto un modello di rete di Petri che non rappresenta accuratamente il processo sottostante.

Regular Paper

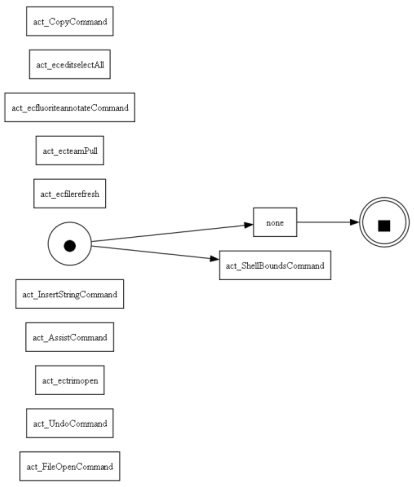

Fig.1. Parte di output alpha sul dataset completo.

2.4 Inductive Miner

L'Inductive Miner é un algoritmo di scoperta di processi che puó essere utilizzato per scoprire un modello di rete di Petri da un event log. In genere, l'Inductive Miner é in grado di gestire event log di grandi dimensioni e complessi in modo efficiente. Tuttavia, in alcuni casi, l'Inductive Miner potrebbe richiedere piú tempo per scoprire un modello di rete di Petri, a seconda delle caratteristiche dell'event log e dei parametri utilizzati. Nel nostro caso l'inductive miner é stato pi1ú di un'ora in esecuzione sul dataset completo e non si é comunque fermato. Mentre testandolo sui dataset separati ha risposto molto bene.

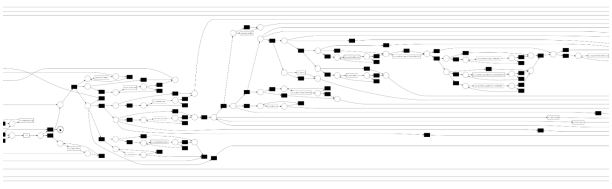

Fig.1. parte di output inductive corretto su uno dei dataset.

Abbiamo inoltre provato ad utilizzare varianti dell'algoritmo Inductive miner, ma sempre senza successo (sempre sul dataset completo). Inoltre, abbiamo anche provato a modificare i parametri dell'algoritmo per migliorarne le prestazioni. Abbiamo utilizzato il parametro *noise_threshold* per specificare una soglia di rumore per l'algoritmo. L'aumento della soglia di rumore puó aiutare l'algoritmo a gestire meglio gli event log rumorosi e migliorare le prestazioni, ma nel nostro caso ció ha portato ad output scorretti come nel caso dell'alpha miner.

3 Conversione rete di Petri da formato Woman a formato pnml

L'idea di questo algoritmo é leggere da un file in input tutti i place, transition e arc nel formato:

1. place(X)
2. transition(X)
3. arc(X,Y)

Vengono salvati all'interno di tre liste indipendenti (conversion_to_pnml.pl, 1):

```
% Dichiaro dinamiche le liste per
% poter essere modificate a runtime
:- dynamic string_list1/1.
:- dynamic string_list2/1.
:- dynamic string_list3/1.
```

In seguito vengono creati i predicati per accedere alle liste (conversion_to_pnml.pl, 10):

```
% Aggiunge stringa nella lista in input (1,2 o 3)
add_string(ListNumber, String) :-
    (   retract(string_list(ListNumber, List))
    ->  assertz(string_list(ListNumber, [String|List]))
    ;   assertz(string_list(ListNumber, [String]))
    ).

% Restituisce la lista in base al numero in input (1,2 o 3)
get_strings(ListNumber, Strings) :-
    (   string_list(ListNumber, Strings)
    ->  true
    ;   Strings = []
    ).
```

3.1 Lettura dei dati dal file

3.1.1 process_file(FileName)

Il predicato che legge il file e salva i place, le transition e gli arc all'interno delle liste si chiama **process_file(FileName)**.
Questo predicato riceve in input il percorso del file da leggere (conversion_to_petri.pl, 25).

```
% Processa il file in input ed esporta il file pnml nuovo
process_file(FileName) :-
    retractall(string_list(1, List)),
    retractall(string_list(2, List)),
    retractall(string_list(3, List)),
    open(FileName, read, Stream),  % Apre il file per la lettura
    process_lines(Stream),         % Inizia ad analizzare le linee
    close(Stream),                 % Chiude il file
    get_strings(1, Places),
    get_strings(2, Transitions),
    get_strings(3, Arcs),
    ask_folder_path(Path),
    write_pnml(Path, Places, Transitions, Arcs).
```

"retractall" viene usato per azzerare tutte le liste ogni volta che viene consultato il file. "process_lines" é il predicato che processa le linee dello Stream di input.

3.1.2 process_lines(Stream)

Questo predicato ha in inputo lo stream da cui leggere le informazioni. Possiamo trovare due casi:

1. Caso base in cui lo stream da leggere é finito (conversion_to_petri.pl, 43):

   ```
   % Caso in cui lo Stream ha raggiunto la fine del file
   process_lines(Stream) :-
       at_end_of_stream(Stream).
   ```

2. Caso in cui esistono ancora righe da analizzare (conversion_to_petri.pl, 47):

   ```
   % Dato lo stream in input lo analizzo
   % e richiamo la funzione con il nuovo Stream
   process_lines(Stream) :-
       \+ at_end_of_stream(Stream),
       read_line_to_string(Stream, Line),
       process_line(Line),
       process_lines(Stream).
   ```

Nel secondo caso troviamo il predicato **process_line(Line)**

3.1.3 process_line(Line)

Questo predicato dalla stringa in input trova l'atomo, che é utile a identificare se la riga letta é un place, una transition o un arc.

3.1.4 process_term(Term)

Questo predicato ha tre casi ognuno che richiama la funzione necessaria ad estrarre il tipo di dato da leggere:

1. Caso place (conversion_to_petri.pl, 59):

   ```
   % Caso in cui leggo una riga place
   process_term(Term) :-
       sub_atom(Term, _, _, _, 'place'),
       read_places(Term).
   ```

   In questo caso andiamo a chiamare la funzione "read_places(Term)"

2. Caso transition (conversion_to_petri.pl, 64):

   ```
   % Caso in cui leggo una rica transition
   process_term(Term) :-
       sub_atom(Term, _, _, _, 'transition'),
       read_transition(Term).
   ```

   In questo caso andiamo a chiamare la funzione "read_transition(Term)"

3. Caso arc (conversion_to_petri.pl, 69):

   ```
   % Caso in cui leggo una riga arc
   process_term(Term) :-
       sub_atom(Term, _, _, _, 'arc'),
       read_arc(Term).
   ```

   In questo caso andiamo a chiamare la funzione "read_arc(Term)"

3.1.5 read_places(Term) e read_transition(Term)

Questi due predicati sono molto simili, la differenza é che salvano il valore ottenuto in liste diverse. Il loro funzionamento é il seguente:

1. Richiamano la funzione "extract_info(Line, Info)"
2. Questa funzione conta quanti caratteri ci sono prima del carattere "("
3. In seguito crea una sottostringa tagliando i caratteri fino al carattere "("
4. Infine elimina il primo e ultimo carattere (che sarebbero le due parentesi), facendo rimanere solamente la stringa tra parentesi.

(conversion_to_petri.pl, 84)

```
% Leggo arco dal Term in input e lo metto nella terza lista
read_arc(Term) :-
    extract_arc(Term, Info),
    add_string(3, Info).

% Estraggo il contenuto tra parentesi e lo restituisco come Info
extract_info(Line, Info) :-
    sub_string(Line, Before, _, After, '('),
    sub_string(Line, Before, _, 0, InfoWithParenthesis),
    sub_string(InfoWithParenthesis, 1, _, 1, Info).
```

3.1.6 read_arc(Term)

Questo predicato richiama **extract_arc(Line, Info)** (conversion_to_petri.pl, 84):

```
% Estraggo arco dalla linea in input e lo restituisco come Info
extract_arc(Line, Info) :-
    sub_string(Line, Before, _, After, '('),
    sub_string(Line, BeforeComma, _, AfterComma, ','),
    sub_string(Line, Before, _, AfterComma, FirstInfoDirty),
    sub_string(FirstInfoDirty, 1, _, 1, FirstInfo),
    sub_string(Line, BeforeComma, _, 0, SecondInfoDirty),
    sub_string(SecondInfoDirty, 1, _, 1, SecondInfo),
    Info = (FirstInfo, SecondInfo).
```

Funziona nel seguente modo:

1. Richiamano la funzione "extract_arc(Line, Info)"
2. Per prima cosa conta quanti caratteri ci sono prima del carattere "("
3. Dopo conta quanti caratteri ci sono prima e dopo il carattere ","
4. In seguito crea una sottostringa tagliando i caratteri fino al carattere "(" e tagliando i caratteri dopo ","
5. Il primo oggetto viene salvato cancellando il primo carattere "(", e l'ultimo carattere ","
6. Per salvare il secondo oggetto taglia tutti i caratteri prima del carattere ","
7. Elimina il primo "," e ultimo carattere ")", facendo rimanere solamente la stringa tra parentesi ottenendo il secondo oggetto.
8. Infine crea una coppia con i due oggetti.

Con questa funzione abbiamo finito di analizzare la parte relativa alla lettura dei dati da file in input.

3.2 Scrittura del file pnml

Per scrivere nel file pnml utilizziamo il predicato **write_pnml(File, Places, Transitions, Arcs)**.

3.2.1 write_pnml(File, Places, Transitions, Arcs)

Questa funzione apre il file del percorso inserito in console e inizia a scrivere le varie parti del file pnml (conversion_to_petri.pl, 122).

```
% Scrivo il file pnml dati place, transition e arc
write_pnml(File, Places, Transitions, Arcs) :-
    open(File, write, StreamWrite),
    write_header(StreamWrite),
    write_places(StreamWrite, Places),
    write_transitions(StreamWrite, Transitions),
    write_arcs(StreamWrite, Arcs),
    write_footer(StreamWrite),
    close(StreamWrite).
```

3.2.2 write_header(StreamWrite)

Questa funzione scrive l'header del file pnml (conversion_to_petri.pl, 133).

```
% Scrivo intestazione file
write_header(StreamWrite) :-
    write(StreamWrite, '<?xml version="1.0" encoding="UTF-8"?>'),
    nl(StreamWrite),
    write(StreamWrite, '<pnml>'), nl(StreamWrite),
    write(StreamWrite, '  <net>'), nl(StreamWrite).
```

3.2.3 write_places, write_transitions, write_arcs

Dato che sono tutti molto simili, cambia solamente la formattazione per rispettare il formato pnml, prendiamo in analisi **write_places** Questo predicato ha due casi:

1. Caso base (conversion_to_petri.pl, 140):

   ```
   % Questo predicato serve quando abbiamo
   % finito tutti i place
   write_places(StreamWrite, []) :-
       true.
   ```

   Questo é il caso in cui sono finiti i place da scrivere su file

2. Caso con lista piena (conversion_to_petri.pl, 144):

   ```
   % Questo predicato serve quando
   % ci sono ancora place da scrivere
   write_places(StreamWrite, [Place|Places]) :-
       write(StreamWrite, '    <place id="'),
       counter(Count),
       write(StreamWrite, Count),write(StreamWrite, '">'),
       nl(StreamWrite),
       increment_counter,
       write(StreamWrite, '      <name>'),
       write(StreamWrite, Place),
       write(StreamWrite, '</name>'),
       nl(StreamWrite),
       write(StreamWrite, '    </place>'),
       nl(StreamWrite),
       write_places(StreamWrite, Places).
   ```

   In questo caso ci sono place da scrivere su file e grazie alla funzione write andiamo a scrivere il place rispettando la sintassi del formato pnml. "increment_counter" aumenta il contatore che rappresenta l'id dei place, transition e arc (chiedere professore problema con contatore). Infine la funzione richiama se stessa iterando con la lista senza il primo elemento.

3.2.4 write_footer

Questo predicato scrive il pié di pagina del file pnml (conversion_to_petri.pl, 183)

```
% Questo predicato serve quando dobbiamo
% scrivere il pie di pagina pnml
write_footer(StreamWrite) :-
    write(StreamWrite, '  </net>'), nl(StreamWrite),
    write(StreamWrite, '</pnml>'), nl(StreamWrite).
```