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

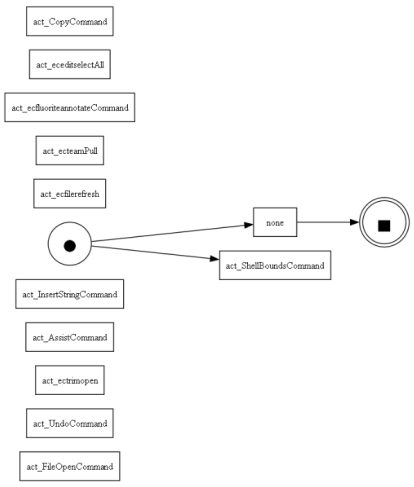

Fig.1. Parte di output alpha sul dataset completo.

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.

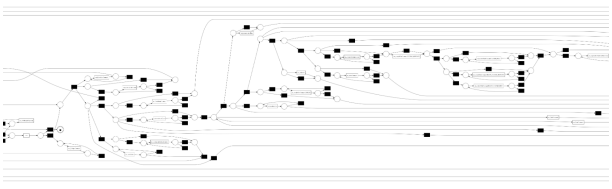

Fig.1. parte di output inductive corretto su uno dei dataset.

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 pl 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, 24).

```
% 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)),
    dynamic(counter/1),
    consult(FileName),
    process_net,        % Inizia ad analizzare le linee
    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_net" é il predicato che processa la rete.

3.1.2 process_net

In questo predicato troviamo le operazioni che vengono eseguite per estrarre i dati dai fatti letti in input:

1. extract_places(NewPlaces): estrae tutti i place dal database e li salva all'interno della lista NewPlaces
2. copy_list(1, NewPlaces): copia la lista in input nella lista dinamica 1
3. extract_transitions(NewTransitions): estrae tutti le transition dal database e li salva all'interno della lista NewTransitions
4. copy_list(2, NewTransitions): copia la lista in input nella lista dinamica 2
5. extract_arcs(NewArcs): estrae tutti gli arc dal database e li salva all'interno della lista NewArcs
6. copy_list(3, NewArcs): copia la lista in input nella lista dinamica 3

(conversion_to_pnml.pl, 65)

```
% Estraggo le informazioni dai fatti
process_net :-
    extract_places(NewPlaces),
    copy_list(1, NewPlaces),
    extract_transitions(NewTransitions),
    copy_list(2, NewTransitions),
    extract_arcs(NewArcs),
    copy_list(3, NewArcs).
```

3.1.3 extract_places(Places)

Analizziamo questo predicato che é molto simile a **extract_transitions(Transitions)** e **extract_arcs(Arcs)**, cambia solamente la sintassi in base al tipo di dato da estrarre (conversion_to_pnml.pl, 43):

```
% Estraggo tutti i place in una lista
extract_places(Places) :-
    findall(Place, place(Place), Places).
```

Il predicato findall ricerca tuttii fatti nel formato **place(Place)** e li salva nella lista **Places**.

3.1.4 copy_list

Con questo predicato abbiamo due casi (conversion_to_pnml.pl, 55):

1. copy_list(IdList, []): Caso in cui la lista in input é vuota e quindi abbiamo finito la copia
2. copy_list(IdList, [H|T]): Caso in cui la lista bisogna ancora copiarla tutta

```
% Predicato che ci permette di copiare una lista
% in una delle 3 liste dinamiche dato indice in input
copy_list(IdList, []) :-
    true.

copy_list(IdList, [H | T]) :-
    add_string(IdList, H),
    copy_list(IdList, T).
```

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, 90).

```
% 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 ask_folder_path(String)

Con questo predicato chiediamo il percoso in cui salvare il file di output (Il percorso alla fine deve includere anche il nome del file e l'estensione pnml) (conversion_to_petri.pl, 38).

```
% Chiedo dove salvare il file pnml
ask_folder_path(String) :-
    write('Enter path where to save file: '),
    read_line_to_string(user_input, String).
```

3.2.3 *write_header(StreamWrite)*

Questa funzione scrive l'header del file pnml (conversion_to_petri.pl, 101).

```
% Scrivo intestazione file
write_header(StreamWrite) :-
    write(StreamWrite, '<?xml version="1.0" encoding="UTF-8"?>'),
    nl(StreamWrite),
    ...
    (Abbrevio il codice nella relazione per problemi di spazio)
```

3.2.4 *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, 112):

   ```
   % 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 ancora da svuotare (conversion_to_petri.pl, 116):

   ```
   % Questo predicato serve quando
   % ci sono ancora place da scrivere
   write_places(StreamWrite, [Place|Places]) :-
       write(StreamWrite, '        <place id="'),
       counter(Count),
       ...
       (Abbrevio il codice nella relazione per problemi di spazio)
       ...
       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. Infine la funzione richiama se stessa iterando con la lista senza il primo elemento.

3.2.5 *write_footer*

Questo predicato scrive il pié di pagina del file pnml (conversion_to_petri.pl, 172)

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