UNIVERSITA' POLITECNICA DELLE MARCHE

FACOLTA' DI INGEGNERIA

Corso di Laurea magistrale in Ingegneria Informatica e dell'Automazione

Implementazione di un algoritmo di process repairing per la riparazione di modelli di processo a partire da local instance graph

Implementation of a process repairing algorithm for the repair of process models starting from local instance graph

Relatore:

Prof. DOMENICO POTENA

Tesi di Laurea di:

DAVIDE MANZONI

A.A. 2020 / 2021

Ringraziamenti

Desidero ringraziare prima di tutto i miei genitori e i miei familiari, che mi hanno sempre incoraggiato e sostenuto anche nei momenti più difficili.

Un forte abbraccio va alle persone che mi sono sempre state vicine, hanno sempre creduto in me e che mi hanno spinto a tirare fuori il meglio.

Un ringraziamento particolare va anche al Prof. Domenico Potena, relatore di questa tesi di laurea, oltre che per l'aiuto fornitomi in tutti questi mesi, per la disponibilità e il sostegno dimostratemi durante tutto il periodo di realizzazione della tesi.

INDICE

Capitolo 1

INTRODUZIONE

1.1 Prefazione

Oggigiorno, in un mondo con uno di stile di vita sempre più frenetico e rapido, le aziende hanno la necessità di soddisfare richieste sempre più esigenti da parte dei propri utenti finali, in termini di qualità e velocità nella realizzazione di prodotti ed erogazione di servizi.
Per questo le aziende cercano di rendere i loro processi interni sempre più efficienti, così da minimizzare i costi e i tempi per la loro esecuzione, per supportare concretamente le loro strategie organizzative e per fronteggiare un mercato che diventa sempre più globalizzato e competitivo. Proprio per questo motivo i programmi software di gestione dei processi di business stanno diventando sempre più importanti e richiesti, in quanto consentono di gestire e ottimizzare l'esecuzione dei processi.
Un processo di business viene definito come una serie di attività eseguite in maniera coordinata all'interno di un contesto organizzativo e tecnico, con lo scopo di perseguire un certo obiettivo aziendale. Le attività all'interno di un processo di business possono essere le più disparate: dall'ordinazione di un libro, alla prenotazione di un biglietto aereo, dall'iscrizione ad un sito, alla richiesta di un prestito bancario, e si cerca proprio di sfruttare i dati relativi ad esse per fornire suggerimenti durante l'esecuzione di un processo, identificare colli di bottiglia, prevedere problemi nell'esecuzione, registrare violazioni e raccomandarne contromisure.
Tali processi di business si basano sull'utilizzo di modelli di processo che servono per descrivere le singole attività svolte all'interno di un processo, sia per quanto riguarda i metodi di svolgimento, che la sincronizzazione, le dipendenze logiche e gli strumenti utilizzati nelle varie attività.
Questi modelli si sono subito diffusi grazie alla loro immediata leggibilità, in quanto utilizzano una notazione grafica molto semplice e di facile comprensione per tutti, anche per le persone con un background limitato del contesto aziendale.
L'implementazione di questi modelli di business è stata possibile per la presenza di un'abbondante quantità di dati, dei quali non era possibile disporre in periodo predigitale. In ambito imprenditoriale, la maggior parte dei dati si riferisce ai processi operativi e per questo motivo si tiene traccia di essi all'interno dei sistemi informativi. Ad ogni modo, nonostante la loro raccolta non sia complicata, le organizzazioni hanno difficoltà ad analizzare ed interpretare i dati, ed estrarre informazioni utili da essi per il loro business.

L'implementazione di questi modelli all'interno di contesti reali, ha portato all'introduzione di un sistema di registrazione degli eventi all'interno di file di log. Il log è un file sequenziale che viene utilizzato per registrare, in modo cronologico, tutte le operazioni che vengono eseguite. Per mezzo dei log, è possibile ricostruire tutte le operazioni che sono state attuate, conoscere chi e quando le ha effettuate.
Questo garantisce una grande facilità nell'analisi ed un continuo monitoraggio delle attività presenti nel processo, consentendo così di individuare eventuali punti di dispersione, sia di tempo che di risorse, all'interno del processo. Grazie a queste misurazioni sarà possibile individuare potenziali miglioramenti al fine di rendere il processo più performante, sia per l'azienda che per gli attori coinvolti.
Ed è qui che entra in gioco il concetto di Process Mining. Esso è una disciplina di ricerca relativamente giovane situata tra machine learning e data mining, e tra modellazione e analisi di processi. L'idea di base del process mining, come spiegato nell'articolo [1] pubblicato da Dirk Fahland e Wil M.P. van der Aalst, è quella di modellare, monitorare e migliorare i processi reali estraendo conoscenza dai log, che rappresentano gli eventi registrati all'interno dei sistemi informativi aziendali e che, come già evidenziato, oggi sono disponibili in grandi quantità.
Le tecniche di Process Mining, quindi, mettono in relazione il comportamento osservato (ad esempio i log degli eventi) con il comportamento modellato (il modello di processo sotto forma di BPMN o rete di Petri).
Come indicato dall'articolo [1] di van del Aalst, il process mining può essere identificato come l'elemento che permette il completamento il ciclo BPM. I dati memorizzati dai sistemi informativi sono importanti per ottenere una panoramica migliore sui processi attuali, in modo che possano essere analizzate le eventuali deviazioni e possa essere migliorata la qualità dei modelli.
Le tecniche esistenti permettono solo di scoprire queste differenze ma la riparazione effettiva del modello è lasciata all'utente e non è ancora supportata.
In questo documento di tesi verrà analizzato il problema della riparazione di un modello di processo in modo tale che sia possibile ottenere un nuovo modello che sia conforme al log di eventi (cioè possa riprodurre gli eventi presenti in esso), facendo assimilare tali eventi al modello originale.
Per risolvere il problema, in questa tesi si discute un approccio che, attraverso tecniche di Conformance Checking, permette di collegare al modello originale una porzione di traccia non conforme, in modo che questo possa essere riparato e sia così in grado di eseguire anche comportamenti che al momento non sono riproducibili nel modello.

1.2 Struttura della tesi

Il progetto di tesi è strutturato come segue: nel primo capitolo vengono esplicitati, in maniera generale, l'ambito di applicazione del progetto, gli obiettivi e vengono specificati il linguaggio e le librerie utilizzati.
Nel secondo capitolo, vengono approfonditi i concetti di Process Mining e Process Repairing semplicemente accennati nel primo capitolo.
Nel terzo, vengono descritte tutte le conoscenze teoriche di base che sono necessarie per poter comprendere i vari aspetti dell'algoritmo realizzato in tale progetto di tesi.
Nel quarto capitolo verrà esplicitata e analizzata l'implementazione dell'algoritmo per il Process Repairing su cui è focalizzato il progetto. Verranno analizzate tutte le varie fasi del programma, cercando di spiegarle seguendo un esempio di applicazione dell'algoritmo cercando di rendere così più chiaro il suo funzionamento.
Nel quinto, verranno analizzati i risultati della sperimentazione tramite i valori assunti dalle metriche calcolate e utilizzate per valutare l'effettivo funzionamento dell'algoritmo, mentre nell'ultimo capitolo, per concludere, verranno illustrati i possibili sviluppi futuri del progetto.

Capitolo 2

PROCESS MINING

Nella vita di tutti i giorni, visto l'utilizzo sempre più in grandi quantità di sistemi informativi e l'ampliamento dell'ambito di questi, soprattutto nelle attività in campo amministrativo, economico e produttivo, si è andata espandendo anche la quantità di dati per monitorare e analizzare tutte queste funzioni.
Per facilitare l'analisi di tali attività sono stati introdotti i modelli di processo.
Essi sono una rappresentazione delle diverse attività che sono coinvolte in un processo aziendale e ne definiscono un ordine in base alle relazioni che vi sono tra esse, stabilendo così il flusso di esecuzione. Questi modelli permettono di:

- determinare la complessità di un processo;
- migliorarne la verificabilità, la manutenibilità, la comprensibilità, e la produttività;
- aumentare la capacità di prevedere tempi e costi;
- rendere il processo più facilmente automatizzabile;
- migliorarne la qualità dei prodotti.

Esistono molte notazioni che consentono alle aziende di modellare i loro processi operativi, come ad esempio BPMN o Reti di Petri.
Proprio queste ultime sono il tipo di rappresentazione che verrà preso in considerazione in questo lavoro di tesi, in quanto sono estremamente semplici e intuitive per la descrizione di

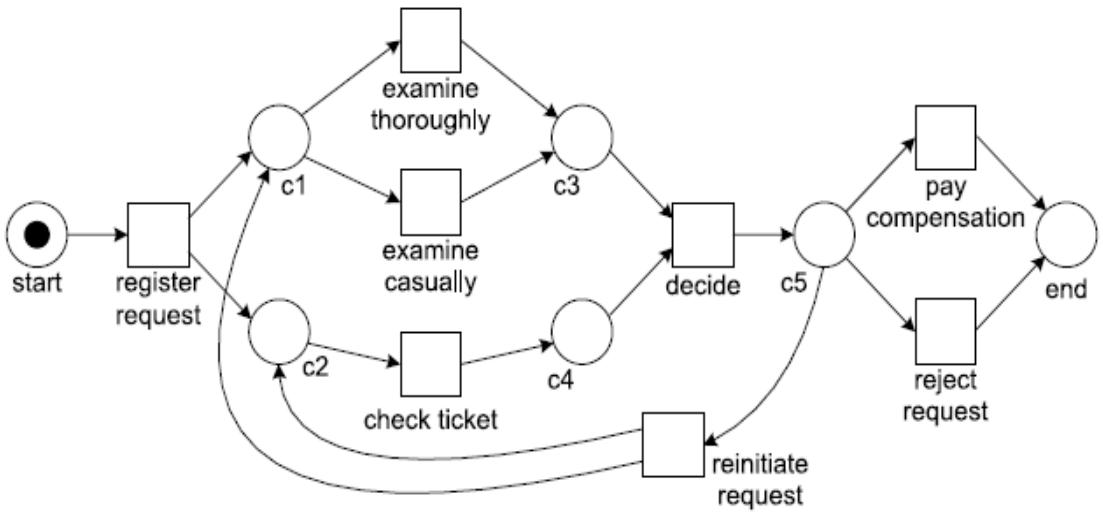


Figura 1: *Un esempio di rete di Petri per la gestione di una richiesta di risarcimento*

processi aziendali, permettendo così di determinare facilmente le varie dipendenze esistenti fra le attività che compongono il processo stesso.
La realizzazione di questi modelli all'interno di contesti aziendali, ha condotto all'introduzione di un sistema di rilevamento dei dati, permettendo un'analisi di ciò che accade nella realtà sulla base di quanto registrato in un file di log.
Un log può contenere eventi di vario tipo: un utente che ritira del denaro contante da uno sportello bancario, un dottore che effettua una visita medica, una persona che fa richiesta per una patente di guida, un contribuente che sottomette una dichiarazione dei redditi o un viaggiatore che prenota un biglietto aereo, sono tutti scenari in cui l'azione viene tracciata in un file di log.
Tutta questa enorme mole di dati fornisce una straordinaria fonte di informazioni e per questo, si presenta la sfida di cercare di sfruttare tutti questi dati in modo significativo, per esempio per fornire suggerimenti durante l'esecuzione di un processo, per identificare colli di bottiglia, prevedere problemi nell'esecuzione, registrare violazioni e raccomandarne contromisure. Lo scopo del process mining è fare esattamente tutto questo.

2.1 Process Mining

Il process mining è una disciplina di ricerca relativamente giovane situata tra machine learning e data mining, e tra modellazione e analisi di processi.
L'idea di base del process mining, come spiegato nell'articolo [1] pubblicato da Dirk Fahland e Wil M.P. van der Aalst, è quella di modellare, monitorare e migliorare i processi reali estraendo conoscenza dai log, che rappresentano gli eventi registrati all'interno dei sistemi informativi aziendali.
Quindi tramite l'attività di process mining è possibile:

- verificare la conformità, e cioè effettuare il monitoraggio di eventuali discrepanze tra un modello ed un file di log;
- mostrare le interdipendenze sussistenti tra le differenti attività;
- comprendere in che modo vengono impiegate le risorse aziendali;
- determinare il livello ottimale di risorse da assegnare al singolo processo aziendale;
- stabilire i costi delle attività aziendali, soprattutto legate a processi di natura produttiva e manageriale;
- valutare la convenienza economica di differenti piani d'azione attraverso la determinazione dei costi delle attività connesse all'alternativa prescelta;
- predire possibili evoluzioni future di un'istanza di processo e suggerire raccomandazioni su come poter operare.

Come detto in precedenza, l'elemento di partenza per qualsiasi tecnica di process mining è sempre un log degli eventi.
Tutte le tecniche di Process Mining prendono in considerazione il fatto che sia possibile memorizzare gli eventi in maniera sequenziale, cosicché ogni evento faccia riferimento ad una certa attività del processo considerato e sia associato ad una particolare istanza di processo.
Un'istanza di processo è una singola esecuzione del processo. Ad esempio prendendo in considerazione il processo di gestione di prestiti emessi da un istituto bancario, ogni esecuzione del processo è relativa alla gestione di una richiesta di prestito.
I log solitamente contengono anche delle informazioni supplementari circa gli eventi come ad esempio le risorse che eseguono o che danno il via ad un'attività, i timestamp o il tipo di transizione eseguita.

Come indicato nell'articolo [3] dal professor van der Aalst, i log di eventi possono essere utilizzati per attuare tre diverse tecniche di Process Mining:

- **Process Discovery**: tecnica in cui si parte da un log di eventi e si produce un modello conforme con i comportamenti presenti su tale log, senza usare altre informazioni a priori.
- **Conformance Checking**: tecnica in cui un modello di processo già esistente viene confrontato con informazioni relative al medesimo processo ma estratte da un log di eventi. Questa tecnica permette di misurare il grado di allineamento tra il modello di processo realizzato e i dati reali estrapolati dal processo stesso, permettendo di verificare se la realtà è conforme al modello creato o vi siano delle divergenze.
- **Process Enhancement**: l'idea di questa tecnica è di estendere o migliorare un modello di processo preesistente con le informazioni contenute nel log di eventi.
  A differenza del Conformance Checking non viene soltanto misurato il livello di allineamento tra il modello e i dati reali, ma si cerca proprio di modificare o estendere il modello esistente e adeguarlo alla realtà.
  Esistono due tipologie di Process Enhancement:
  - *Extension*: estensione del modello aggiungendo una nuova prospettiva al modello di processo a partire da un'analisi incrociata con il log di eventi.
  - *Repairing*: si modifica il modello di processo per fare in modo che rappresenti in maniera più fedele la realtà. Per esempio se delle attività sono rappresentate nel modello in maniera sequenziale, ma nella realtà vengono eseguite in qualsiasi ordine, sarà necessario modificare il modello, per far sì che rappresenti correttamente la successione reale delle attività. Questa è la tipologia che verrà utilizzata in questo lavoro di tesi.

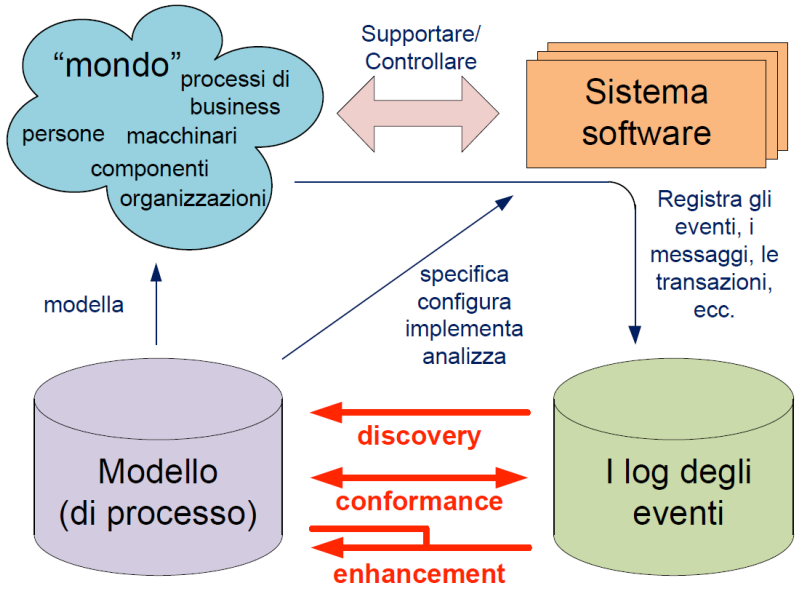


Figura 2: *I tre tipi di Process Minign: Discovery, Conformance Checking, Enhancement*

Come evidenziato precedentemente, uno degli aspetti principali del process mining è l'enfasi con cui si cerca di creare una forte relazione fra il modello di processo e la realtà rilevata nei file di log.
Come descritto nell'articolo [4] sempre del professor van del Aalst, per specificare il tipo di tale relazione si utilizzano i termini Play-out, Play-in e Replay:

- *Play-out:* è il classico approccio che viene attuato con i modelli di processo. Ossia, dato un certo modello di processo, esso viene analizzato per ricavarne gli eventi che l'hanno generato. Questo è utilizzato sia per l'analisi, che per la simulazione con lo scopo di effettuare esperimenti e raccogliere statistiche e intervalli di confidenza.

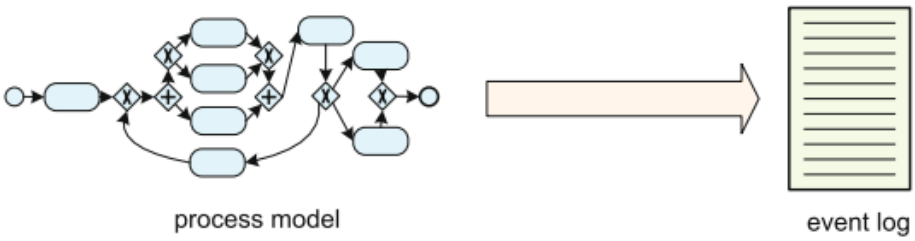


Figura 3: *Play-out*

- *Play-in:* è il contrario del Play-out. In questo caso l'input è costituito dal comportamento tenuto dal processo, rappresentato dall'event log, con lo scopo di andare a estrapolare il modello di tale processo. Questo approccio viene utilizzato nel Process Discovery.

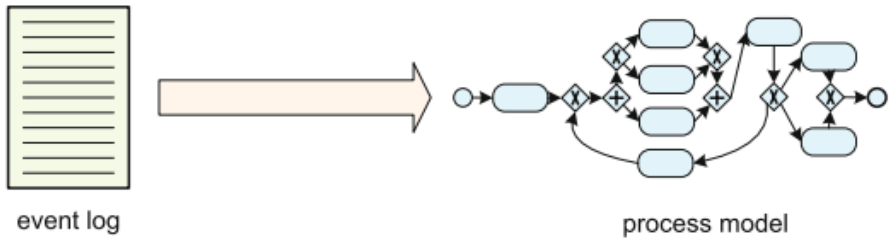


Figura 4: *Play-in*

- *Replay:* utilizza come input sia l'event log che il modello di processo. Si cerca di riprodurre la sequenza di eventi contenuti nel log, ripercorrendo il modello a disposizione. Sono diversi gli scopi per cui può essere utilizzato questo approccio:
  - Conformance Checking: si cerca di individuare le discrepanze tra il modello di processo e l'event log;
  - riparazione ed estensione del modello di processo (Enhancement);
  - costruzione di modelli predittivi per realizzare predizioni di possibili stati che il modello potrebbe assumere in futuro;
  - supporto alle decisioni: vengono rilevate deviazioni a run-time che non si adattano al modello, ed è anche possibile rilevare il tempo rimanente per il processo o la probabilità che il caso in esame non sia completato.

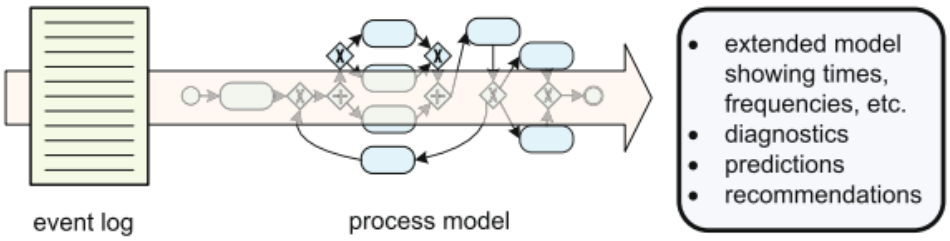


Figura 5: *Replay*

2.2 Process Repairing

L'approccio analizzato e che verrà utilizzato in questo lavoro di tesi, è quello del *Process Repairing*.
Come accennato nel paragrafo precedente, il Process Repairing è una delle tipologie di Process Enhancement.
Esso è un approccio promettente che consiste nel riparare il modello originale in modo che possa riprodurre le attività del registro eventi mantenendolo il più simile possibile al modello di riferimento.
A differenza del Process Discovery, le parti del modello che non sono considerate dall'event log, vengono mantenute nel modello così come sono, andando soltanto ad aggiungere o modificare quelle parti che sono presenti nel log, ma non sono contemplate dal modello.
Di conseguenza il modello riparato risultante dall'applicazione di tale approccio, può essere inteso come una combinazione tra il modello di processo originale e il registro di eventi.
Sono diverse le motivazioni o casi d'uso per i quali si ritiene necessario applicare un approccio di Process Repairing. Uno di questi è quello di migliorare la diagnostica nel Conformance Checking.
Come descritto nel precedente capitolo, il Conformance Checking individua le discrepanze tra il comportamento modellato e quello osservato, mostrando disallineamenti o parti in cui mancano degli elementi durante la riproduzione. Però non è sempre semplice vedere quale sia il vero problema di conformità e poterlo risolvere. Quindi con il Process Repairing, è possibile andare a evidenziare le parti riparate e mostrare in maniera più precisa le varie discrepanze.
Un'ulteriore motivazione è quella del monitoraggio dell'evoluzione dei processi: infatti, considerando una realtà aziendale, i processi possono cambiare nel tempo in quanto i lavoratori potrebbero modificare il modo di gestione delle attività produttive oppure perché potrebbero cambiare proprio le procedure aziendali. Per questo si renderà necessario attuare un processo di riparazione del modello, in quanto alcune parti del vecchio modello non potrebbero più adattarsi alle innovazioni apportate alle procedure o alle attività aziendali.
Oltre ai casi d'uso precedenti, vi è anche quello di effettuare repairing come supporto alla personalizzazione: infatti inizialmente le aziende potrebbero fare uso di un modello di riferimento standard che descrive le migliori pratiche da adottare a livello generale, ma poi i processi effettivi potrebbero discostarsi da tali modelli iniziali e avere delle parti specifiche differenti per diversi ambiti. Di conseguenza, la riparazione del modello viene effettuata per personalizzare il modello di riferimento iniziale adottato, in modo che il nuovo modello descriva più accuratamente la realtà per una specifica unità organizzativa.

Per tutti i casi d'uso indicati precedentemente, è sempre importante che il modello riparato rimanga simile il più possibile al modello originale, in modo da mantenere l'idea di base del modello e evidenziare più chiaramente le differenze.
Però effettuando la riparazione del modello originale, si potrebbero andare ad attuare delle modifiche non desiderabili, in quanto permetterebbero ad esso di adattarsi a comportamenti rari o poco frequenti, ma che complicherebbero troppo il modello stesso.
Per questo, come verrà effettuato in questo lavoro di tesi, si può scegliere di riparare il modello andando ad utilizzare solo comportamenti devianti osservati frequentemente, in modo che il modello riesca a comprendere la maggior parte dei comportamenti, senza diventare eccessivamente complicato da interpretare e analizzare.

Capitolo 3

CONCETTI PRELIMINARI

Nel seguente capitolo vengono analizzate alcune nozioni di base necessarie per poter comprendere appieno l'ambito e il lavoro realizzato in questo progetto di tesi.
Di seguito verranno descritte le conoscenze tecniche relative ai concetti di modello di processo, Rete di Petri, event log, tracce, eventi, Local Instance Graph, embedding e alignment.

3.1 Modello di processo

Come già accennato nei capitoli precedenti di questo lavoro di tesi, i modelli di processo sono una rappresentazione delle attività svolte all'interno di un processo, sia per quanto riguarda i metodi di svolgimento, che la sincronizzazione, le dipendenze logiche e gli strumenti utilizzati nelle varie attività.
Quindi il loro scopo è quello di decidere quali attività devono essere eseguite e in quale ordine. Le attività possono essere svolte in maniera sequenziale o concorrente, possono essere opzionali e magari essere eseguite anche più volte all'interno di una stessa sequenza.

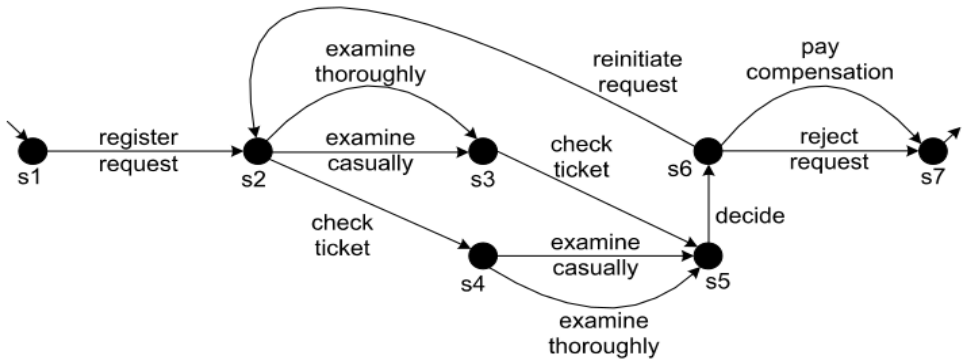


Figura 6: *Esempio di sistema a transizione di stati*

La notazione più semplice e basilare per rappresentare modelli di processo è quella di sistema a transizione di stati come riporta nell'esempio in figura 6.
Tuttavia questo tipo di rappresentazione per quanto semplice, ha problemi nell'esprimere la concorrenza in modo conciso, e soffrono il problema dell'esplosione dei dati.
Per questo solitamente vengono utilizzati altri tipi di notazioni, come ad esempio le Reti di Petri, la quale è la rappresentazione che è stata considerata in questo lavoro di tesi.

3.2 Rete di Petri

Una Rete di Petri è un grafo bi-partito, in cui sono presenti due tipi di nodi: i *places* che rappresentano gli stati, i quali a loro volta permettono di attivare l'altro tipo di nodi che sono le *transitions*, che rappresentano le attività che possono essere eseguite nel modello.
Inoltre sono presenti anche degli elementi indicati con dei pallini pieni all'interno dei places che vengono detti *token*: essi rappresentano un modo per tenere traccia del flusso di esecuzione all'interno del modello di processo. È possibile avere un numero qualsiasi di token all'interno di ogni place.

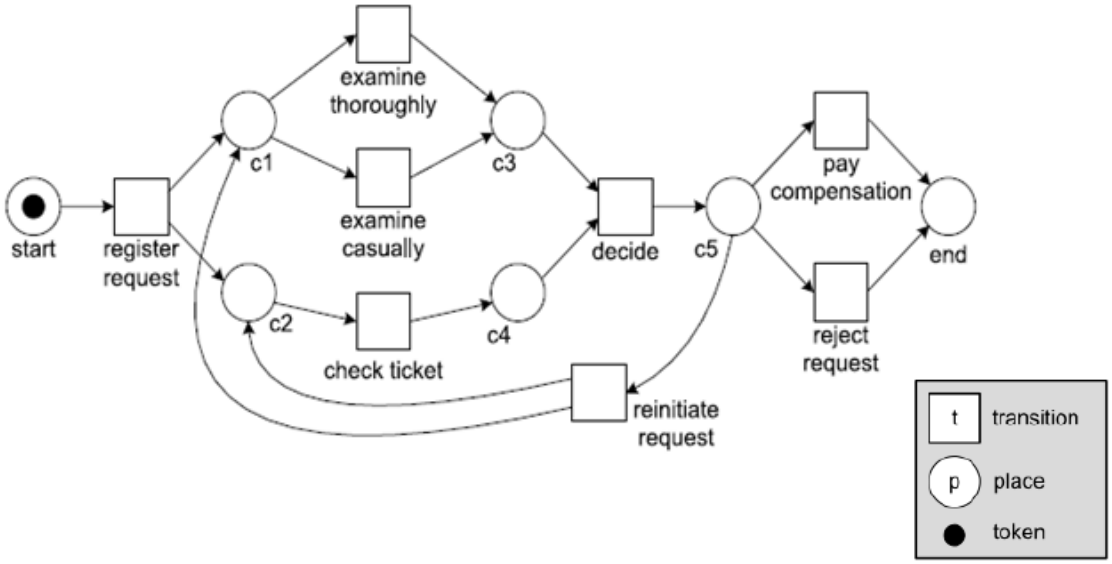


Figura 7: *Un esempio di rete di Petri*

Quindi formalmente, come descritto nell'articolo "*Model Repair Supported by Frequent Anomalous Local Instance Graphs*" [2], la rete di Petri può essere definita come una tupla di elementi *(P, T, F, A, l, mi, mf)*, in cui:

- *P* è un insieme di places;
- *T* è un insieme di transizioni;
- *F* è la relazione di flusso che collega places e transitions in maniera alternata;
- *A* è un insieme di etichette per le transizioni;
- *l: T → A* è una funzione che associa un'etichetta a ogni transizione in T;
- *mi* e *mf* rappresentano rispettivamente il nodo iniziale e finale della rete.

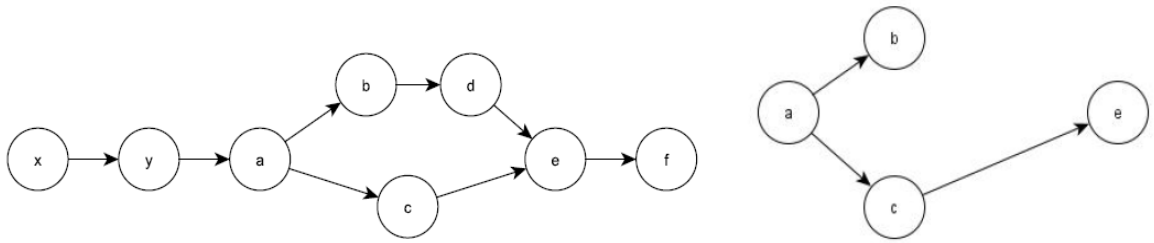
Una transizione è abilitata soltanto quando è presente un token in ogni place in ingresso a quella transizione.
Quando una transizione viene eseguita, consuma un token da ogni place in ingresso e genera in output un token per ogni place in uscita ad essa.
Viene definito *marking* lo stato di una rete di Petri. Esso è indicato da una certa distribuzione dei token nei place all'interno della rete ed è associato in modo univoco a un insieme di transizioni abilitate che potrebbero essere attivate in base alla posizione del token.

3.3 Event Log, Tracce, Eventi

Come accennato nei capitoli precedenti di questo lavoro di tesi, l'*event log* è un file sequenziale che viene utilizzato per registrare, in modo cronologico, tutte le operazioni che vengono eseguite. Per mezzo dei log, è possibile ricostruire tutte le attività che sono state attuate, conoscere chi e quando le ha effettuate.
Questo garantisce una grande facilità nell'analisi ed un continuo monitoraggio delle attività presenti nel processo, consentendo così di individuare eventuali punti di dispersione, sia di tempo che di risorse, all'interno del processo.
Gli event log sono il punto di partenza del Process Mining e sostanzialmente rappresentano un insieme di tracce.
Nell'articolo "*Building instance graphs for highly variable processes*" [5] viene definita una $traccia = \langle a_1, a_2, \dots, a_n \rangle \, \epsilon \, A$ (con A insieme di attività), come una sequenza delle attività svolte durante l'attuazione di un processo in base al loro ordine di esecuzione.
Quindi ogni singola traccia può essere vista come un'istanza di processo, cioè una singola esecuzione del processo stesso.
Le attività che vengono eseguite all'interno di una traccia sono dette *eventi*.
Ogni evento, oltre al nome dell'attività a cui si riferisce, possiede altri attributi, come le risorse coinvolte, chi ha svolto l'attività, il tipo di transizione, il timestamp, ecc.
Gli eventi sono registrati nei file di log sfruttano il formato standard XES (EXtensible Event Stream) adottato dalla IEEE Task Force on Process Mining e supportato da tutti gli strumenti che fanno uso delle tecniche di Process Mining.

3.4 Instance Graph e Local Instance Graph

Un problema che potrebbe verificarsi è che gli eventi nelle tracce vengono registrati in un determinato ordine, in base al timestamp, non permettendo di rilevare le situazioni di concorrenza tra le attività.
Per evitare tale problema, le tracce del registro possono essere rappresentate attraverso dei grafi detti *Instance Graph* (IG), cioè dei grafi diretti e aciclici attraverso i quali è possibile osservare il reale flusso di esecuzione delle attività di processo.
È bene notare, che gli IG possono rappresentare comportamenti sequenziali e simultanei, ma in ogni caso, non includono alcuni costrutti di flusso di processo, come i cicli e le scelte. Infatti un Instance Graph permette di rappresentare solo costrutti di split/join AND. Quindi, quando un IG ha rami paralleli, significa che nell'istanza di processo corrispondente, gli eventi dei rami sono stati eseguiti in parallelo.
Nel caso in cui un processo abbia costrutti split/join OR, nell'istanza del processo (e quindi nella traccia) verranno riportati solo i rami effettivamente eseguiti.
Dato un insieme di IG, è possibile derivare un insieme di Local Instance Graph (LIG), cioè dei sottografi che rappresentano porzioni di comportamenti di processo.



(a) (b)

Figura 8: *(a) Esempio di IG per una sequenza di eventi σ*
(b) Esempio di un possibile LIG contenuto nell'IG a sinistra

3.5 Embeddind e Extended Embedding

L'approccio utilizzato in questo progetto, si pone l'obiettivo di riparare un modello di processo introducendo comportamenti anomali rappresentati tramite dei LIG.
Per poter posizionare questi sottografi nel modello, è necessario prima rilevare la posizione nel processo nel quale questi si verificano.
Per fare questo è utile andare ad individuare uno o più *Embedding*, cioè delle sottotracce della traccia σ che si sta considerando, che contengono al loro interno tutti gli elementi del sottografo (LIG) individuato.
Inoltre è possibile definire l'*Extended Embedding* (indicato con σ_e), cioè la sottosequenza contigua di σ il cui primo e ultimo elemento sono uguali, rispettivamente, al primo e all'ultimo elemento della sottosequenza relativa al LIG.
Ad esempio, relativamente alla figura 8 del paragrafo precedente, considerando la sottosequenza anomala $s = \langle a, b, c, e \rangle$ (relativa all'immagine 8(b)) e la traccia
$\sigma = \langle x, y, a, b, c, d, e, f \rangle$ (con IG riportato nell'immagine 8(a)), l'extended embedding della traccia σ rispetto alla sottosequenza *s*, sarà $\sigma_e = \langle a, b, c, d, e \rangle$ (rappresentato in Figura 9) in quanto vengono considerati tutti gli elementi della traccia, compresi tra il primo e l'ultimo elemento della sottosequenza *s*.

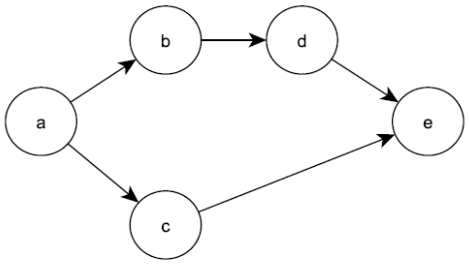


Figura 9: *Esempio di Extended Embedding ottenuto dall'IG della figura 8*

3.6 Alignment

Prendendo in considerazione ogni traccia, l'*alignment* è una lista di coppie di elementi, in cui per ognuna di esse, il primo elemento corrisponde a una transizione del modello di processo, mentre il secondo si riferisce a un evento del file di log.
Nella figura 10 sono riportate due tabelle di esempio, rappresentanti l'alignment tra la traccia ottenuta seguendo un percorso all'interno del modello di processo (riportata nella riga superiore delle tabelle), e la traccia presente nell'event log (indicata nella riga inferiore delle tabelle). È possibile notare che nella tabella (a) vi sia un perfetto allineamento tra le due tracce, in quanto in questo caso, tutti gli eventi di entrambe coincidono.
Invece, osservando la tabella (b), si può stabilire che si ha una traccia dell'event log che non si adatta al modello, in quanto sono presenti due eventi mancanti.
Quindi attraverso l'alignment è possibile notare che il modello ha eseguito la mossa *b* che non è prevista nella realtà, e al contrario, nella realtà è stata eseguita un'attività (la *c*) che non è prevista dal modello realizzato. Di conseguenza in questo caso si dovrà valutare se andare ad apportare delle modifiche al modello in modo che possa coincidere il più possibile con la realtà registrata.

a	b	d	e	g
a	b	d	e	g

(a)

a	b	»	d	e	g
a	»	c	d	e	g

(b)

Figura 10: *Esempi di Alignment*

In un alignment, vi sono tre possibili tipi di coppie o mosse:

- *Synchronous move*: coppia in cui entrambi gli elementi hanno la stessa etichetta, e corrisponde ad un'attività che viene svolta sia nel log che nel modello;
- *Move on log*: coppia in cui l'etichetta dell'attività è presente solo nel secondo elemento e rappresenta una transizione prevista dalla traccia del log ma non riproducibile sul modello;
- *Move on model*: coppia in cui l'etichetta della transizione è presente solo al primo elemento, e corrisponde ad un'attività che è prevista dal modello ma non è svolta nella realtà.

Data la presenza di diversi percorsi possibili all'interno del modello, possono esserci più allineamenti corrispondenti alla stessa traccia, e di conseguenza, sarà sempre scelto quello ottimale, cioè più conforme alla realtà. Per identificare tale allineamento, solitamente viene utilizzata una funzione di costo che dipende dal costo associato ad ogni mossa, e la scelta ottimale ricadrà in quella che ha il valore della funzione di costo minore.

Capitolo 4

IMPLEMENTAZIONE

4.1 Linguaggi e programmi utilizzati

L'algoritmo realizzato in questo progetto di tesi, è stato implementato interamente in linguaggio Python, tranne per due eseguibili realizzi in linguaggio C, di cui l'algoritmo fa uso nella prima parte, che è stata ripresa da un progetto precedente [2].
Python è un linguaggio di programmazione ad alto livello, orientato agli oggetti, dalla sintassi semplice e potente che ne facilita l'apprendimento. Supporta il paradigma object oriented, la programmazione strutturata e molte caratteristiche di programmazione funzionale.
Le caratteristiche più immediatamente riconoscibili di Python sono le variabili non tipizzate, l'uso dell'indentazione per la definizione dei blocchi e un garbage collector che si occupa automaticamente dell'allocazione e del rilascio della memoria.
Per quanto riguarda i due eseguibili utilizzati nella prima sezione dell'algoritmo sono il file *SGISO.exe*, che prende in input due file, uno contenente un grafo e uno contenente il LIG, e restituisce l'istanza del LIG nel grafo, se quest'ultima è presente.
Mentre il secondo eseguibile, *GM.exe*, confronta due grafi e restituisce in output il matching cost, ossia il numero di trasformazioni che devono essere svolte per rendere i due grafi isomorfi. Per trasformazioni sono da intendersi l'aggiunta o l'eliminazione di un arco o di un nodo, la modifica di un'etichetta di un arco o di un nodo e l'inversione della direzione di un arco.
Inoltre per l'implementazione dell'algoritmo si è fatto uso della libreria PM4PY [6].

***Figura 11**: Logo della libreria PM4PY*

PM4PY è una libreria Python open source creata dal Fraunhofer Institute for Applied Information Technology per supportare il Process Mining.
Questa libreria permette di gestire e manipolare gli event log, in quanto supporta l'input di dati tabulari come CSV, e soprattutto di dati in formato XES (EXtensible Event Stream):

formato più utilizzato per l'archiviazione di log, basato sul linguaggio XML prescritto come standard dall'IEEE.
Inoltre tale libreria è stata importante in questo lavoro di tesi, in quanto consente di applicare tecniche di Conformance Checking, fondamentali per l'algoritmo sviluppato, e perché permette di lavorare con strutture che possano rappresentare al meglio i modelli di processo, e nello specifico, in questo progetto sono state prese in considerazione le Reti di Petri, che come è già stato spiegato precedentemente.

4.2 Algoritmo di Process Repairing

L'approccio utilizzato ed ampiamente descritto nel paper "*Model Repair Supported by Frequent Anomalous Local Instance Graphs*" [2], in generale può essere visto come composto da 3 passaggi:

- **Estrazione e selezione di LIG anomali**: in questa prima fase, vengono estratti, dall'event log, una serie di LIG anomali che si verificano più frequentemente. Dall'insieme di questi LIG anomali, ne viene scelto uno secondo cui poi andare ad effettuare la riparazione del modello. Tale scelta può essere fatta o casualmente in base ad una decisione umana, o in base ad alcuni indicatori di performance dei vari LIG;

- **Selezione della traccia**: fase in cui si sceglierà una traccia tra tutte quelle presenti, in cui occorre il LIG selezionato nella fase precedente. Per fare questa scelta viene calcolato l'alignment delle tracce corrispondenti rispetto al modello da riparare. Una volta calcolato l'alignment, gli IG vengono ordinati sulla base del matching cost (numero di anomalie aggiuntive) del grafo tra l'IG della traccia e il LIG e verrà selezionato quello con costo minore;

- **Integrazione del LIG al modello**: ultima fase in cui avviene l'ampliamento del modello adattandolo al LIG selezionato. Questa è la fase che verrà analizzata nel dettaglio in questo lavoro di tesi, in quanto ci si è concentrati principalmente sulla realizzazione dell'algoritmo (riportato in figura 15) che permette di agganciare il LIG anomalo all'IG considerato e poter così riparare il modello secondo quel determinato comportamento anomalo.

Per analizzare e spiegare l'algoritmo realizzato in questa tesina, verrà utilizzato un esempio esplicativo, che rappresenta un caso su cui è stato applicato l'algoritmo in questione e di cui verranno riportati i vari risultati parziali e spiegati di conseguenza per espletare l'algoritmo. L'esempio che verrà illustrato di seguito, tiene in considerazione un event log contenente le varie tracce, rappresentato dal file "*testBank2000NoRandomNoise.xes*" e il modello di processo, riportato in figura 12, rappresentato sotto forma di rete di Petri e contenuto nel file "*testBank2000NoRandomNoise_petriNet.pnml*".

4.3 Estrazione di LIG anomali e selezione della traccia

In questo paragrafo verranno analizzate e spiegate senza entrare troppo nel dettaglio, le prime due fasi dell'algoritmo generale di repairing che è stato adottato. Tali fasi sono state implementate in un altro progetto precedente al lavoro descritto in questo documento, (riportato nell'articolo [2]), e di cui questa tesi ne è lo sviluppo, in quanto ci si è concentrati principalmente sull'implementazione dell'algoritmo costituente la terza ed ultima fase del processo di repairing attuato, ossia la fase di integrazione del LIG al modello.

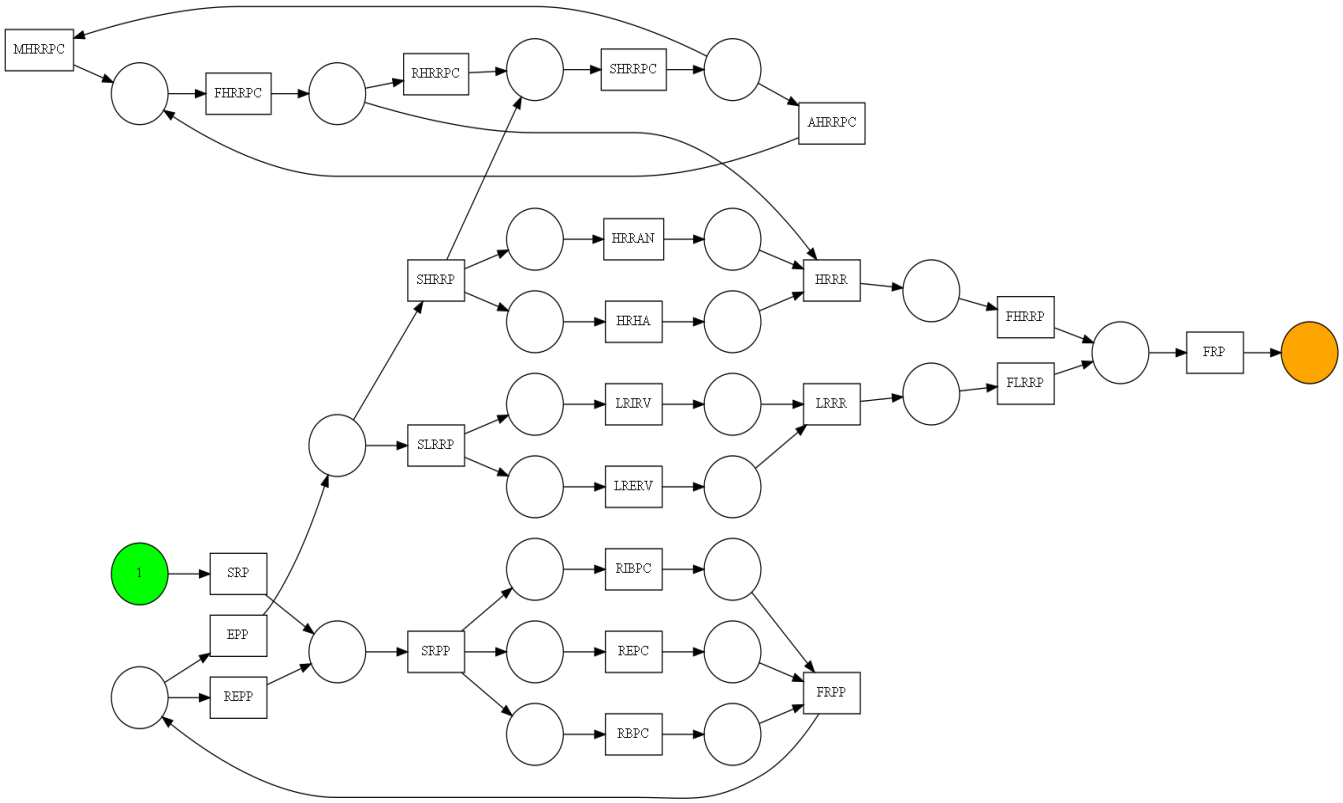


***Figura 12**: Modello di processo iniziale*

La tecnica di riparazione descritta prende come input comportamenti devianti di alto livello, descrivendo i LIG anomali che si verificano più frequentemente.
Dato un registro eventi che memorizza una serie di esecuzioni di processi, prima di tutto, ciascuna traccia sequenziale viene convertita in un Instance Graph (IG).
Una volta che si ha l'insieme degli IG, per prima cosa, si applica una tecnica di estrazione dei sottografi (LIG) anomali più frequenti e viene selezionato un LIG anomalo con cui riparare il modello.
Nel nostro caso è stato scelto il LIG riportato in figura 13(a), descritto utilizzando una particolare notazione, in cui i nodi vengono indicati con la lettera "v" davanti al numero e all'etichetta del nodo, mentre gli archi vengono indicati con la lettera "d" o "e".
Inoltre, tale LIG può anche essere visualizzato come un grafo diretto come riportato nella figura 13(b) per averne una rappresentazione più esplicita.

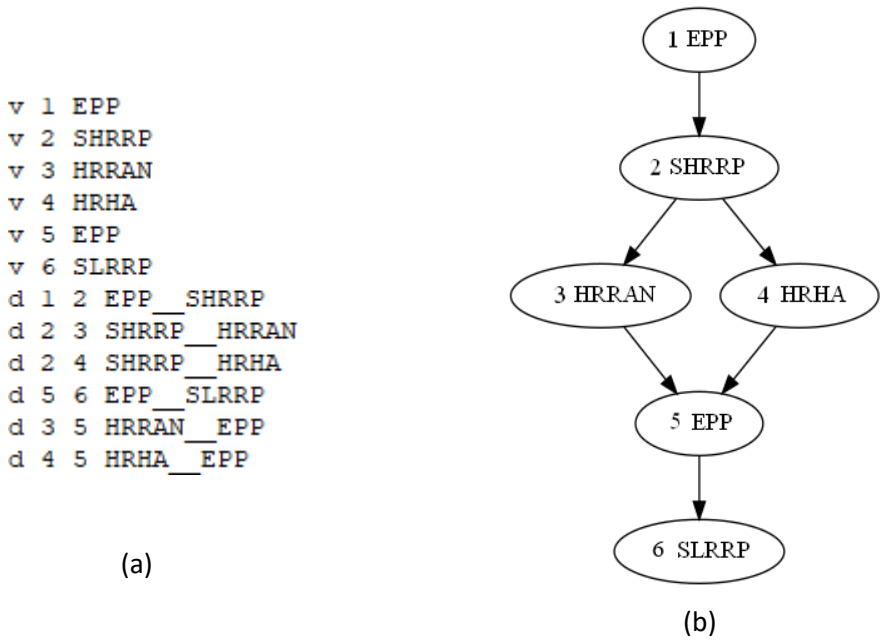


***Figura 13**: (a) Sottografo anomalo: "v" indica il nodo e "d" l'arco;*
(b) visualizzazione grafica del LIG anomalo

Successivamente, nella seconda fase, viene scelta la traccia candidata per la riparazione, selezionando gli IG contenenti il LIG scelto nella fase precedente.
Per fare questo, inizialmente si va a creare una lista di tutti gli IG in cui occorre il LIG considerato, e li si va ad ordinare in base al matching cost inteso come numero di trasformazioni necessarie per rendere i due grafi isomorfi.

Questo viene fatto tramite un tool apposito denominato "*GM.exe*", che prende in input due grafi (nel nostro caso il LIG selezionato e l'IG della traccia considerata), li confronta calcolandone l'alignment delle tracce corrispondenti rispetto al modello da riparare, e restituisce il costo di matching.
Così gli IG delle tracce vengono ordinati sulla base del matching cost tra l'IG della traccia e il LIG selezionato precedentemente, permettendo così di poter scegliere la traccia con il numero minimo di anomalie aggiuntive.

In seguito, dopo aver scelto la traccia, viene eseguito un altro tool denominato "*SGISO.exe*", che prendendo in input il LIG e l'IG della traccia scelta precedentemente, e restituisce i nodi dell'IG su cui sono mappati i nodi del grafo della sottotraccia anomala secondo cui fare il repairing.

Ad esempio, supponendo di aver selezionato la traccia in figura 3(a), applicando l'eseguibile "*SGISO.exe*", si ottiene l'istanza del IG della traccia con gli elementi che corrispondono ai nodi del LIG scelto (considerato precedentemente nella figura 2(a)) come riportato in figura 3(b), e che può essere rappresentato graficamente per una maggiore comprensione come in figura 3(c).

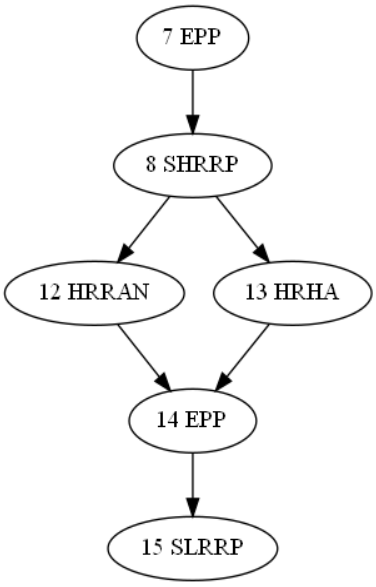
```
v 1 SRP
v 2 SRPP
v 3 RIBPC
v 4 RBPC
v 5 REPC
v 6 FRPP
v 7 EPP
v 8 SHRRP
v 9 SHRRPC
v 10 AHRRPC
v 11 FHRRPC
v 12 HRRAN
v 13 HRHA
v 14 EPP
v 15 SLRRP

e 1 2 SRP__SRPP
e 2 3 SRPP__RIBPC
e 2 4 SRPP__RBPC
e 2 5 SRPP__REPC
e 3 6 RIBPC__FRPP
e 4 6 RBPC__FRPP
e 5 6 REPC__FRPP
e 6 7 FRPP__EPP
e 7 8 EPP__SHRRP
e 8 9 SHRRP__SHRRPC
e 8 12 SHRRP__HRRAN
e 8 13 SHRRP__HRHA
e 9 10 SHRRPC__AHRRPC
e 10 11 AHRRPC__FHRRPC
e 14 15 EPP__SLRRP
e 11 14 FHRRPC__EPP
e 12 14 HRRAN__EPP
e 13 14 HRHA__EPP
```

(a)

```
v 7 EPP
v 8 SHRRP
v 12 HRRAN
v 13 HRHA
v 14 EPP
v 15 SLRRP
d 7 8 EPP__SHRRP
d 8 12 SHRRP__HRRAN
d 8 13 SHRRP__HRHA
d 14 15 EPP__SLRRP
d 12 14 HRRAN__EPP
d 13 14 HRHA__EPP
```

(b)



(c)

***Figura 14**: (a) traccia selezionata del file di log;*
(b) LIG con numero nodi corrispondenti alla traccia selezionata
(c) visualizzazione grafica del LIG

4.4 Integrazione del LIG nel modello

Algorithm 1: LIG integration

Input : The graph $\text{LIG}_{\sigma_e} = (E, W, l, \lambda)$, the alignment η of the trace σ over the process model $M = (P, T, F, A, \ell, m_i, m_f)$, the extended embedding $\sigma_e = \langle e_1^*, \ldots, e_n^* \rangle$ of LIG in σ

Output: Repaired model M'

1 $((E, W, l, \lambda), \sigma_e, start_{new}, end_{new}) = addStartEnd(\text{LIG}_{\sigma_e}, \sigma_e, null, null)$;
2 $(start_{LIG}, start_M, posStart_M, T_{MS}) = findLinks(\sigma_e, \eta)$;
3 $\sigma'_e = inv(\sigma_e)$; /* inv() reverses a sequence */
4 $\eta' = inv(\eta)$;
5 $(end_{LIG}, end_M, posEnd_M, T_{ME}) = findLinks(\sigma'_e, \eta')$;
6 $\sigma_M = \eta\,|_M$;
7 $P_S = token\_based\_replay\,(\sigma, \sigma_M, \eta, posStartM)$;
8 $P_E = token\_based\_replay\,(\sigma, \sigma_M, \eta, posEndM - 1)$;
9 $toRemove = \neg(\sigma_e[1] == start_{LIG})$; $E' = \{\}$;
10 **for** $i = 1; i < length(\sigma_e); i = i + 1$ **do**
11 **if** $toRemove$ **then**
12 $E' = E' \cup \{\sigma_e[i]\}$;
13 **if** $\sigma_e[i] == start_M$ **then**
14 $toRemove = False$;
15 **else if** $\sigma_e[i] == end_M$ **then**
16 $E' = E' \cup \{\sigma_e[i]\}$;
17 $toRemove = True$;
18 $E'' = E \setminus E'$;
19 $W'' = W \setminus \{(e_i, e_j) \in W | e_i \in E' \vee e_j \in E'\}$;
20 $((E, W, l, \lambda), \sigma_e, start_{LIG}, end_{LIG}) = addStartEnd((E'', W'', l, \lambda), \sigma_e \setminus E', start_{LIG}, end_{LIG})$;
21 $T_{LIG} = \emptyset$;
22 **forall** $e \in E'$ **do**
23 $T_{LIG} = T_{LIG} \cup \{create\_transition(e, M, \text{“}l\text{”})\}$;
24 $P_{LIG} = F_{LIG} = \{\}$;
25 **forall** $(e_i, e_j) \in W'$ **do**
26 $P_{LIG} = P_{LIG} \cup \{p_{ij}\}$;
27 $F_{LIG} = F_{LIG} \cup \{(find(e_i, T_{LIG}), p_{ij}), (p_{ij}, find(e_j, T_{LIG}))\}$;
28 $T' = T \cup T_{LIG}$; $P' = P \cup P_{LIG}$; $F' = F \cup F_{LIG}$;

***Figura 15**: Algoritmo per integrazione del LIG al modello*

Una volta individuati l'IG relativo alla traccia e il LIG all'interno della traccia stessa secondo cui fare repairing, è possibile passare alla fase di integrazione del LIG al modello, andando ad applicare l'algoritmo riportato in figura 15, che è stato effettivamente il lavoro principale di questa tesi e che andremo ad analizzare di seguito più nel dettaglio.

Innanzitutto, l'algoritmo considera in input principalmente 3 elementi: il LIG riportato sulla traccia che è stato individuato con l'eseguibile "*SGISO.exe*", l'alignment tra la traccia e il modello che si ha a disposizione, e l'extended embedding del LIG individuato sulla traccia.

4.4.1 Funzione *addStartEnd*

La prima operazione che viene svolta dall'algoritmo è quella di richiamare la funzione *addStartEnd* riportata di seguito in figura 16.

Function $(LIG',\sigma_e',start,end)$=*addStartEnd*$((E,W,l,\lambda),\sigma_e,start,end)$
- $Snode = \{e \in E | \nexists e' \in E, (e',e) \in W\}$; /* the set of start nodes */
- $Enode = \{e \in E | \nexists e' \in E, (e,e') \in W\}$; /* the set of end nodes */
- **if** $|Snode| > 1$ **then**
  - $E = E \cup \{H_S\}$;
  - $\lambda = \lambda \cup \{hidden\}$;
  - $l = l \cup \{(H_S, hidden\}$;
  - $\forall e_i \in Snode, W = W \cup \{(H_S, e_i)\}$;
  - $\sigma_e = append(\langle H_S \rangle, \sigma_e)$; /* append two sequences */
  - $start = H_S$;
- **if** $|Enode| > 1$ **then**
  - $E = E \cup \{H_E\}$;
  - $\lambda = \lambda \cup \{hidden\}$;
  - $l = l \cup \{(H_E, hidden\}$;
  - $\forall e_i \in Enode, W = W \cup \{(e_i, H_E)\}$;
  - $\sigma_e = append(\sigma_e, \langle H_E \rangle)$;
  - $end = H_E$;
- return (E,W,l,λ),σ_e,*start*,*end*;

***Figura 16**: Pseudocodice dell'algoritmo della funzione addStartEnd*

Questa funzione è utilizzata per andare ad individuare quelli che sono i nodi iniziali e finali del LIG considerato, e nel caso fossero presenti più nodi di inizio o di fine del grafo, vengono aggiunti dei nodi fittizi, uno di inizio e uno di fine, per fare in modo di avere soltanto un singolo nodo iniziale e finale del LIG e quindi semplificare la procedura di integrazione al modello.
Nello specifico, prima vengono individuati i nodi di inizio del grafo, come i nodi che non hanno archi entranti, e poi i nodi di fine, come i nodi che non hanno archi uscenti.
Nel caso in cui gli insiemi dei nodi iniziali e dei nodi finali, contenessero più di un elemento, verrebbero creati un nodo fittizio iniziale e/o finale, e collegati rispettivamente a tutti i nodi iniziali e a tutti i nodi finali del LIG.

Nell'esempio che è stato preso in considerazione, come è possibile vedere dal grafo del LIG rappresentato precedentemente in figura 14(c), il LIG ha già un solo nodo iniziale e un solo nodo finale, per questo verranno individuati tali nodi, e non verrà aggiunto alcuno nodo fittizio.

4.4.2 Funzione *findLinks*

Successivamente, l'algoritmo richiama una delle funzioni più importanti di questa procedura, ossia la funzione *findLinks* riportata in figura 17.

```
Function (link_LIG, link_M, posLink_M, T_M) = findLinks(σ_e, η)
    i = j = k = 1;
    done = True;
    T_M = {};
    while i < length(η) ∧ done do
        (t_i, v_i) = η[i];
        if t_i ==≫ then                                   /* move-on-log */
            if v_i == σ_e[j] then
                startNode = σ_e[j];
                done = False;
        else
            if v_i ≠≫ then                              /* synchronous move */
                posLastSync = k;
                lastSync = t_i;
                if v_i == σ_e[j] then
                    j = j + 1;
            else if j > 1 then                  /* move-on-model within σ_e */
                T_M = T_M ∪ {t_i};
            k = k + 1;                  /* synchronous move or move-on-model */
        i = i + 1;
    return startNode, lastSync, posLastSync, T_M;
```

***Figura 17**: Pseudocodice dell'algoritmo della funzione findLinks*

Questa funzione va a scansionare tutto l'alignment della traccia sul modello, per permette di determinare 4 elementi che saranno utili per andare ad individuare i punti di aggancio della LIG al modello per poterlo riparare.
Questi elementi che vengono restituiti come output sono:

- *startNode*: rappresenta la transizione corrispondente al primo move on log appartenente all'extended embedding del LIG sulla traccia. Questo elemento rappresenterebbe il primo evento che si verifica nella realtà ma non è possibile eseguire sul modello e per questo andrebbe aggiunto a quest'ultimo;
- *lastSync*: identifica l'ultima mossa sincrona eseguita prima del primo move on log individuato precedentemente;
- *posLastSync*: indica la posizione all'interno del modello dell'ultima mossa sincrona individuata prima della transizione memorizzata nella variabile *startNode*;
- *Tm*: insieme di transizioni move on model che si trovano tra l'inizio della traccia dell'extended embedding e il primo move on log identificato dalla variabile *startNode*.

Nell'esempio che stiamo considerando per illustrare l'algoritmo, vengono calcolati l'alignment tra modello e la traccia, e l'extended embedding del LIG sulla traccia stessa, ottenendo i risultati riportati di seguito:

***Alignment*:**

[('SRP', 'SRP'), ('SRPP', 'SRPP'), ('RIBPC', 'RIBPC'), ('RBPC', 'RBPC'), ('REPC', 'REPC'), ('FRPP', 'FRPP'), ('EPP', 'EPP'), ('SHRRP', 'SHRRP'), ('SHRRPC', 'SHRRPC'), ('AHRRPC', 'AHRRPC'), ('FHRRPC', 'FHRRPC'), ('HRRAN', 'HRRAN'), ('HRHA', 'HRHA'), ('HRRR', '>>'), ('>>', 'EPP'), ('FHRRP', '>>'), ('>>', 'SLRRP'), ('FRP', '>>')]

***Extended Embedding*:**

[['v', '7', 'EPP'], ['v', '8', 'SHRRP'], ['v', '9', 'SHRRPC'], ['v', '10', 'AHRRPC'], ['v', '11', 'FHRRPC'], ['v', '12', 'HRRAN'], ['v', '13', 'HRHA'], ['v', '14', 'EPP'], ['v', '15', 'SLRRP']]

Successivamente viene eseguita la funzione *findLinks* che restituisce i seguenti risultati:

startLIG: *['v', '14', 'EPP']*
startM: *HRHA*
posStartM: *13*
TmS: *['HRRR']*

Questi risultati rappresentano gli elementi illustrati precedentemente e restituiti dall'applicazione della funzione *findLinks* e, in particolare, indicano che il primo move on log dell'alignment che si trova all'interno dell'extended embedding è quello relativo al nodo con etichetta 'EPP'; di conseguenza l'ultima mossa sincrona presente nell'alignment prima di tale nodo, è la transizione 'HRHA' che si trova in posizione 13 all'interno del modello; inoltre, tra l'inizio della traccia dell'extended embedding e il primo move on log, è presente un move on model, corrispondente all'attività 'HRRR' che viene inserito nell'insieme *TmS*.

Una volta individuati tali elementi, l'alignment e l'extended embedding vengono invertiti, in modo che poi sia possibile richiamare la funzione *findLinks* scorrendo le sequenze dalla fine e determinare così gli elementi indicati in precedenza, utili in seguito per stabilire i punti di aggancio finali del LIG considerato.

In particolare, nel nostro esempio, richiamando la funzione *findLinks* con l'alignment e l'extended embedding invertiti si ottengono i seguenti risultati:

endLIG: *['v', '15', 'SLRRP']*
endM: 'Nd'
posEndM: *0*
TmE: *[]*

Questi risultati indicano che l'ultimo move on log dell'alignment che si trova all'interno dell'extended embedding, è rappresentato dal nodo con etichetta 'SLRRP'; la cosa particolare da notare, è che in questo caso, dopo il nodo individuato come *endLIG*, non vi sono ulteriori mosse sincrone, di conseguenza la variabile *endM* rimane non definita e viene assegnato valore 0 alla variabile posEndM; infine tra il nodo indicato dalla variabile *endLIG* e l'ultimo elemento dell'extended embedding non è presente nessun move on model e quindi l'insieme *TmE* rimane vuoto.

4.4.3 Places di aggancio

A questo punto, è possibile andare ad individuare, all'interno del nostro modello rappresentato come rete di Petri, i places di aggancio del LIG (*Ps* e *Pe* rispettivamente alla riga 7 e 8 dell'algoritmo in figura 15).
Questo viene fatto tramite la tecnica del *Token-Based Replay*. Tale funzione va a riprodurre l'esecuzione della traccia sulla rete di Petri del modello, restituendo diversi risultati che permettono di stabilire alcune caratteristiche della rete: infatti consente di determinare se la traccia sia conforme al modello; restituisce l'elenco delle transizioni attivate nel modello; indica il numero di token mancanti, consumati, rimanenti e prodotti nella rete; e infine stabilisce qual è il marking (stato in cui si trova la rete di Petri in un certo istante) raggiungo alla fine della procedura di replay applicata.
Proprio questa ultima informazione, ottenuta dall'attuazione della funzione di *Token-Based Replay*, risulta di notevole utilità, in quanto, il marking raggiunto in corrispondenza dell'ultima mossa sincrona prima del primo movimento anomalo identificato nella variabile *startLIG*, consente di individuare i places con i token nel momento precedente alla deviazione tra traccia e modello, mostrata dall'alignment. Tali places saranno i places di aggancio iniziali, a cui poter collegare l'inizio della sottotraccia anomala utilizzata per la riparazione del modello.
Successivamente viene applicata di nuovo la tecnica del *Token-Based Replay*, per stabilire il marking, e di conseguenza i places successivi all'ultima attività anomala tra traccia e modello, indentificata dalla variabile *endLIG.* Questi saranno i places di aggancio finali, a cui collegare la fine della sottotraccia anomala nella fase di riparazione del modello.

Nel nostro esempio, andando ad applicare la tecnica del *Token-Based Replay* (funzione built-in messa a disposizione dalla libreria PM4PY con cui si è realizzato questo progetto) si ottengono i seguenti risultati:

Place start: ['n10', 'n11, 'n12']
Place end: ['n15']

I risultati riportano gli identificatori dei place che vengono individuati tramite questa tecnica e, in particolare, essi sono tre places iniziali e un place finale a cui agganciare il LIG, evidenziati in blu nell'immagine 18.

In questo caso particolare, essendo la variabile *endM* non definita, non sarebbe possibile applicare la tecnica del *Token-Based Replay* per calcolare il place finale a cui agganciare il LIG.
Per questo si è stabilito che in questa circostanza si debba effettuare il collegamento dell'ultima attività direttamente al place di fine della rete. Quindi il place finale del modello, che nel nostro caso è il place 'n15', sarà anche il place di end a cui verrà collegato il LIG da inserire per la riparazione.

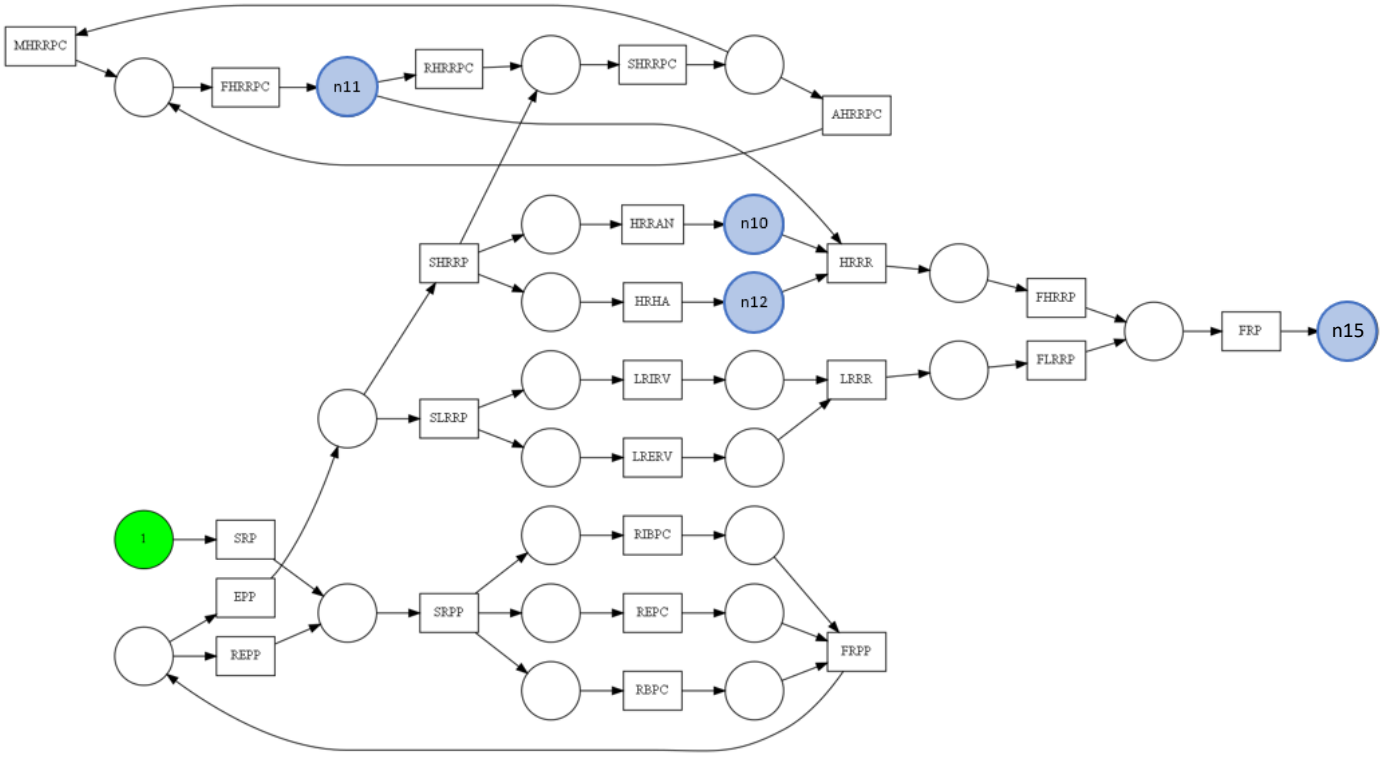


***Figura 18**: Modello di processo con place di aggancio evidenziati*

4.4.4 Semplificazione del LIG

La parte successiva dell'algoritmo, compresa tra le righe 9 e 19 dello pseudocodice in figura 15, si occupa di semplificare il più possibile il LIG, rimuovendo tutti i nodi corrispondenti a transizioni che non sono da aggiungere al modello, ossia quei nodi che rappresentano mosse sincrone e quindi già presenti nel modello.
Per fare questo si scorre l'extended embedding e si scartano dal LIG tutte le transizioni corrispondenti a mosse sincrone nell'alignment, fino ad arrivare all'ultima mossa sincrona (*startM*) prima del primo move on log individuato precedentemente (*startLIG*).
La stessa cosa viene fatta anche per le mosse sincrone che si trovano dopo l'ultimo move on log presente nell'extended embedding (*endLIG*). Questo perché tali mosse sincrone corrispondono a mosse che non comportano deviazioni in quanto, tali sequenze di attività vengono svolte anche all'interno del modello stesso.

Guardando in particolare al nostro esempio, è possibile vedere che la sequenza di transizioni 'EPP', 'SHRRP', 'SHRRPC', 'AHRRPC', 'FHRRPC', 'HRRAN', 'HRHA' appartenente all'extended embedding, può essere ritrovata anche all'interno del modello in quanto corrispondono tutte a mosse sincrone. Di conseguenza, le transizioni di tale sequenza che sono anche presenti all'interno del LIG che deve essere aggiunto, possono essere rimosse da esso, ottenendo così il LIG semplificato in figura 19.

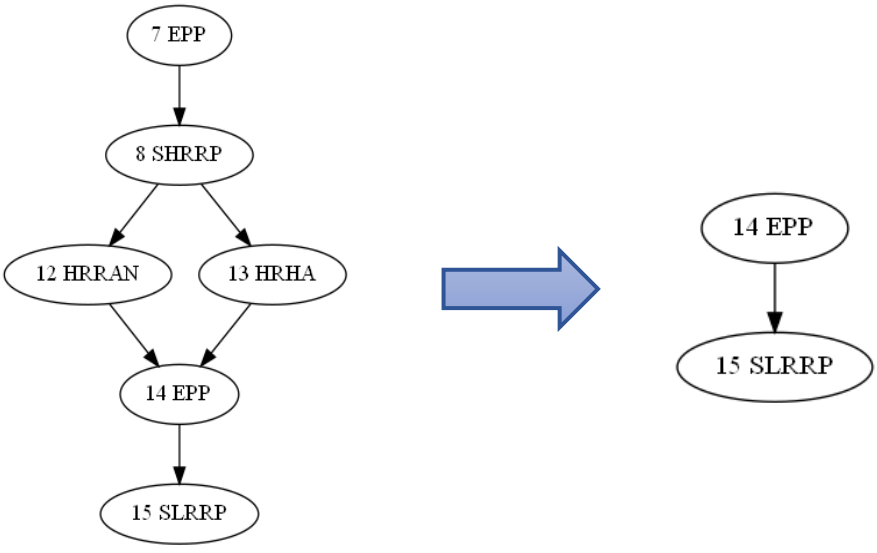


***Figura 19**: Semplificazione del LIG*

Inoltre nel caso in cui una volta semplificato, ci fossero nodi di inizio e fine multipli, viene riapplicata di nuovo la funzione *addStartEnd,* in modo da avere un solo nodo di inizio e di fine per semplificare le operazioni di aggancio del LIG al modello.
Nel nostro esempio, come è evidente dall'immagine precedente, anche nel grafo semplificato non si hanno nodi di inizio e di fine multipli e quindi non vi è la necessità di applicare la funzione *addStartEnd*.

4.4.5 Aggancio del LIG al modello

A questo punto, dopo aver semplificato il LIG da dover aggiungere al modello per poterlo riparare e aver individuato i places di aggancio nel modello, è possibile andare ad attuare l'operazione di inserimento del LIG al modello.
Per fare ciò, i nodi del grafo semplificato del LIG, vengono trasformati in transizioni da aggiungere all'insieme di transizioni della rete di Petri del modello, e collegate tra loro tramite dei place, anch'essi aggiunti all'insieme dei places del modello.
Quindi vengono inseriti degli archi che partono da ognuno dei place di start individuati precedentemente, fino alla prima transizione del LIG da inserire, e inoltre vengono aggiunti degli archi uscenti dall'ultimo nodo del LIG, diretti ai places di end determinati.

Nel nostro esempio, è possibile vedere il risultato dell'aggancio del LIG al modello in figura 20, in cui sono stati evidenziati in rosso gli elementi inseriti per la riparazione del modello: in particolare i due nodi del LIG semplificato in figura 19, sono stati trasformati in transizioni della rete di Petri e collegate tra loro tramite un place; inoltre sono stati aggiunti tre archi in ingresso e uno in uscita, per collegare la nuova sezione aggiunta, ai places di inizio e fine evidenziati in blu ed individuati nelle fasi precedenti.

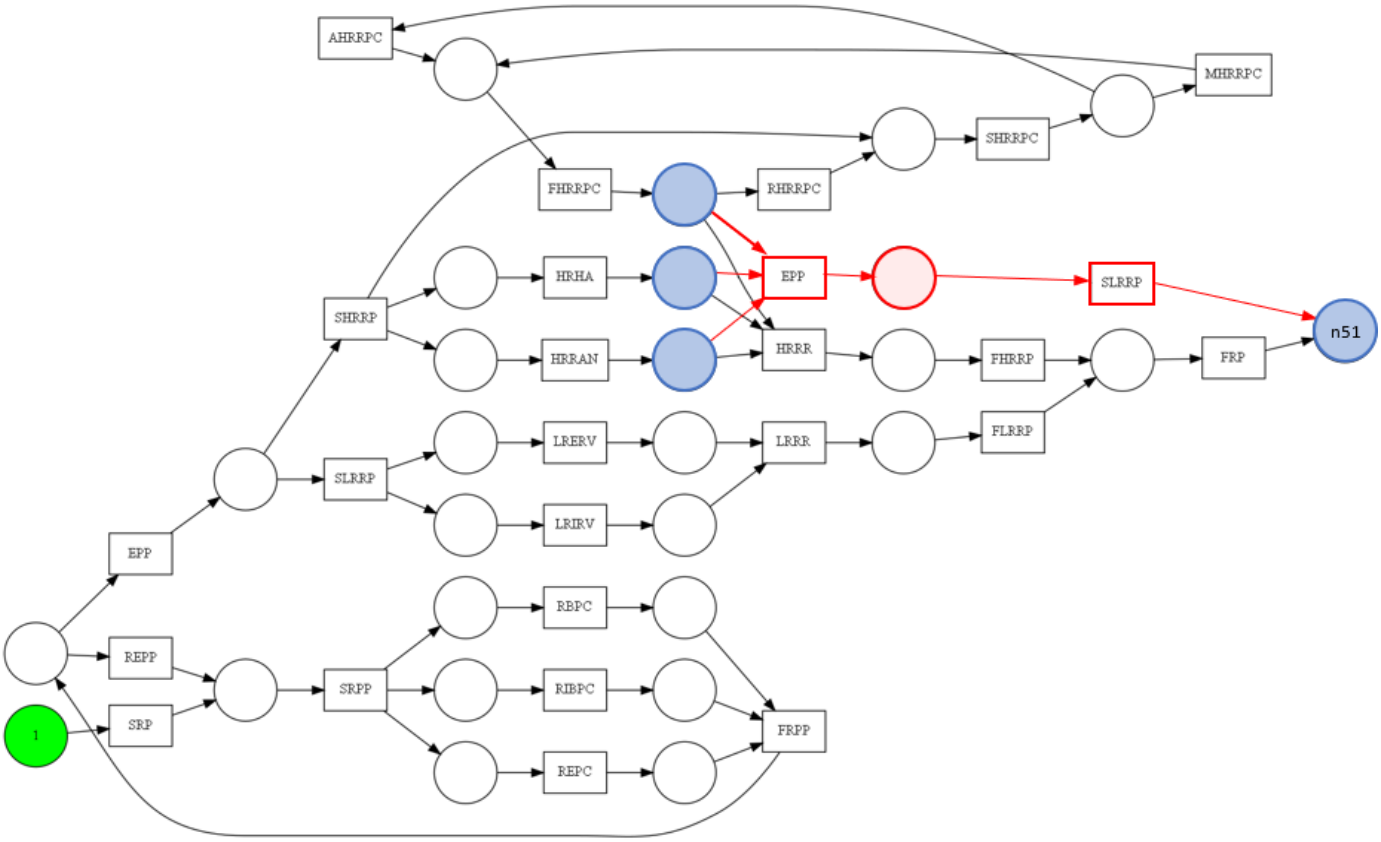


***Figura 20**: Modello di processo dopo la riparazione*

Capitolo 5

SPERIMENTAZIONE

5.1 Metriche

Per poter valutare la correttezza e la bontà dell'algoritmo realizzato, si è cercato di valutarlo sulla base di alcuni criteri di qualità solitamente utilizzati per la valutazione delle reti nell'ambito del Process Mining.
Per fare questo, sono state prese in considerazione alcune metriche importanti dei modelli di processo, che ne permettono una valutazione in termini di qualità:

- **Soundness**: una rete di flussi si dice soundness se rispetta 4 proprietà:
  - *Safeness*: i places della rete non possono contenere molteplici gettoni contemporaneamente;
  - *Corretta di compilazione*: se il place di uscita è segnato (cioè ha almeno un token al suo interno), tutti gli altri dovrebbero essere vuoti;
  - *Opzione di completamento*: deve esserci sempre la possibilità di raggiungere il marking che indica il place finale;
  - *Assenza di parti morte*: ossia tutte le transizioni del modello sono potenzialmente raggiungibili e attivabili.

  Il rispetto della proprietà di Soundness è un requisito fondamentale, perché fa sì che la rete possa essere considerata valida e solida. Per questo, nel caso la rete non fosse sound, significa che è presente qualche tipo di errore all'interno della rete e la riparazione non è stata fatta nella maniera corretta. Di conseguenza, in questo caso, le metriche seguenti non vengono neanche calcolate, altrimenti si potrebbero ottenere dei risultati indesiderati ed errati, ma dovrà essere analizzato nel dettaglio il caso considerato per determinarne eventuali problemi e poterli risolvere.
- **Fitness**: il modello di processo dovrebbe consentire la replicazione del comportamento osservato nel file di log. Quindi è una misura utilizzata per determinare quanto un dato modello supporta il comportamento rilevato nel log.
  Un modello ha fitness massima (pari a 1) se tutte le istanze nel log possono essere riprodotte sul modello dall'inizio alla fine;

- **Simplicity**: il modello di processo dovrebbe essere il più semplice possibile. Più il modello di processo contiene un numero elevato di place e transition, più esso sarà complesso e la misura di simplicity sarà bassa.
- **Precision**: questa metrica indica quanto il modello si avvicini all'event log e non consenta altri tipi di comportamenti oltre a quelli osservati nel log di eventi. Essa è una misura dell'*underfitting*: indica se il modello sovrageneralizza il comportamento usuale registrato nel log, cioè cattura anche comportamenti molto differenti da quelli effettivamente memorizzati dall'event log.
- **Generalization**: il modello dovrebbe generalizzare il comportamento esemplificativo osservato nel log. Al contrario della precision, essa è una misura dell'*overfitting*: indica che il modello generato è estremamente specifico, cioè è in linea con quel determinato file di log. L'overfitting dovrebbe essere ridotto il più possibile (generalization elevata) in modo che il modello non sia troppo specifico per quel particolare log, ma sia più generale anche per log registrati in un secondo momento che potrebbero avere comportamenti differenti.

Dalla definizione delle metriche prese in considerazione, è evidente che risulta complicato riuscire a bilanciare tali misure e ottenere dei valori ottimi per tutte.
Questo soprattutto per quanto riguarda le metriche di precision e generalization, perché se da un lato si cerca di rendere il modello più preciso possibile e far sì che non contenga comportamenti troppo differenti dall'event log considerato, dall'altro è anche vero che non è del tutto conveniente avere un modello che catturi esattamente tutti i comportamenti del log. Infatti un file di log, contiene solo i comportamenti che è stato possibile memorizzare fino ad un dato istante temporale, ma molte delle tracce ammissibili potrebbero ancora essere memorizzate in futuro.
Quindi bisognerebbe cercare allo stesso tempo di avere un modello con una buona generalizzazione, cosicché se si presentasse un'altra traccia con alcuni comportamenti diversi, possa comunque essere riprodotta dal modello.

5.2 Testing

Nello svolgimento di questo lavoro di tesi, è stato preso in considerazione il log di eventi 'testBank2000NoRandomNoise' contenuto nel file "*testBank2000NoRandomNoise.xes*".
L'insieme di sotto-grafi anomali frequenti ricevuti in input, è disponibile in un semplice database, costituito dalle seguenti relazioni:

- *'traceid':* composta dalle colonne "numTrace", che indica il numero del grafo e "idTrace", che rappresenta l'identificativo della traccia, utile per individuare per ogni traccia il numero del grafo corrispondente;
- *'submeasures':* i cui record delle colonne "subcontent" e "subelements" contengono i LIG espressi in insieme di nodi e archi.

Per verificare l'efficacia dell'algoritmo realizzato, abbiamo riparato il modello utilizzando un LIG alla volta, e una volta ottenuto il modello riparato per ognuno di essi, è stata verificata prima la proprietà di Soundness.
Nel caso tale proprietà assumesse valore positivo (True), vengono calcolati i valori delle quattro metriche indicate nel paragrafo precedente per valutare la qualità del modello ottenuto.

Lo svolgimento dei test, è stato attuato aggiungendo un'ulteriore parte di codice, all'algoritmo precedentemente descritto (riportata in figura 21), precisamente tra l'applicazione della funzione *addStartEnd* e quella della funzione *findLinks*.
Tale sezione di codice permette di selezionare una parte dell'extended embedding e considerarlo come l'extended embedding vero e proprio estratto dalla traccia in relazione al LIG che è stato scelto. Una volta riparato il modello con il segmento di extended embedding estratto, viene fatta scorrere tale sezione sull'extended embedding completo, andandone così a considerare un'altra parte per la riparazione.
Questo permette di effettuare più test in maniera più rapida senza dover ripetere le prime due fasi dell'algoritmo generale (estrazione del LIG e selezione della traccia), ma come se in realtà fossero eseguite, poiché si considera ogni nuovo extended embedding come se fosse estratto da una nuova traccia in relazione ad un nuovo LIG scelto.

Algorithm 2: Testing

```
1  σ_e = ⟨e*_1, ..., e*_n⟩                          // Extended embedding del LIG in σ
2  k = 0,      j = 0,      inc = 0
3  while True do
4      a = parametri.a
5      b = int(round(len(σ_e) * (parametri.b)/100    // Scelgo dei valori
6      k = random({x ∈ ℕ\{0} | a ≤ x ≤ b})
7      j = random ({x ∈ ℕ | x < lenght(σ_e) − k})
8
9      σ*_e = ⟨e*_j, e*_{j+1}, ... , e*_{j+k}⟩          // Estraggo una parte dell'extended embedding
10     inc += 1
11     while True do
12         count = 0
13         forall e ∈ σ*_e do
14             if e is move on log then
15                 count + 1                          // Verifico se all'interno del nuovo embedding
16         if count = 0 and (j + k + inc) ≤ len(σ_e) then   // sia presente almeno un evento anomalo
17             σ*_e = ⟨e*_{j+1}, e*_{j+2}, ... , e*_{j+k+1}⟩   // Se tale evento non è presente, faccio scorrere la
18             inc += 1
19         else                                       // finestra di eventi, in modo che vi sia presente almeno
20             break                                  // un evento anomalo nel nuovo extended embedding
21     if count = 0 and (j + k + inc) > len(σ_e) then
22         break
23
```

***Figura 21**: Pseudocodice sezione aggiuntiva per il testing*

Analizzando il codice più nello specifico, si può osservare che inizialmente vengono stabiliti a priori due parametri (*a* e *b*) e inseriti in un altro file (denominato *parametri.py*). Tali parametri verranno utilizzati per il calcolo delle altre due variabili *k* e *j*.

Queste variabili vengono scelte in maniera randomica a partire dagli intervalli ottenuti dai valori *a* e *b* precedentemente stabiliti, e rappresentano rispettivamente la lunghezza del segmento di extended embedding che verrà considerato e l'indice dell'elemento dell'extended embedding da cui iniziare ad estrarre il segmento.

A questo punto, si verifica che all'interno della sezione di extended embedding considerata, vi sia almeno un elemento che corrisponda ad una attività anomala non presente nel modello di processo e con cui andare a realizzare la riparazione.

Se non è presente nessun evento anomalo all'interno del segmento di extended embedding estratto, allora si farà scorrere in avanti di una posizione la sezione di lunghezza *k,* fino a che non ci sia, al suo interno, almeno un elemento coincidente con un'attività anomala dell'event log.

Una volta effettuata la riparazione del modello, il segmento precedentemente individuato viene fatto scorrere nuovamente controllando sempre che al suo interno vi sia almeno un evento anomalo. In questo modo è possibile realizzare una nuova riparazione del modello come se si fosse preso in considerazione un nuovo LIG e di conseguenza un nuovo extended embedding relativo da un'altra traccia.

Questo processo viene poi ripetuto fino a che non si arriva al termine dell'extended embedding e non è più possibile scorrerlo ulteriormente.

5.3 Analisi risultati

In questo paragrafo verranno analizzati i risultati ottenuti dall'applicazione dell'algoritmo realizzato in questo lavoro di tesi.
Tali risultati sono stati memorizzati all'interno delle due tabelle riportate di seguito.
Nella Tabella 1, sono indicati i valori delle metriche calcolate sul modello originale, in modo da poter essere presi come riferimento, e confrontarli poi con i valori ottenuti dal modello riparato.
Come è possibile osservare, le prime quattro colonne rappresentano i valori delle quattro metriche principali usate per valutare la qualità di un modello di processo (di cui si è già discusso nel primo paragrafo di questo capitolo), mentre le ultime tre colonne sono rispettivamente il numero di places, il numero di transizioni e il numero di archi presenti nella rete di Petri che rappresenta il modello di processo iniziale, e che sono dei valori a supporto della metrica di Simplicity e che permettono di dare una misura sulla complessità del modello stesso.

Log Fitness Initial Model	Precision Initial Model	Generalization Initial Model	Simplicity Initial Model	Num Places	Num Transitions	Num Arcs
0.8511988654	0.93648397585	0.9676365958	0.7575757576	26	24	58

***Tabella 1**: Misure del modello di processo iniziale prima della riparazione*

Invece nella Tabella 2, sono indicate le misure calcolate sul modello di processo riparato.
In particolare, la prima colonna riporta l'identificativo del LIG utilizzato per la riparazione del modello; la seconda esprime il numero di tracce in cui è presente il LIG considerato e ci da una misura di quanto la riparazione secondo quel LIG anomalo incida sulla riparazione del modello per l'intero log.
Nelle successive due colonne, vengono riportati il segmento di extended embedding considerato dall'algoritmo di repairing, come spiegato nel paragrafo precedente, e la sua relativa lunghezza, cioè il numero di attività che lo costituiscono.
Nelle tre colonne seguenti sono indicati altri valori utili per le successive analisi, che sono il la lunghezza della traccia considerata e il numero di asincronie (cioè eventi anomali) presenti sia nell'extended embedding considerato, che nella traccia stessa.
Nella colonna "Soundness" viene espresso il valore ottenuto dalla verifica della proprietà di Soundness, potendo così vedere se sia rispettata tale proprietà dal modello; essa è una delle informazioni più importanti all'interno della tabella, in quanto se assume valore "True" significa che la proprietà è soddisfatta, di conseguenza la riparazione è stata effettuata in maniera corretta e il modello può essere considerato valido e solido.

ID LIG	Numero tracce in cui occorre il LIG	Embedding	Lung. Embedding	Lung. Traccia	Asincronie Embedding	Asincronie Traccia	Soundness	Trace Fitness Initial Model	Trace Fitness Repaired Model	Log Fitness Repaired Model	Generalization	Simplicity	Num Places	Num Transitions	Num Arcs
8	187	[['v', '11', 'FHRRPC'], ['v', '12', 'HRRAN'], ['v', '13', 'HRHA'], ['v', '14', 'EPP']]	4	15	1	2	True	0,821429	0,964286	0,872554	0,966406	0,69863	26	25	62
8	187	[['v', '12', 'HRRAN'], ['v', '13', 'HRHA'], ['v', '14', 'EPP'], ['v', '15', 'SLRRP']]	4	15	2	2	True	0,821429	1	0,877893	0,965034	0,706667	27	26	64
4	278	[['v', '23', 'LRIRV'], ['v', '24', 'FLRRP'], ['v', '25', 'LRRR']]	3	26	1	4	True	0,820513	0,846154	0,858674	0,965466	0,73913	26	25	60
4	278	[['v', '24', 'FLRRP'], ['v', '25', 'LRRR'], ['v', '26', 'FRP']]	3	26	1	4	True	0,820513	0,846154	0,858674	0,966315	0,73913	26	25	60
63	345	[['v', '14', 'REPP'], ['v', '15', 'FRPP']]	2	26	1	4	True	0,820513	0,871795	0,890097	0,96747	0,69863	26	25	62
105	604	[['v', '1', 'SRP'], ['v', '2', 'FRPP']]	2	16	1	2	True	0,862069	0,931034	0,890097	0,967374	0,69863	26	25	62
43	25	[['v', '10', 'REPC'], ['v', '11', 'RBPC'], ['v', '12', 'SRPP']]	3	19	1	4	True	0,75	0,875	0,890097	0,928931	0,69863	26	25	62
44	49	[['v', '7', 'REPP'], ['v', '8', 'REPP'], ['v', '9', 'FRPP'], ['v', '10', 'REPC'], ['v', '11', 'RIBPC']]	5	32	2	5	True	0,822222	0,888889	0,874483	0,965223	0,706667	27	26	64
44	49	[['v', '8', 'REPP'], ['v', '9', 'FRPP'], ['v', '10', 'REPC'], ['v', '11', 'RIBPC'], ['v', '12', 'RBPC']]	5	32	2	5	True	0,822222	0,888889	0,874483	0,965749	0,706667	27	26	64
44	49	[['v', '9', 'FRPP'], ['v', '10', 'REPC'], ['v', '11', 'RIBPC'], ['v', '12', 'RBPC'], ['v', '13', 'SRPP']]	5	32	2	5	True	0,822222	0,911111	0,928994	0,966739	0,73494	32	29	72

***Tabella 2**: Porzione della tabella con le misure del modello di processo riparato con il nostro algoritmo di repairing*

Le due colonne successive, rappresentano i valori della metrica di Fitness calcolata prima e dopo la riparazione del modello, relativamente alla traccia presa in considerazione evidenziando il fatto di quanto la traccia in questione si adatti al modello iniziale e al modello riparato.
Infine le ultime colonne della Tabella 2, riportano i valori delle metriche relative al modello riparato e che possono quindi essere confrontate con quelle della Tabella 1 per poter fare una valutazione immediata della qualità della riparazione attuata.

Da una prima analisi, come riportato nella tabella 3, è possibile osservare che, dopo la riparazione per ogni LIG, il valore medio della Fitness migliora in tutti i casi, passando ad un valore medio di circa 0,93 (cioè il 93%) sia per il caso con 1 asincronia nell'extended embedding, sia nel caso con 2, con un miglioramento anche più accentuato in questo secondo caso.
Questo sta a significare che sono resi possibili alcuni percorsi in più che non erano permessi nel modello originale, e quindi ora il modello è in grado di replicare ulteriori comportamenti presenti nell'event log.

	Asincronie Embedding	Valori		
	1		2	
Lung. Embedding	Media di Trace Fitness Initial Model	Media di Trace Fitness Repaired Model	Media di Trace Fitness Initial Model	Media di Trace Fitness Repaired Model
2	0,841290893	0,901414677		
3	0,856244065	0,906517094		
4	0,882518797	0,95018797	0,821428571	1
5	0,893390313	0,931794872	0,841666667	0,922222222
6	0,897435897	0,948717949		
7	0,846153846	0,923076923	0,800824176	0,90297619
8	0,923076923	0,961538462		
9	0,821428571	0,964285714	0,821428571	1
Totale comple	**0,874135064**	**0,932552541**	**0,821282051**	**0,930079365**

***Tabella 3**: Valori di Fitness relative alla traccia considerata ordinate in base alla lunghezza dell'extended embedding*

Considerando la Fitness relativa all'intero file di log, dalla tabella 4 è possibile vedere ugualmente un incremento complessivo di tale metrica nonostante sia contenuto (con un incremento medio di circa il 2,7%), dato che l'adattamento del modello alla traccia viene misurato sull'intero log e non solo sulle singole tracce come calcolato nella tabella precedente.

Numero tracce in cui occore LIG	Media di Log_Fitness Repaired Model	Diff Log Fitness
25	0,890096625	0,038897759
49	0,892653664	0,041454798
79	0,862856131	0,011657265
88	0,85867416	0,007475295
90	0,8752239	0,024025035
106	0,854782758	0,003583893
118	0,897876176	0,046677311
121	0,890096625	0,038897759
149	0,890096625	0,038897759
179	0,855239379	0,004040514
187	0,8752239	0,024025035
203	0,864402549	0,013203684
220	0,857090582	0,005891717
278	0,85867416	0,007475295
345	0,890096625	0,038897759
604	0,890096625	0,038897759
846	0,890096625	0,038897759
Totale complessivo	**0,878144729**	**0,026945864**

***Tabella 4**: Valori di Fitness del modello riparato calcolate sull'intero log, e relative differenze con il valore del modello originale.*

Invece, osservando la tabella 5, si può notare che la Generalization, in media, ha una diminuzione, anche se non molto significativa, dovuta probabilmente alla lunghezza troppo breve degli extended embedding considerati e del numero molto basso di asincronie riparate ogni volta (1 o 2 per ogni extended embedding). Una cosa possibile da notare è che andando a riparare più attività all'interno di una traccia, come ci si poteva aspettare, il modello generalizza di più.
Solitamente sarebbe meglio avere un valore di generalization più alto possibile, ma allo stesso tempo bisognerebbe che tale valore non sia troppo elevato, altrimenti il modello rischia di generalizzare troppo e quindi considerare anche troppi eventi che non sono strettamente correlati con gli eventi del log, o che sono poco frequenti all'interno di esso.

Per quanto riguarda la Simplicity si nota che ha un lieve peggioramento, e questo era un risultato che ci si aspettava, in quanto andando ad effettuare il repairing, si va a complicare il modello stesso. Infatti questo risultato è anche supportato dal leggero aumento dei valori corrispondenti al numero di places, transitions e archi presenti nel modello.

Invece per quanto riguarda la misura di Precision (valori non riportati nelle tabelle), si ottiene un peggioramento dei suoi valori, nonostante ci si sarebbe aspettati un miglioramento, ma si è deciso di non prenderla in considerazione per questo lavoro di tesi, poiché, dalla documentazione della libreria PM4PY [6], si evince che il metodo utilizzato per il calcolo

del valore di questa metrica, tiene in considerazione soltanto le parti di traccia che possono essere replicate sul modello, scartando quelle che presentano delle anomalie.
Quindi una volta riparato il modello, esso riuscirà a considerare più prefissi di traccia poiché dopo la riparazione ci saranno più parti di traccia replicabili dal modello. Ma di conseguenza cambia anche il valore con cui viene calcolata la media della misura di Precision dell'intero log. Per questo i valori che si ottengono non possono essere considerati veritieri, e quindi non confrontabili per poter estrarre delle informazioni utili da essi.

Lung. Embedding	Media di Generalization	Media di Simplicity
2	0,967421771	0,698630137
3	0,953828437	0,712130236
4	0,9669475	0,699778213
5	0,966201843	0,714343449
6	0,948179741	0,698630137
7	0,951775401	0,710797923
8	0,928931132	0,739130435
9	0,965719967	0,702648402
Totale complessivo	**0,95796219**	**0,709051046**

***Tabella 5**: Valori di Generalization e Simplicity del modello riparato calcolate sull'intero log.*

Di seguito verranno riportati degli esempi esplicativi per poter interpretare e visualizzare meglio i dati ottenuti nelle tabelle precedenti.
Il primo esempio considerato, è quello relativo alla riparazione del modello tramite il LIG anomalo numero 21, con cui dopo l'applicazione dell'algoritmo di repairing, si ottengono i risultati riportati nella tabella 6 e il relativo modello di processo rappresentato nella figura 22 nella pagina successiva.

ID LIG	Numero tracce in cui occorre il LIG	Embedding	Lung. Embedding	Trace_Fitness Initial Model	Trace_Fitness Repaired Model	Log_Fitness Repaired Model	Generalization	Simplicity	Num Places	Num Transitions	Num Arcs
21	88	[['v', '5', 'REPC'], ['v', '6', 'FRPP'], ['v', '7', 'EPP'], ['v', '8', 'SLRRP'], ['v', '9', 'LRIRV'], ['v', '10', 'LRERV'], ['v', '11', 'FLRRP'], ['v', '12', 'LRRR']]	8	0.9230769231	0.9615384615	0.8586741602	0.9289311320	0.7391304348	26	25	60

***Tabella 6**: Valori del modello riparato tramite il LIG numero 21.*

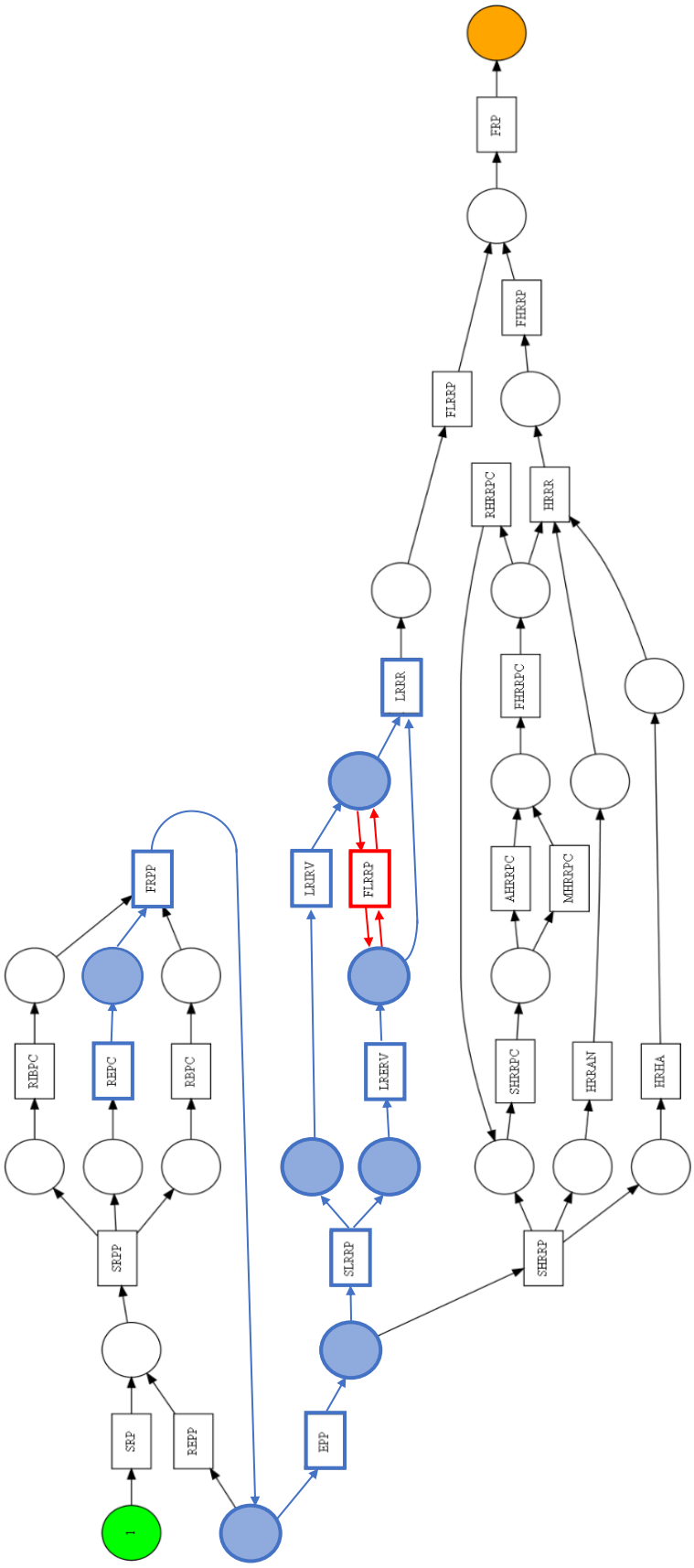


***Figura 22**: Modello di processo riparato tramite il LIG numero 21.*

Nella figura 22 è stato evidenziato in blu l'extended embedding considerato, e in rosso l'attività anomala che è stata aggiunta al modello originale per effettuare la riparazione.
Confrontando la figura 12 che illustra il modello di processo originale, con la figura 22 appena mostrata e la tabella 6 dei relativi risultati, è possibile notare che la Fitness del log aumenta pochissimo rimanendo praticamente quasi identica alla Fitness del modello originale, in quanto il modello viene riparato soltanto tramite un'unica asincronia (l'attività evidenziata in rosso nella figura 22) e inoltre il LIG considerato si trova in un numero molto esiguo di tracce dell'intero log.
Per quanto riguarda la Generalization, dalla tabella 6 si vede che ha un notevole calo, in quanto, nonostante l'extended embedding considerato sia abbastanza grande, esso contiene una sola asincronia che allo stesso tempo è l'unica asincronia presente nell'intera traccia. Quindi il modello generalizzerà molto meno in quanto sarà molto più preciso riguardo il comportamento della traccia considerata.
Infine, osservando i valori di Simplicity, si evince che tale metrica peggiora rispetto al valore calcolato sul modello originale, ma non in maniera eccessiva, in quanto il modello riparato non viene reso troppo complesso dato che viene inserita soltanto un'ulteriore transizione e 4 archi per collegarla al resto della rete (elementi evidenziati in rosso nella figura 22).

Il secondo esempio riportato, riguarda la riparazione del modello originale tramite il LIG numero 17. I dati mostrati nella tabella 7 rappresentano i valori ottenuti dall'applicazione dell'algoritmo di repairing, e la figura 23 raffigura il relativo modello riparato restituito.

ID LIG	Numero tracce in cui occorre il LIG	Embedding	Lung. Embedding	Lung. Traccia	Asincronie Embedding	Asincronie Traccia	Trace_Fitness Initial Model	Trace_Fitness Repaired Model	Log_Fitness Repaired Model	Generalization	Simplicity	Num Places	Num Transitions	Num Arcs
17	203	[['v', '7', 'EPP'], ['v', '8', 'SRP'], ['v', '9', 'FRP'], ['v', '10', 'SHRRPC']]	4	17	2	2	0.9	1.0	0.8670432857	0.9653711177	0.7066666666	27	26	64

***Tabella 7**: Valori del modello riparato tramite il LIG numero 17.*

In questo esempio è possibile notare che nonostante anche in questo caso il modello venga riparato utilizzando un numero molto basso di anomalie, esse sono proprio tutte le anomalie presenti all'interno della traccia considerata, e di conseguenza la Fitness relativa alla traccia aumenta raggiungendo il valore massimo.
Osservando la Generalization, si può constatare che rimane sostanzialmente invariata per il fatto che si ha un extended embedding molto ridotto (formato soltanto da 4 attività, quelle evidenziate nella figura 23), e dato che nonostante la traccia considerata sia abbastanza lunga, il modello viene riparato sulla base di solo 2 attività.
Infine, per quanto riguarda la Simplicity, essa si riduce di più che nel caso precedente, visto l'incremento maggiore del numero di transizioni e di archi aggiunte all'interno della rete.

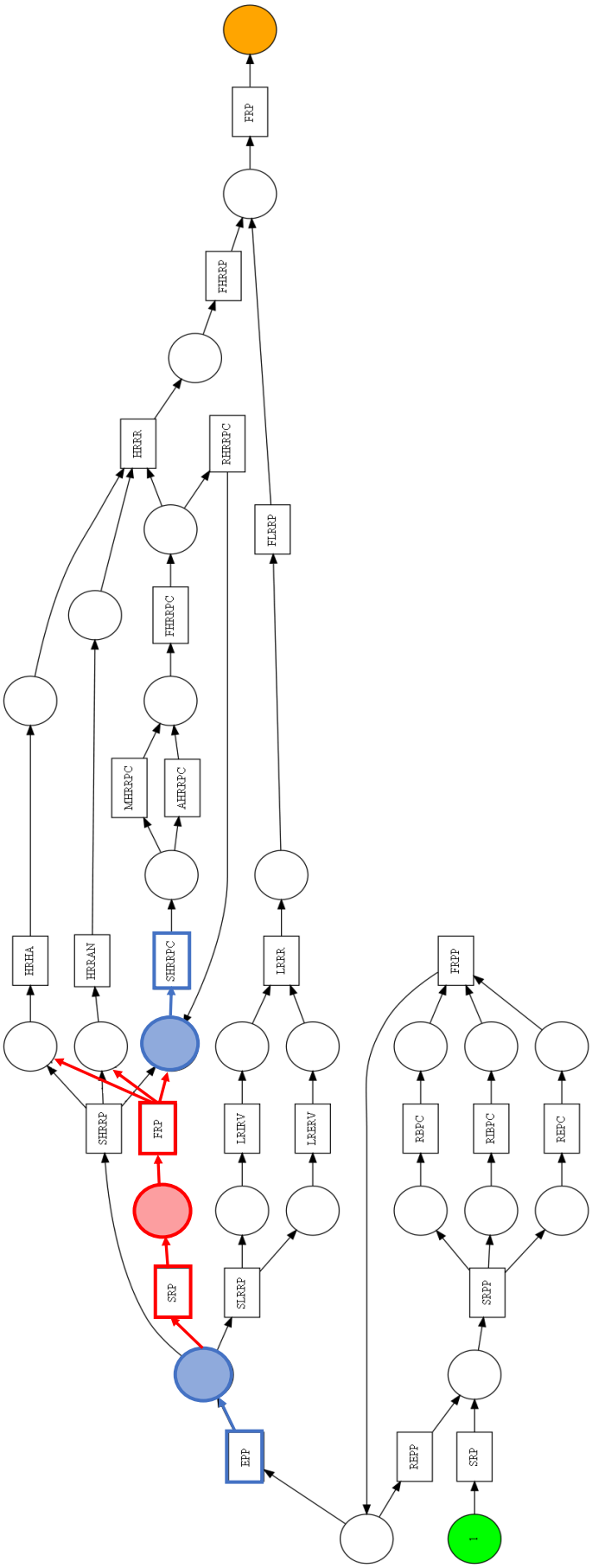


***Figura 23**: Modello di processo riparato tramite il LIG numero 17.*

5.4 Implementazione aggiuntiva

Oltre all'algoritmo di repairing descritto finora in questo lavoro di tesi, si è cercato anche di implementare una parte ulteriore dell'algoritmo, riportata in figura 24, da inserire alla fine del codice descritto nei capitoli precedenti e che sarebbe stata utile per effettuare una riparazione aggiuntiva del modello di processo.

```
29 forall t ∈ T_MS ∪ T_ME do
30 |  h' = create_transition(M);
31 |  h'' = create_transition(M);
32 |  T' = T' ∪ {h', h''}; P' = P' ∪ {p', p'', p'''};
33 |  F' = F' ∪ {(p, h')|(p, t) ∈ F} ∪ {(h', p)|(t, p) ∈ F};
34 |  if t ∈ T_MS then
35 |  |  F' = F' ∪ {(t, p''), (p'', h''), (h'', p''')} ∪ {(h'', p)|(t, p) ∈ F};
36 |  |  F' = F' \ {(t, p)|(t, p) ∈ F};
37 |  |  F' = F' ∪ {(h', p'), (p', start_LIG), (start_LIG, p'''), (p''', end_M)};
38 |  else
39 |  |  F' = F' ∪ {(p'', t), (h'', p''), (p''', h'')} ∪ {(p, h'')|(p, t) ∈ F};
40 |  |  F' = F' \ {(p, t)|(p, t) ∈ F};
41 |  |  F' = F' ∪ {(end_LIG, p'), (p', h'), (p''', end_LIG), (start_M, p''')};
42 if T_ME = {} then
43 |  forall p_s ∈ P_S do
44 |  |  F' = F' ∪ {(p_s, start_LIG)};
45 if T_MS = {} then
46 |  forall p_e ∈ P_E do
47 |  |  F' = F' ∪ {(end_LIG, p_e)};
48 M' = (P', T', F', A ∪ λ, ℓ ∪ l, m_i, m_j);
```

***Figura 24**: Parte algoritmo per la riparazione aggiuntiva del modello*

Prendendo come esempio il modello raffigurato in figura 25, che rappresenta un modello di processo riparato secondo l'algoritmo discusso finora, questa riparazione viene fatta perché potrebbero verificarsi delle situazioni in cui, nel caso di eventi corrispondenti a move on model (ossia attività presenti nel modello ma non registrati nella realtà dal registro di eventi, come nel caso della transizione t_2 in figura 25), essi potrebbero essere eseguiti nella stessa esecuzione insieme alla parte di LIG aggiunta al modello per la riparazione (transizione t_{15} in figura 25). Questo però non dovrebbe verificarsi, poiché se viene eseguito il move on model considerato, non dovrebbe essere eseguita la parte di traccia aggiunta per la riparazione, e viceversa, dando la possibilità di saltare tali attività.
Per questo motivo la sezione aggiuntiva dell'algoritmo in figura 24, va ad inserire all'interno del modello alcuni place e transizioni nascoste (indicate con dei riquadri pieni) che permetterebbero di modificare il modello (come è visibile nella figura 26), in modo che se dovesse essere attivata la transizione t_2, di conseguenza non sarebbe possibile eseguire la parte di traccia aggiunta precedentemente per la riparazione del modello (in questo caso la transizione t_{15}) e viceversa.

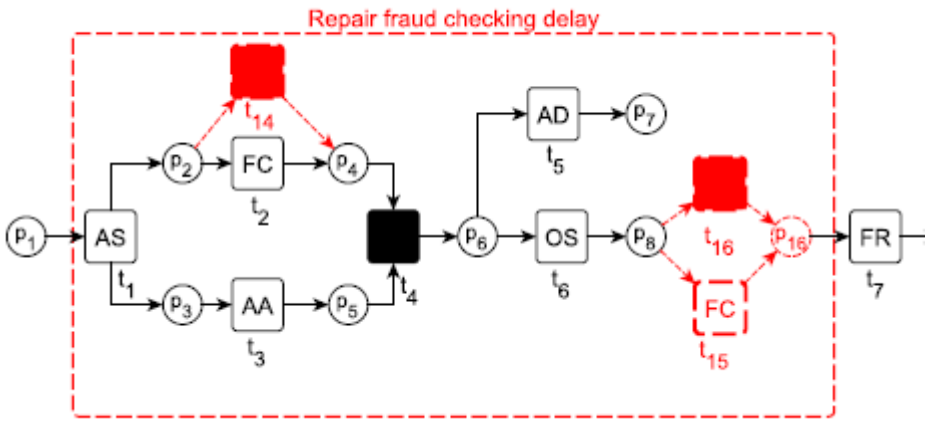


***Figura 25**: Esempio di modello di processo riparato con la transizione t_{15}*

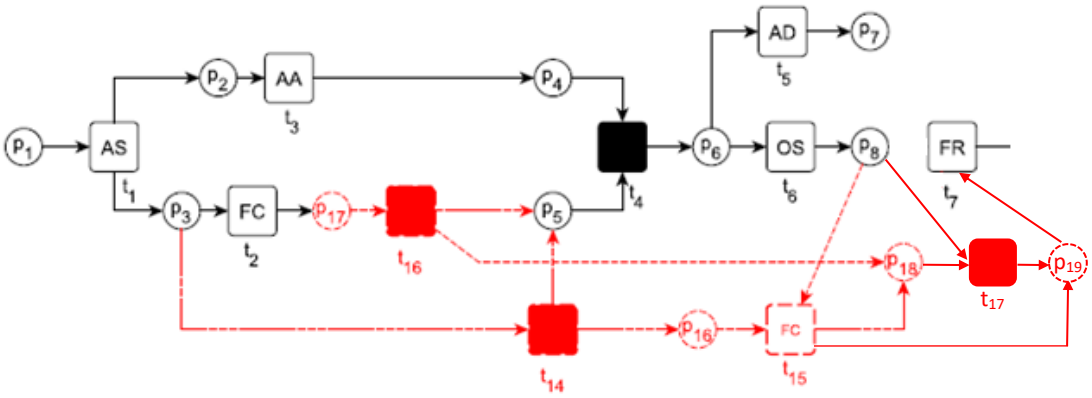


***Figura 26**: Esempio di modello di processo dopo la riparazione aggiuntiva*

Però effettuando i test, si è potuto constatare che questa ulteriore riparazione aggiuntiva del modello, nei casi in cui all'interno del modello di processo fossero presenti dei cicli, avrebbe portato alla violazione della proprietà di Soundness, e quindi il modello ottenuto non poteva più essere considerato valido e solido.

In figura 27 è riportato un esempio di un modello dopo la riparazione aggiuntiva.

Nel riquadro verde è indicato il move on log interno all'extended embedding considerato, che dovrebbe essere eseguito in alternativa alla parte di modello compresa nel riquadro rosso, che rappresenta la sezione di traccia anomala aggiunta al modello originale.

Da tale esempio risulta evidente che vi è la presenza di eventuali cicli all'interno del modello, e questo porterebbe alla generazione di token residui in alcuni places che rimarrebbero nella rete nonostante si riesca ad arrivare al marking finale.

Tale condizione viola la principio di correttezza di compilazione della proprietà di Soundness, non rendendo il modello riparato del tutto corretto, ed è per questo problema che si è deciso di non utilizzare questa riparazione aggiuntiva in questo lavoro.

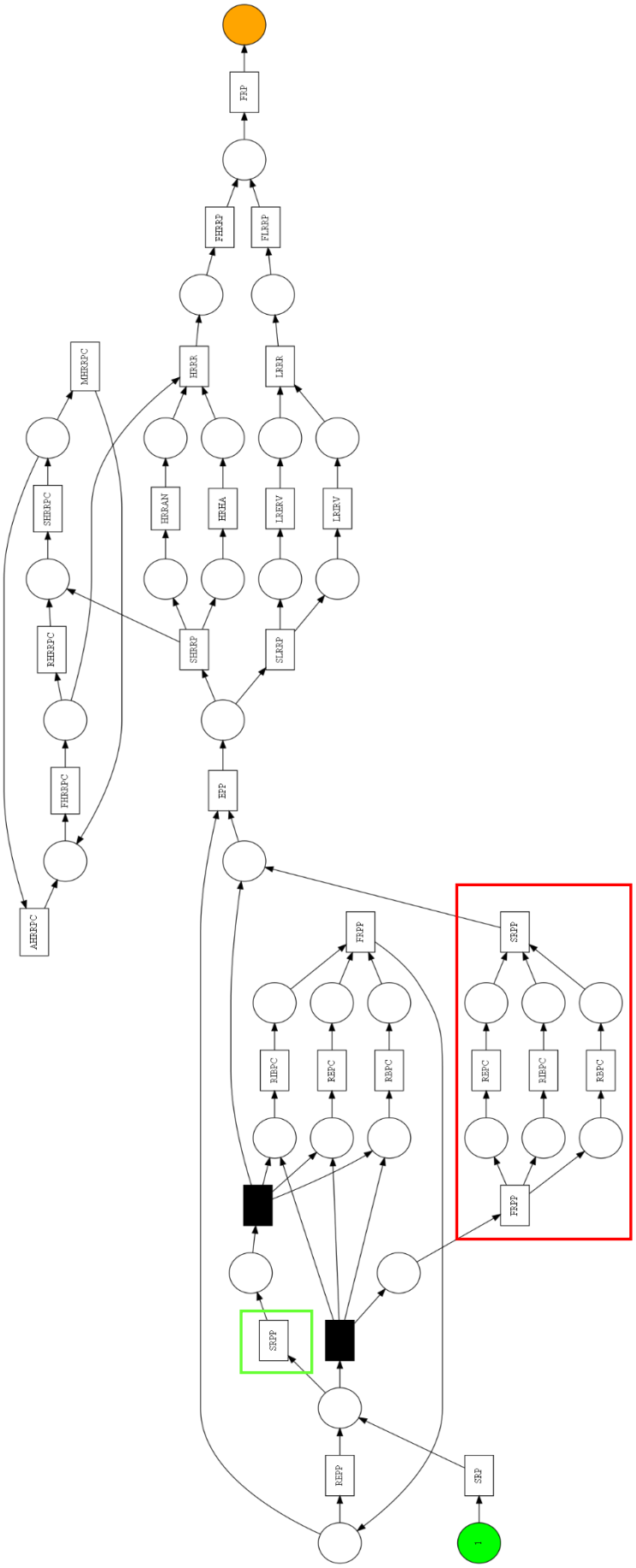


***Figura 27**: Esempio di modello che non rispetta la proprietà di Soundness*

Capitolo 6

CONCLUSIONI E SVILUPPI FUTURI

In conclusione, in questo lavoro di tesi è stato proposto e discusso un approccio di Process Repairing che apre a tante possibilità di riparazione e aggiornamento, e quindi di miglioramento del modello in questione.
Dall'analisi dei risultati effettuata, è possibile dire che questo approccio permette di avere delle misurazioni e dei valori delle metriche abbastanza buoni, consentendo di ottenere un modello riparato che possa replicare abbastanza fedelmente i valori registrati all'interno dei registri eventi.
Nell'approccio presentato, inizialmente viene scelto una traccia di riferimento che ci indichi la posizione esatta, cioè i places del modello, a cui agganciare il LIG scelto. Si suppone che tali attività del LIG occorrano sempre nella stessa posizione in ogni traccia, e qualunque venga scelta inizialmente, restituisca lo stesso marking di aggancio, anche se ciò non è sempre del tutto vero.
Nella valutazione dei risultati ottenuti, come è già stato spiegato in precedenza, non è stata presa in considerazione la metrica della Precision, che sarebbe molto interessante da misurare in maniera più accurata, poiché consente di dare una misura di quanto il modello si avvicini all'event log.
Sicuramente uno sviluppo successivo a tale lavoro, è quello di cercare di riparare il modello sull'intero pattern individuato, che comprende non soltanto un LIG, ma tentando di riparare il modello per più LIG di seguito.
Un altro possibile sviluppo, è quello di cercare testare tale algoritmo anche con altri dataset, in modo da poter verificare l'efficacia della soluzione proposta anche con differenti registri di eventi possibilmente anche di dimensioni più ampie.
Un'ultima possibilità da tenere in considerazione è quella di cercare ritoccare e di sviluppare meglio, la parte dell'algoritmo relativa alla riparazione aggiuntiva presentata nell'ultimo paragrafo del capitolo 5. In questo lavoro, tale soluzione è stata accantonata per i problemi descritti, ma sarebbe interessante da riuscire ad implementare poiché permetterebbe sicuramente di migliorare i risultati calcolati sul modello riparato, anche se porterebbe ad un inevitabile aumento della complessità del modello dato che introduce diversi places e transizioni aggiuntive nella rete.

BIBLIOGRAFIA E SITOGRAFIA

[1] Dirk Fahland, Wil M.P. van der Aalst: "*Model repair — aligning process models to reality*"; Eindhoven University of Technology, The Netherlands (2013)

[2] Laura Genga, Fabio Rossi, Claudia Diamantini, Emanuele Storti, and Domenico Potena: "*Model Repair Supported by Frequent Anomalous Local Instance Graphs*"; Eindhoven University of Technology, Università Politecnica delle Marche

[3] van der Aalst, W. and Adriansyah, A.: "*Process Mining Manifesto*"; In Daniel, F., Barkaoui, K., and Dustdar, S., editors, Lecture Notes in Business Information Processing (2012)

[4] W.M.P. van der Aalst, Process Mining: "*Discovery, Conformance and Enhancement of Business Processes*", Springer-Verlag, Berlino (2011)

[5] C. Diamantini, L. Genga, D. Potena, Wil van der Aalst: "*Building instance graphs for highly variable processes*"; Information Engineering Department, Università Politecnica delle Marche, Italy; Faculty of Mathematics and Computer Science, Eindhoven University of Technology, The Netherlands (2016)

[6] Sito ci PM4PY – Process Mining for Python, documentazione ultima release. (https://pm4py.fit.fraunhofer.de/)

[7] http://www.processmining.org/ - research tools application, © 2016 by the Process Mining Group, Math&CS department, Eindhoven University of Technology