Università degli Studi di Torino

Dipartimento di Informatica

Corso di Laurea in Informatica

Relazione di tirocinio

MINING DI SOCIAL COMMITMENT ANALIZZANDO LOG DI PROCESSI DI BUSINESS

Relatore:
Prof. Micalizio Roberto

Candidato:
Mossio Giacomo

Sessione DICEMBRE 2020
a.a. 2019/2020

Dichiarazione di originalità

"Dichiaro di essere responsabile del contenuto dell'elaborato che presento al fine del conseguimento del titolo, di non avere plagiato in tutto o in parte il lavoro prodotto da altri e di aver citato le fonti originali in modo congruente alle normative vigenti in materia di plagio e di diritto d'autore. Sono inoltre consapevole che nel caso la mia dichiarazione risultasse mendace, potrei incorrere nelle sanzioni previste dalla legge e la mia ammissione alla prova finale potrebbe essere negata."

Abstract

La quantità di dati prodotta dai sistemi informatici che controllano i processi è in costante aumento. Tramite l'analisi di queste informazioni si possono correggere e perfezionare i processi per far sì che venga migliorato il loro rendimento.
Esaminando i file di log in cui vengono memorizzate le attività che costituiscono un processo, si possono generare i social commitment che lo caratterizzano.
I commitment sono una sorta di accordo tra due agenti e la loro utilità è stata, ed è tutt'ora supportata, dagli articoli scritti a riguardo, che dimostrano come il loro utilizzo sia un'eccellente tecnica per la gestione della coordinazione degli agenti.
La tesi propone una visione generale dei commitment, con un'attenzione particolare a quelli riguardanti i processi di business; inoltre contiene un'indagine su alcuni degli strumenti e degli standard che compongono la sfera dei commitment. Successivamente la tesi si occupa della ricerca della presenza di commitment in determinati processi tramite l'esplicitazione delle caratteristiche che rendono un commitment tale. Infine descrive un algoritmo, detto miner, in grado di generare commitment tramite l'osservazione di log e suggerisce un'implementazione in python di tale miner, grazie all'aiuto di librerie apposite.

Indice

1. INTRODUZIONE

Il mining, cioè l'analisi, dei processi [1][2] può aiutare significativamente la comprensione dei processi stessi, e costituire eventuali miglioramenti che portano successivamente ad un aumento di produttività con la sola modifica di certe abitudini, senza la necessità di nuove implementazioni.
Nella letteratura attuale viene descritto l' "handover of work", che potrebbe essere tradotto in "trasferimento di lavoro", come la relazione tra due agenti che interagiscono in un sistema, misurata in base alla frequenza con cui un'attività di un agente *x* è causalmente seguita da un'attività di un agente *y*.
Grazie a questo legame si possono osservare le relazioni causali tra gli agenti, però questo parametro non tiene in considerazione le attività stesse poiché si concentra esclusivamente sulle risorse.
Uno strumento che si concentra invece sulle attività è la rete euristica, che cattura lo schema che contiene tutte le sequenze di attività di un processo.
Aggiungendo ad una rete euristica il principio che sta alla base dell' "handover of work", si possono ottenere le dipendenze causali tra le attività con associate le risorse che le eseguono; questo è il principio su cui si basa il mining di commitment.
È con lo scopo di coordinazione nei processi che nascono i commitment, definiti come uno strumento concettuale per modellare le relazioni sociali tra gli agenti che costituiscono un sistema.
I primi studi sui commitment risalgono al 1995 [8] e al 1999 [9]; nel corso degli anni sono state pubblicate varie ricerche a riguardo.
Un'altra definizione di commitment potrebbe essere: accordo tra agenti; questo accordo può addirittura catturare relazioni sociali e abitudini che altrimenti non verrebbero considerate. Si può dire che i commitment catturano relazioni non esplicite, o ad alto livello, ed è anche per questo motivo che hanno avuto una rilevanza nella letteratura scientifica.
L'uso dei commitment ha anche lo scopo di stabilire la compliance delle esecuzioni di un processo rispetto al loro modello. In altre parole, ha come obiettivo l'analisi di una determinata esecuzione di un processo in cerca di discrepanze dal modello del processo.
Un altro proposito per i commitment è quello di essere usati in modelli per il calcolo di diagnosi distribuite [13]. Grazie a queste procedure si ha la possibilità di catturare le "exception", cioè eccezioni, che si possono verificare durante un'esecuzione, con il fine di riorganizzare gli agenti che causano tali eccezioni.
L'obiettivo di questa tesi è proprio quello di studiare i commitment e proporre una metodologia e uno strumento per ricavarli direttamente dai log dei processi.
La ricerca effettuata ha seguito un iter di studio iniziale in cui ho indagato le caratteristiche principali che rendono un commitment tale; in secondo luogo ho

analizzato gli strumenti presenti sul mercato che permettono di lavorare con i commitment e, in un certo senso, aiutano anche la loro comprensione. L'ultimo passo è stato sviluppare un algoritmo che, basandosi su tutte le nozioni precedentemente apprese, genera i commitment a partire da un log in cui è presente il registro di tutte le attività che compongono un determinato processo, con un occhio di riguardo ai processi di business.
La ricerca è proseguita analizzando log di diversa complessità ed eseguendo adattamenti man mano che l'algoritmo prendeva forma, per ottenere uno strumento finale il più completo possibile.

La tesi è strutturata come segue:

- il capitolo 2 ha il compito di esporre quali strumenti e quali standard sono stati sviluppati con l'obiettivo del mining dei processi, utili al mining dei commitment
- il capitolo 3 si occupa dell'analisi dettagliata delle peculiarità dei commitment e delle entità che li circondano
- il capitolo 4 costituisce l'implementazione dell'algoritmo di estrazione dei commitment
- il capitolo 5 propone il case study, in cui si dimostra l'efficacia dell'algoritmo sviluppato su esempi reali
- il capitolo 6 trae le conclusioni dell'elaborato e propone alcuni possibili sviluppi futuri

2. STATO ARTE

2.1. Standard e Strumenti

Innanzitutto capiamo tutti i progressi che sono già stati effettuati nel campo del social mining per sapere quali programmi e strumenti sono disponibili al giorno d'oggi.
Se si parla di mining di processi, non si può non citare Wil van der Aalst che è l'autore dei libri fondamentali riguardanti questo campo quali ad esempio: "Process Mining: Data Science in Action" [1] e "Process Mining: Discovery, Conformance and Enhancement of Business Processes" [2].
In questi libri si espongono le motivazioni per cui analizzare i processi aziendali con le tecniche di mining può avere enormi vantaggi.
Per chiunque fosse interessato ad un approccio un pò meno teorico esiste anche il sito processmining.org dove si possono trovare informazioni utili, le notizie relative al tema, ed alcuni strumenti per lavorare con i processi, primo fra tutti il framework ProM.
ProM [3] è una piattaforma open source che mette a disposizione degli utenti, tramite un'interfaccia grafica, vari algoritmi e strumenti utili al process mining.

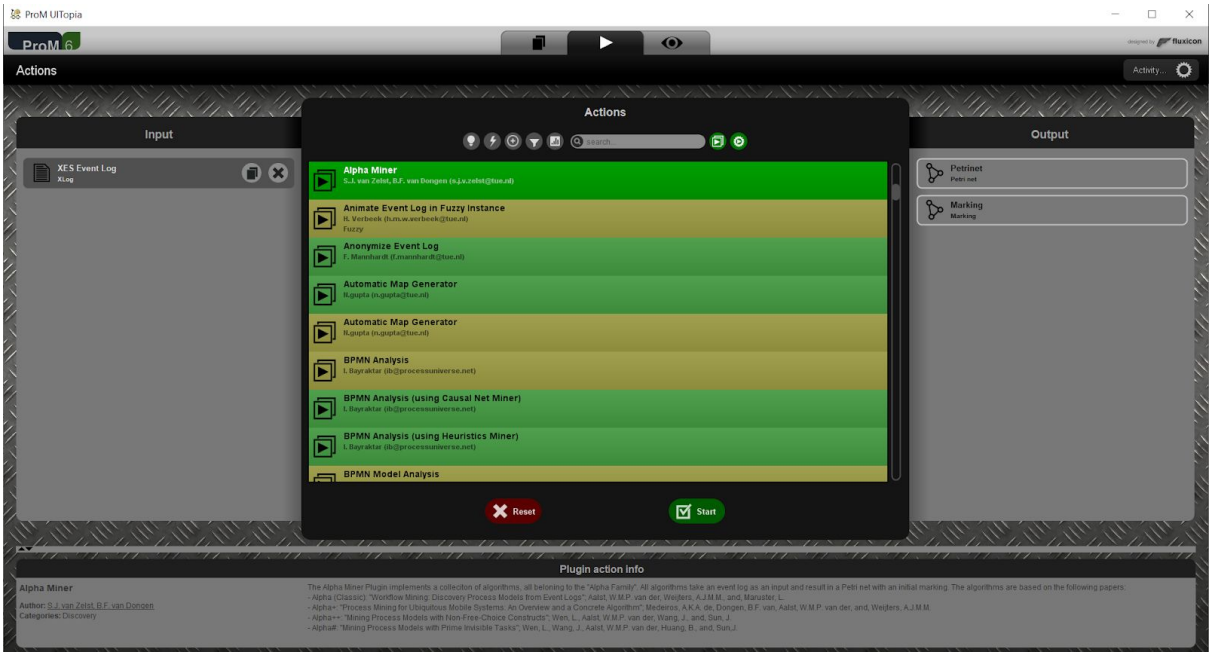


Figura 2.1 Interfaccia di proM

Lo scopo di ProM è quello di diventare la piattaforma standard nel mondo accademico grazie alla creazione e al supporto di una community aperta unificata in un framework facilmente espandibile.

Esplorando la piattaforma si possono incontrare vari tool fondamentali per lavorare con i processi, tra cui per esempio l'algoritmo per convertire i file di log nel formato XES.
Ma facciamo un passo indietro, cos'è il formato XES?
XES [4] è l'acronimo di eXtensible Event Stream, ed è stato approvato come standard ufficiale per gli event log dal IEEE l'11 novembre 2006.
Come mostrato in figura 2.2 Il Formato XES è caratterizzato da uno schema XML che descrive la strutture del log, da un altro schema descrittivo della struttura dell'estensione del log e delle eventuali estensioni.

```
<?xml version="1.0" encoding="UTF-8" ?>
<!-- This file has been generated with the OpenXES library. It conforms -->
<!-- to the XML serialization of the XES standard for log storage and -->
<!-- management. -->
<!-- XES standard version: 1.0 -->
<!-- OpenXES library version: 1.0RC7 -->
<!-- OpenXES is available from http://www.openxes.org/ -->
<log xes.version="1.0" xes.features="nested-attributes" openxes.version="1.0RC7">
        <extension name="Lifecycle" prefix="lifecycle" uri="http://www.xes-standard.org/lifecycle.xesext"/>
        <extension name="Time" prefix="time" uri="http://www.xes-standard.org/time.xesext"/>
        <extension name="Concept" prefix="concept" uri="http://www.xes-standard.org/concept.xesext"/>
        <classifier name="Event Name" keys="concept:name"/>
        <classifier name="(Event Name AND Lifecycle transition)" keys="concept:name lifecycle:transition"/>
        <string key="concept:name" value="XES Event Log"/>
        <trace>
                <string key="concept:name" value="0.1"/>
                <event>
                        <string key="lifecycle:transition" value="complete"/>
                        <string key="concept:name" value="postJob"/>
                        <date key="time:timestamp" value="2020-09-12T07:27:28.504+02:00"/>
                        <string key="RESOURCE" value="hirer1"/>
                </event>
```

Figura 2.2 Esempio di un file .xes

La finalità di tale formato è quello di garantire una comunicazione efficace tra i sistemi che producono i log focalizzati sulle attività e gli analizzatori di tali dati (sistemi o persone fisiche che siano).
Un passaggio quindi fondamentale è quello di convertire file, molto sovente generati in formato CSV, al formato XES per poi poterli analizzare con i vari algoritmi disponibili.
Un algoritmo basico è per esempio l'alpha algorithm, che viene considerato il più semplice per l'analisi di processi ma è allo stesso tempo molto rappresentativo poiché genera in output una rete di Petri dove si può notare l'ordine con cui le attività vengono eseguite e i possibili work-flow eseguiti.

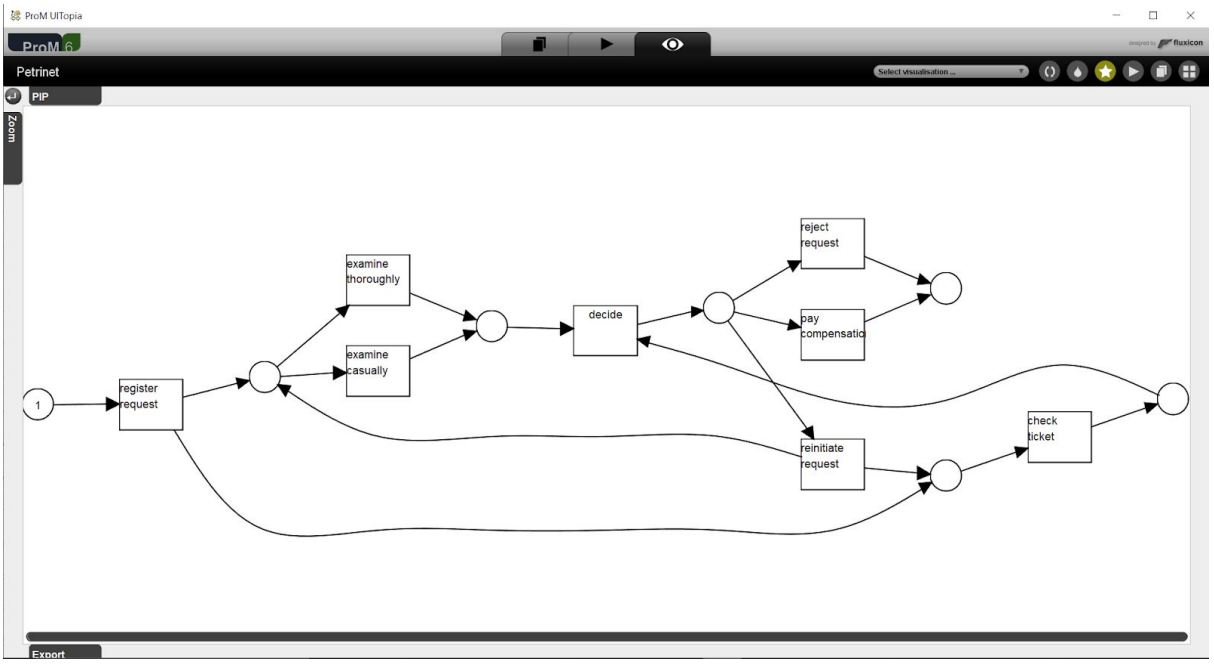


Figura 2.3 Esempio di una Petri Net

Esistono anche implementazioni di algoritmi leggermente più complessi come l'heuristic miner che, successivamente al settaggio di alcuni parametri, restituisce un grafico più informato rispetto all'alpha miner, con annessa la frequenza con cui vengono svolte le attività.

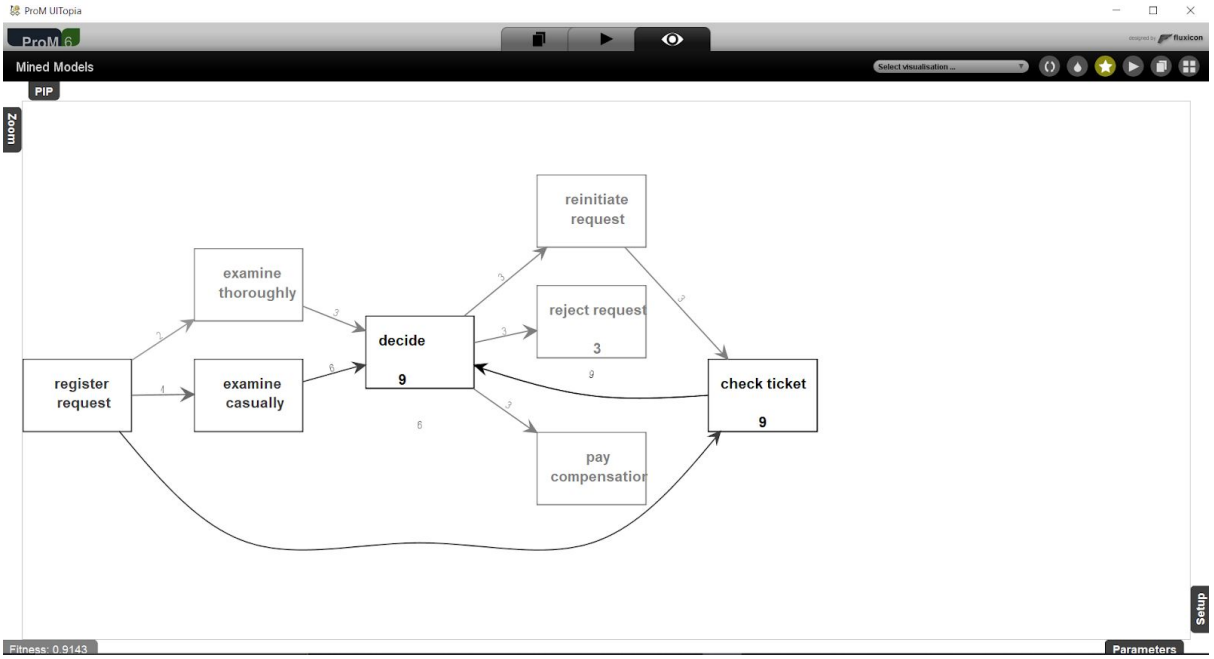


Figura 2.4 Rete euristica ricavata da un heuristic miner

Algoritmi più sofisticati e interessanti sono invece quelli riguardanti il mining di relazioni sociali. Tali algoritmi richiedono però il vincolo che gli eventi che compongono i log, abbiano sempre esplicato chi effettua la determinata attività.

Le reti sociali [5] sono utili perché possono aiutare a comprendere la struttura sociale dell'organizzazione in cui viene eseguito il processo, portando alla luce le attività simili, le risorse che subappaltano attività ad altre, quelle a cui vengono assegnate attività da altre risorse, quelle che tendono a lavorare insieme ed infine identificare le risorse tra cui avviene un passaggio di lavoro e che eseguono quindi attività in un determinato ordine.
Quest'ultimo fattore è di grande importanza per i commitment siccome l'handover of work (cioè il passaggio di lavoro) è una caratteristica da tenere in considerazione per il mining dei commitment dal momento che i commitment sono contratti stipulati tra risorse in cui si vincolano le esecuzioni di determinate attività.

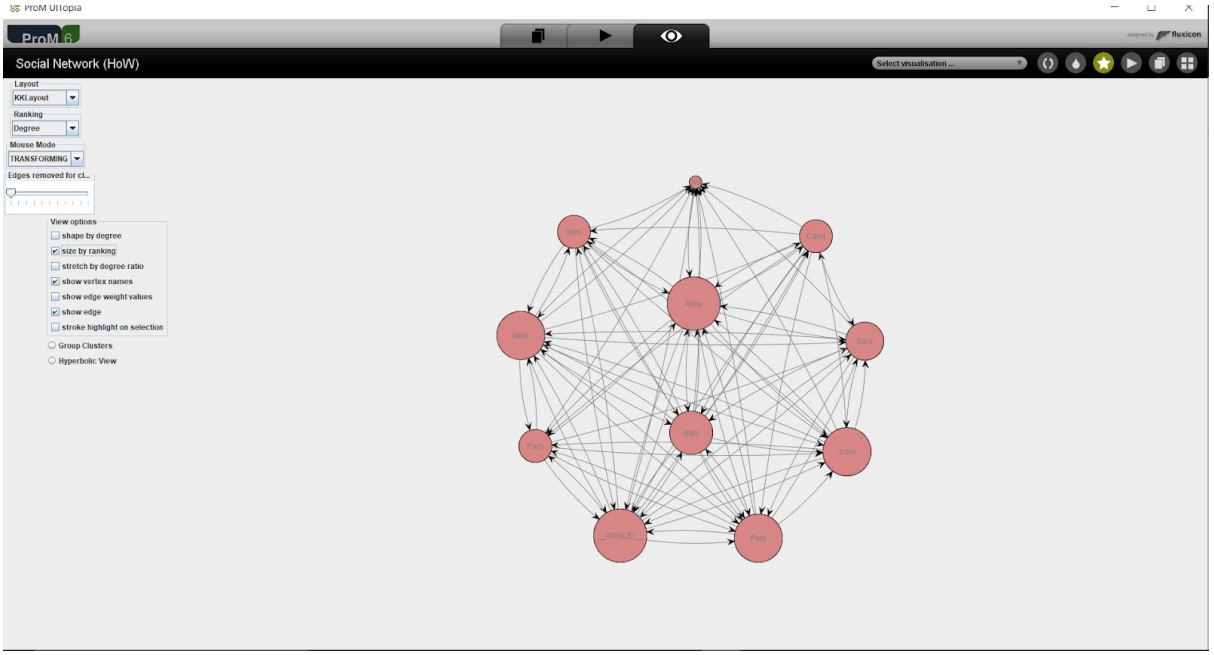


Figura 2.5 Rete dell'handover of work di un processo

ProM si rivolge ad utenti finali, offrendo un'interfaccia intuitiva e semplice da usare, non è pensato per essere integrato in altri strumenti di analisi. I risultati prodotti da ProM sono in genere grafici e non forniscono adeguato supporto agli sviluppatori per accedere alle strutture dati interne.
Un altro prodotto software più recente è invece la libreria open source PM4Py [6] nata con l'obiettivo di colmare le carenze di prodotti come proM.

Figura 2.6 Logo di PM4Py

La libreria PM4Py è interamente scritta in Python ed è stata sviluppata con l'ausilio di librerie di data science moderne come pandas, numpy, scipy e scikit-learn per

garantire un'ottimizzazione in quanto a tempi di esecuzione trattandosi, in alcuni casi, di moli di dati non indifferenti.
Al seguente link (http://pm4py.pads.rwth-aachen.de/benchmarks/) è possibile trovare un articolo, scritto da Alessandro Berti, uno sviluppatore di PM4Py, dove vengono paragonati i tempi di esecuzione di alcuni processi di strumenti di analisi di process mining.
Si evince che PM4Py risulta di gran lunga il software più rapido nella maggior parte dei test effettuati, tra cui gli import dei csv e dei log, le tecniche di process discovery, le tecniche di Conformance Checking e il filtraggio.
Il prodotto è stato creato dal team dedicato al process mining del Fraunhofer Institute for Applied Information Technology (FIT) [7], che è un istituto di ricerca senza scopo di lucro. Degno di nota il fatto che lo stesso Wil van der Aalst sia il consulente scientifico del team.
PM4Py, come ProM, include i principali e più importanti algoritmi descritti nei libri sopra citati, è ben documentato ed è semplice da usare per gli sviluppatori.
In aggiunta, l'accessibilità al codice sorgente in GitHub (https://github.com/pm4py) rende il tutto ancora più chiaro e comodo per lavorare.
Per lo scopo della tesi è stato fondamentale il suo utilizzo siccome gli algoritmi possono essere chiamati tramite poche righe di codice e, a differenza di proM, restituiscono un output ricco di informazioni chiaramente strutturate che possono essere utilizzate per vari scopi.
La libreria comprende funzioni per maneggiare e filtrare i dati degli eventi, per trovare i processi, per gestire le reti di Petri, per controllare la conformità dei log, per ottenere interessanti statistiche, per valutare i processi, per simulare i processi ed infine per analizzare le reti sociali presenti in un processo.

2.2. Social Commitment

Analizziamo ora cosa sono i social commitment e i progressi effettuati a riguardo.
Nel campo dei sistemi multiagente l'interazione fra i vari agenti è alla base di tutto; riuscire quindi a regolare il flusso di attività è imprescindibile ed è con questo scopo che nascono i commitment [8].
Un commitment si basa su una relazione diretta tra due agenti: un debitore e un creditore, e serve a vincolare tale relazione.
Un commitment C è così definito: $C(x, y, p_y, q_x)$ con cui la risorsa x (debitore) si impegna nei confronti di y (creditore) ad eseguire la condizione q_x quando la condizione p_y viene rispettata.
I commitment sono sempre creati dal debitore x, che stabilisce sia la condizione p_y, detta antecedente, sia la condizione q_x, detta conseguente. Detto ciò non è sempre vero che la condizione p_y è composta esclusivamente da attività svolte da y poiché può verificarsi che siano presenti attività di altri agenti. Lo stesso vale per q_x perché

non è sempre vero che la condizione è eseguita esclusivamente da x, ma l'importante è che x garantisca sempre l'avvenimento del conseguente q_x ove p_y venga rispettato. In alcuni casi è possibile che si verifichi il conseguente senza che l'antecedente avvenga.
Come enunciato nell'articolo di Singh [9], entrambe le condizioni possono essere composte da singole attività o da espressioni logico-temporali descritte con la proposizione logica *OR* ($\vee$) con il significato di scelta tra due attività e con il simbolo ($\cdot$) indicante la successione temporale delle attività.
Il fulcro della creazione di un commitment è l'interesse del debitore nell'esecuzione dell'antecedente da parte del creditore, dove il debitore, per ottenere l'esecuzione dell'antecedente, promette l'avvenimento del conseguente, al cui il creditore è interessato.
Si può dire che un commitment è una collaborazione tra agenti, in cui entrambi gli agenti sono interessati all'avvenimento di una condizione da parte dell'altro.
Il concetto di social commitment è abbastanza labile dal momento che non esiste una definizione precisa di esso e nel corso degli anni gli sono state attribuite varie interpretazioni; per esempio Castelfranchi definisce un social commitment come un accordo tra due agenti in cui entrambi lo accettano, mentre per Singh nel commitment, nonostante sia comunque noto ad entrambe le parti, non è presente un atto esplicito in cui il creditore lo accetta ed è perciò una sorta di tentativo di cooperazione intrapresa dal debitore. In questa seconda interpretazione l'accettazione da parte del creditore è, in un certo senso, rappresentata dall'esecuzione dell'attività antecedente.

La creazione di un commitment viene sempre eseguita dal debitore e la sua gestione avviene tramite le azioni: cancel (eseguita dal debitore), release (dal creditore), assign (dal debitore), delegate (dal creditore). Il ciclo di vita è rappresentato in figura 2.7.

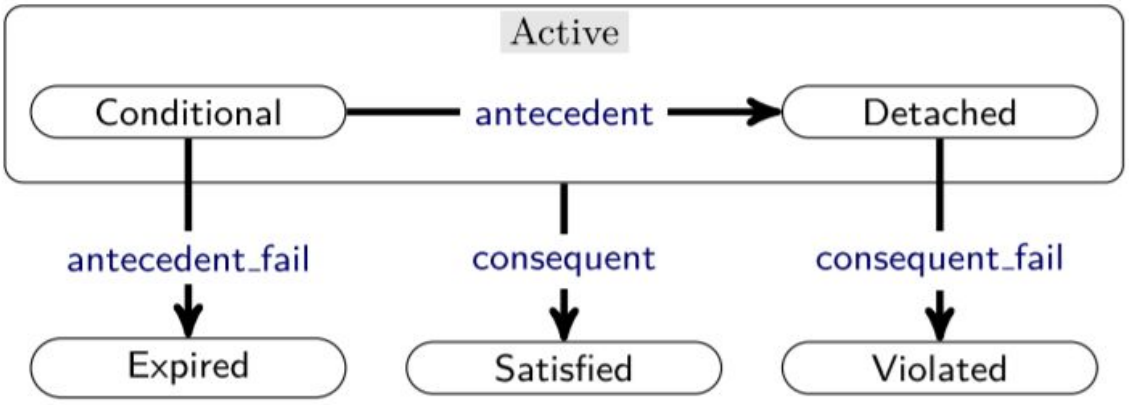


Figura 2.7 Ciclo di vita di un commitment

Lo scopo principale dei commitment è quello di coordinare le interazioni fra gli agenti cercando di vincolare obblighi, convenzioni e impegni di un certo livello di astrazione che normalmente non si riescono a catturare con semplici vincoli al tempo di esecuzione di un sistema.
Dato l'aumento nell'utilizzo di sistemi multiagente, la necessità della coordinazione tra gli agenti risulta imprescindibile ed inoltre l'utilità e l'importanza dei social commitment è già stata dimostrata nel corso degli anni nei vari articoli pubblicati a questo proposito.

3. MINING DI SOCIAL COMMITMENT

3.1. Caratterizzazione dei commitment lungo una linea temporale

Trattandosi di sincronizzazione degli agenti, un fattore fondamentale da tenere in considerazione è il tempo di esecuzione; per questo motivo si analizza quando una determinata attività viene eseguita, per ottenere un ordine cronologico.
I commitment si caratterizzano per un passaggio di lavoro (handover of work) tra agenti in un determinato ordine, è quindi necessario avere informazioni rispetto al momento preciso in cui avviene un'attività.
Ricordiamo che i commitment si descrivono come $C(x, y, p_y, q_x)$. In generale, nei commitment non esiste alcun legame causale tra antecedente e conseguente; in altre parole, un commitment può risultare soddisfatto, cioè si verifica il conseguente, anche se l'antecedente non si è verificato.
Nel caso specifico dei commitment nei processi di business, si assume la presenza di causalità. Per questo motivo, la forma dei commitment dei processi di business sarà $C(x, y, p_y, p_y \cdot q_x)$. La presenza dell'operatore before "$\cdot$" condiziona la temporalità degli eventi, così scritto il commitment vincolerà il verificarsi dell'antecedente affinché possa occorrere il conseguente; in tale modo si crea un vincolo causale tra antecedente e conseguente.
La causalità risulterà essenziale per l'individuazione dei commitment nell'algoritmo di mining successivamente descritto.
In ogni caso, non esiste nessun vincolo che impone che l'attività p_y sia direttamente seguita dall'attività q_x; non è raro che siano presenti altre attività in mezzo, che potrebbero non essere per nulla correlate con il processo che descrivono l'antecedente e il conseguente. L'ordine cronologico è uno degli aspetti chiave che aiuta a riconoscere quando la creazione di un commitment risulta necessario.
Se usassimo esclusivamente questo unico vincolo per il mining di commitment se ne otterrebbe una grande quantità poiché ci si concentrerebbe solo sull'ordine di tutte le coppie di attività presenti in un sistema. Tra la mole di commitment creati ce ne sarebbero però anche alcuni corretti.
Diversamente, come vedremo più avanti, considerando più vincoli riguardanti attività e risorse, si riescono ad ottenere commitment più precisi ed elaborati.

Nei processi di business, per verificare il vincolo temporale $p_y < q_x$ sarà necessario analizzare le tracce di entrambi gli agenti x e y; se si avesse solamente una delle due tracce, ottenere i commitment sarebbe molto difficile se non impossibile.
Inoltre quando si trattano commitment di processi di business, si assume che gli agenti presenti nel sistema siano reattivi, cioè che le esecuzioni delle attività si

verifichino come causa dell'avvenuta esecuzione di altre attività; in questo modo si può parlare di dipendenze causali tra attività.

Ora, un altro vincolo da imporre è che il creditore e il debitore non siano lo stesso agente, in caso contrario si perderebbe il senso stesso di cooperazione tra agenti; però vedremo successivamente che questa particolare tipologia di dipendenza sarà necessaria come passo intermedio nell'algoritmo di mining dei commitment.

3.2. Caratterizzazione dei log

Abbiamo già accennato alla tipologia di log necessaria per riconoscere i commitment, prima facciamo un passo indietro esplicando cosa è un log e successivamente vediamo nel dettaglio le caratteristiche fondamentali.
Un log, in generale, è un file o un insieme di file contenenti la registrazione sequenziale e cronologica delle operazioni che intercorrono in un sistema. Sono uno strumento utile per analizzare e monitorare i processi e vengono creati con lo scopo di analizzare gli errori, le statistiche, le modifiche alle basi di dati e le operazioni eseguite.
Esistono varie tipologie di log, nel nostro caso ci concentreremo sui log di sistema, dove vengono registrati tutti gli eventi significativi che avvengono.
Il formato del file che si genera è solitamente .csv, .txt o .xls dato che la struttura dei log è tabellare; da ciò ne deriva la necessità di una conversione al formato XES già sopra citato, considerato lo standard de facto per i log.

Case ID	Event ID	dd-MM-yyyy:HH.mm	Activity	Resource	Costs
1	35654423	30-12-2010:11.02	register request	Pete	50
1	35654424	31-12-2010:10.06	examine thoroughly	Sue	400
1	35654425	05-01-2011:15.12	check ticket	Mike	100
1	35654426	06-01-2011:11.18	decide	Sara	200
1	35654427	07-01-2011:14.24	reject request	Pete	200
2	35654483	30-12-2010:11.32	register request	Mike	50
2	35654485	30-12-2010:12.12	check ticket	Mike	100
2	35654487	30-12-2010:14.16	examine casually	Sean	400
2	35654488	05-01-2011:11.22	decide	Sara	200
2	35654489	08-01-2011:12.05	pay compensation	Ellen	200
3	35654521	30-12-2010:14.32	register request	Pete	50
3	35654522	30-12-2010:15.06	examine casually	Mike	400
3	35654524	30-12-2010:16.34	check ticket	Ellen	100
3	35654525	06-01-2011:09.18	decide	Sara	200
3	35654526	06-01-2011:12.18	reinitiate request	Sara	200

Figura 3.1 Esempio di log in formato .xls

La struttura della tabella può essere più o meno complessa però devono tassativamente esserci quattro colonne, cioè:

1. Una colonna per il numero di caso, a meno che non si tratti di una singola esecuzione (il che risulterebbe alquanto inutile per le nostre analisi). Questa indicazione si dimostra utile per distinguere i vari casi senza mescolare le attività di esecuzioni diverse del processo.
2. Una colonna indicante il nome dell'evento o attività che si sta registrando. Il nome dell'attività è banalmente fondamentale
3. Una colonna contenente l'agente, anche detto risorsa che esegue l'attività. Non tutti i file di log contengono tale informazione ma per il mining di commitment risulta indispensabile.
4. Una colonna che indichi la temporizzazione. Come discusso nel capitolo precedente, il tempo è una qualità imprescindibile quindi qualunque log si voglia analizzare deve avere una colonna che indichi in qualche modo la cronologia delle attività; ciò può essere ottenuto tramite un indicatore temporale (timestamp) che specifica il momento preciso nel quale viene eseguita un'attività o anche banalmente tramite un numero che indica l'ordine di esecuzione. Se si analizzano log di sistemi esistenti solitamente è presente la colonna dedicata al timestamp.

Altre eventuali colonne presenti non sono utili al nostro scopo e quindi verranno semplicemente ignorate. Per esempio la colonna "cost" nella figura 3.1 potrebbe rivelarsi molto utile per l'analisi di log più mirati all'economia e alla finanza di un'organizzazione.

Degno di nota è lo strumento di ProM che consente di visualizzare i log in formato XES offrendo varie informazioni a riguardo, tra cui:

- Una visione generale del log con informazioni riguardanti il numero di casi e il numero di eventi per caso.

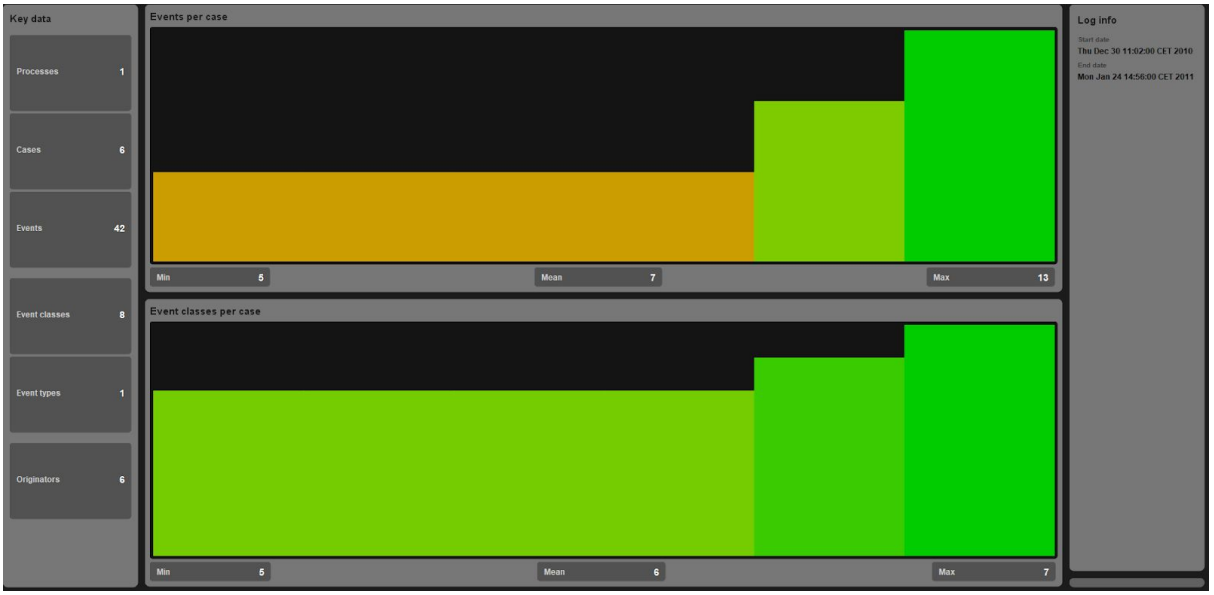
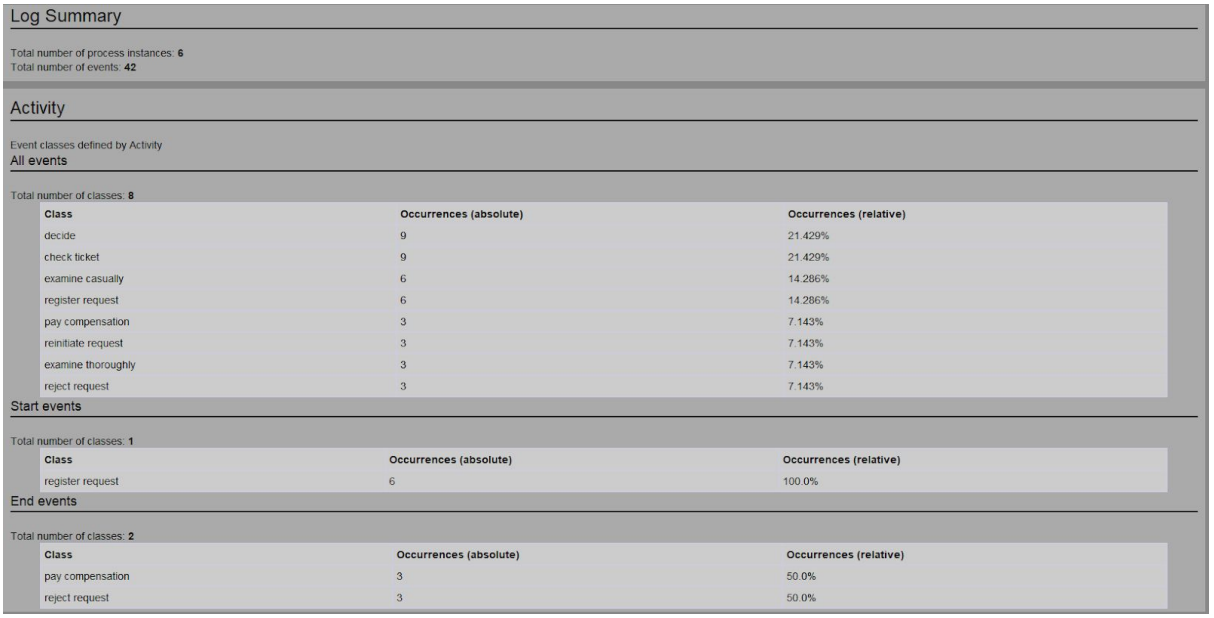


Figura 3.2 Visione generale di un log grazie a proM

- Un sommario raffigurante statistiche riguardo alle frequenze delle attività; utile per capire quali attività costituiscono quelle iniziali e quelle finali e l'occorrenza di esse.



Log Summary

Total number of process instances: **6**
Total number of events: **42**

Activity

Event classes defined by Activity
All events

Total number of classes: **8**

Class	Occurrences (absolute)	Occurrences (relative)
decide	9	21.429%
check ticket	9	21.429%
examine casually	6	14.286%
register request	6	14.286%
pay compensation	3	7.143%
reinitiate request	3	7.143%
examine thoroughly	3	7.143%
reject request	3	7.143%

Start events

Total number of classes: **1**

Class	Occurrences (absolute)	Occurrences (relative)
register request	6	100.0%

End events

Total number of classes: **2**

Class	Occurrences (absolute)	Occurrences (relative)
pay compensation	3	50.0%
reject request	3	50.0%

Figura 3.3 Sommario delle statistiche di un log con proM

- Vari grafici esplicativi che aiutano a visualizzare in modo chiaro e ordinato le tracce e gli eventi, alcuni anche nello spazio temporale.

Figura 3.4 Grafici delle attività con proM

Durante la ricerca effettuata questo strumento mi è risultato molto utile per comprendere i log senza dover leggere file di testo contenenti grandi quantità di dati. Come vedremo più avanti per l'implementazione finale ho optato per usare le funzioni di conversione dei file di log presenti nella libreria PM4Py.

3.3. Algoritmo heuristic miner per dipendenze causali

Abbiamo già accennato all'alpha miner ed è importante ricordare che questo algoritmo permette sì di ottenere una comprensione generale del processo, ma la rete ricavata (Petri net) non è ottimale siccome non copre ogni caso del processo e con processi complessi le reti risultano confuse. Inoltre non sono presenti meccanismi contro il disturbo e con file di log non perfetti, le reti prodotte sono fuorvianti siccome viene assegnata la stessa importanza alle dipendenze tra le attività sia se sono dipendenze molto frequenti, sia se si verificano anche una sola volta.

Per questa ragione, si consiglia di usare differenti tipologie di miner come per esempio l'heuristic miner dove viene valorizzata la frequenza con cui avvengono le attività.

Di seguito viene mostrato la differenza tra i due algoritmi eseguiti sullo stesso file di log per dimostrare come l'heuristic miner sia più intuitivo e chiaro.

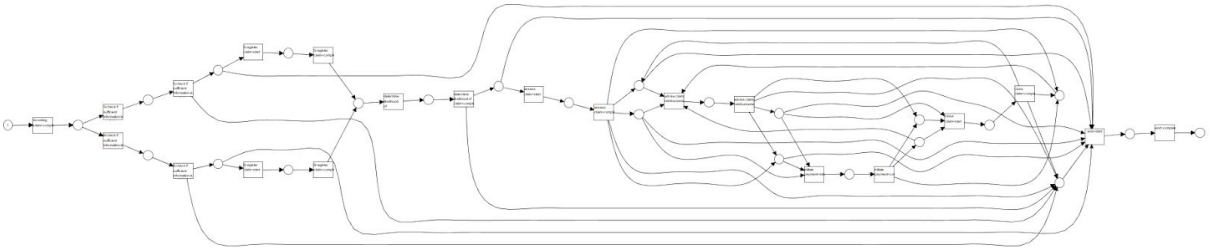


Figura 3.5 Rete di Petri ottenuta con Alpha Miner

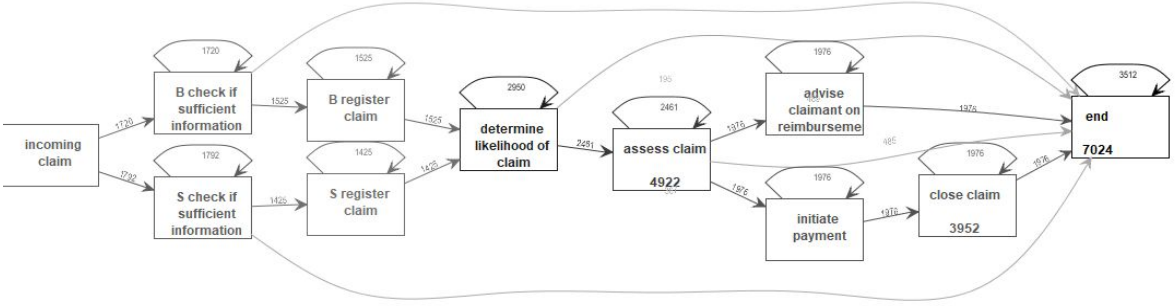


Figura 3.6 Rete euristica ottenuta con Heuristic Miner

L'heuristic miner si basa sulle dipendendenze causali, che si potrebbero definire come i valori che rappresentano quanto un'attività è legata ad un'altra.
In [2], i valori delle dipendenze causali assumono la seguente definizione:

Assumiamo un log L contenente un insieme di attività A e due attività $a, b \in A$.
$|a >L b|$ è il numero di volte che a è direttamente seguito da b.
$|a \Rightarrow L\ b|$ è il valore della dipendenza tra a e b e:

$$|a \Rightarrow L\ b| = \frac{|a >L\ b| - |b >L\ a|}{|a >L\ b| + |b >L\ a| + 1} \text{ se } a \neq b$$

$$|a \Rightarrow L\ b| = \frac{|a >L\ a|}{|a >L\ a| + 1} \text{ se } a = b$$

L'operazione $|a \Rightarrow L\ b|$ fornisce valori compresi tra -1 e 1. Se il valore si avvicina ad 1, a è spesso la causa di b ma non il contrario, in altre parole a è spesso seguita direttamente da b, ma b non è quasi mai seguita da a. In contrapposizione, se il valore fornito dall'operazione $|a \Rightarrow L\ b|$ è prossimo a -1, significa che b è spesso la causa di a ma non il contrario, l'esatto contrario rispetto al valore 1. Quindi per ogni coppia di attività abbiamo che $|a \Rightarrow L\ b| = -(|b \Rightarrow L\ a|)$
Da questi valori si può costruire una matrice contenente tutte le combinazioni tra le varie attività e le loro dipendenze.
Tramite questa matrice, chiamata dependency matrix, si può poi derivare la rete delle dipendenze che è costituita da nodi contenenti le varie attività e archi che collegano due nodi rappresentanti la dipendenza causale tra quelle due attività.
Sono inoltre presenti valori di soglia tramite i quali si possono modificare i risultati ottenuti, se per esempio volessimo rappresentare solo le dipendenze più significative

tralasciando quelle deboli, basterebbe inserire una soglia alta a $|\Rightarrow L|$ come per esempio 0.90 o anche discriminare i disturbi impostando una soglia a $|>L|$; per esempio con $|>L|>5$ le dipendenze non sarebbero considerate se la loro frequenza è ≤ 5.

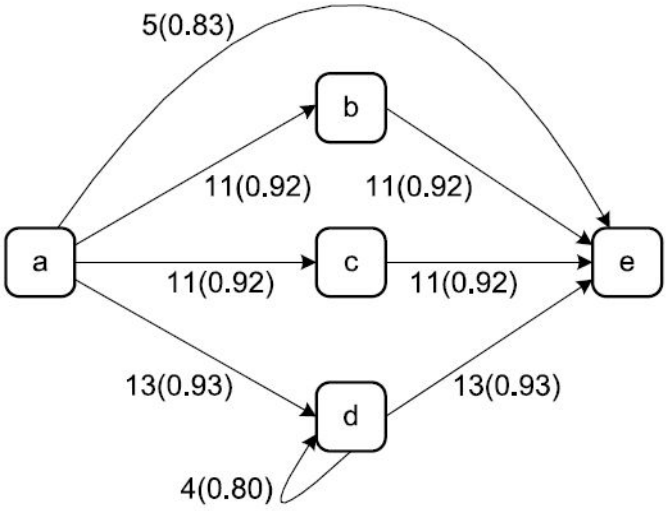


Figura 3.7 Esempio di un grafo delle dipendenze

L'heuristic miner genera infine un grafico che rappresenta la frequenza del workflow, cioè l'ordine in cui si eseguono le attività e quante volte esse si eseguono.

Tramite i valori di dipendenza si può capire quali attività sono dipendenti causalmente tra loro, in modo tale da poter successivamente vincolare tale relazioni con l'ausilio dei commitment.
Senza l'ausilio dell'heuristic miner, per esempio lavorando con una rete di Petri, sarebbe impossibile estrarre commitment.
L'algoritmo da me implementato sfrutta la matrice delle dipendenze costruita nell'implementazione di Pm4PY dove, tramite un comando, si genera le rete euristica contenente la matrice delle dipendenze.
L'implementazione del miner euristico di Pm4PY, come quella di ProM, segue passo a passo le istruzioni contenute in [2].

4. IMPLEMENTAZIONE DELL'ALGORITMO DI ESTRAZIONE DEI COMMITMENT

4.1. Metodologia dell'algoritmo

Prima di analizzare riga per riga il codice dell'algoritmo, procediamo con l'illustrazione dei passi astratti dell'algoritmo.
La struttura del codice è molto lineare e i passaggi sono sequenziali.
Si può dividere l'intero procedimento in 5 passaggi principali:

1. Caricamento dei log ed eventuale conversione
2. Applicazione dell'heuristic miner
3. Estrazione delle dipendenze causali
4. Creazione dei commitment base
5. Perfezionamento dei commitment

Il diagramma di flusso in figura 4.1 ci aiuta a comprendere meglio il funzionamento dell'algoritmo

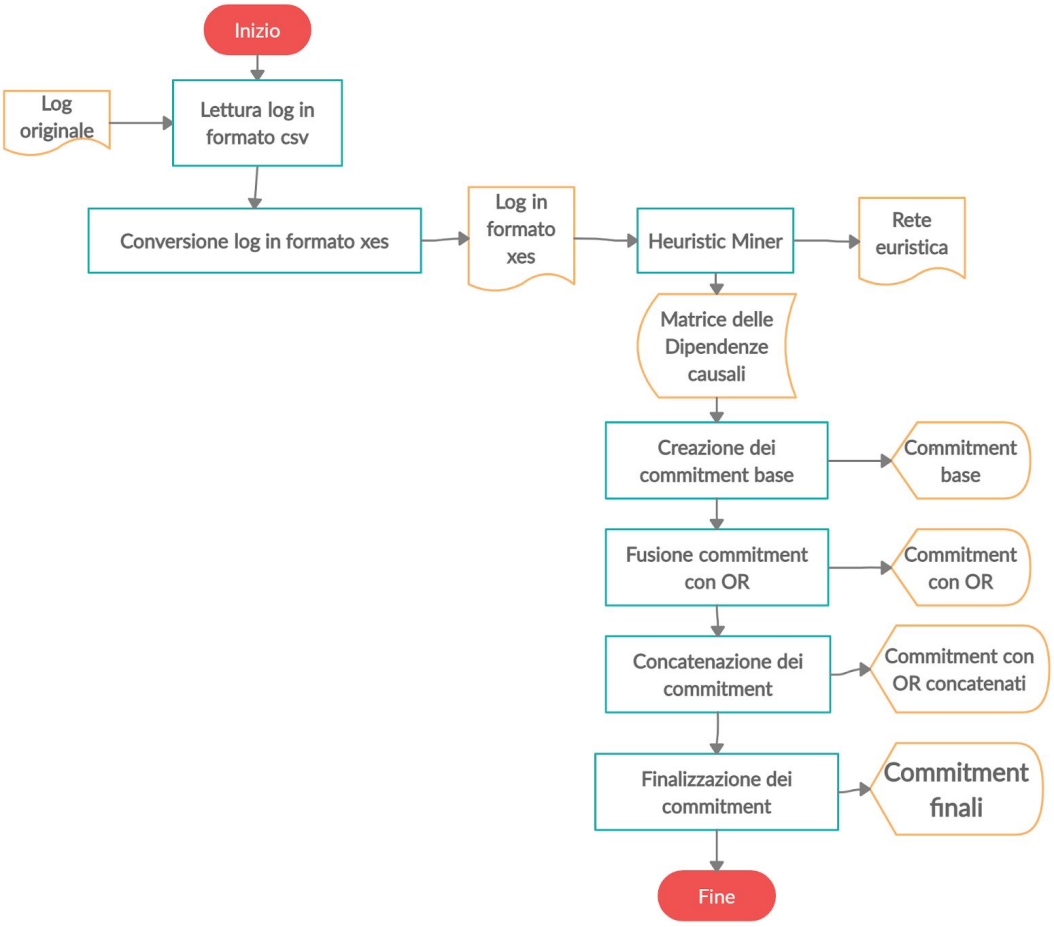


Figura 4.1 Diagramma di flusso dell'algoritmo

I rettangoli rappresentano i processi dell'algoritmo mentre le altre forme sono le rappresentazioni di file e dati.
Come possiamo notare, abbiamo un solo input che è il log, però si possono ricavare vari output come il log in formato xes, la rete euristica e tutte le fasi di lavorazione dei commitment.
La prima sezione, composta dalla conversione e dall'heuristic miner, viene eseguita grazie a funzioni offerte da PM4Py, mentre, una volta ottenuta la matrice delle dipendenze causali, si procede con funzioni appositamente create per i commitment.

4.2. Componenti necessarie al programma

Prima di esplorare il codice, descriviamo i componenti necessari affinché l'algoritmo possa funzionare.
Avremo bisogno di due librerie di Python: Pandas [10] e PM4Py.
Per quanto riguarda pandas, la libreria è inclusa in numpy (nota libreria di Python) e quindi chiunque abbia già installato il pacchetto, non dovrà scaricare niente.
In questo caso prenderemo come esempio un utilizzatore di un ambiente Windows per illustrare le componenti necessarie.
Invece, per Pm4PY sono necessari alcuni pacchetti di supporto quali:

- Microsoft Visual C++ Redistributable; è un ambiente di sviluppo integrato necessario all'esecuzione di alcune applicazioni, disponibile all'indirizzo https://support.microsoft.com/en-us/help/2977003/the-latest-supported-visual-c-downloads,
- GraphViz [11]; è una libreria di Python che, come suggerisce il nome, è fondamentale per visualizzare a video i grafi risultanti dall'applicazione degli algoritmi. Successivamente all'installazione del pacchetto, la sottocartella \bin della cartella GraphViz dovrà essere aggiunta manualmente alle variabili di sistema di Windows.

Giungiamo ora al punto saliente della tesi, cioè l'algoritmo vero e proprio per estrarre i commitment.
La scrittura di questo algoritmo è stata effettuata interamente in Python poichè, in questo modo, ho potuto usufruire delle librerie di Pm4Py e perchè considero Python il linguaggio più adeguato per questa tipologia di programmi potendo sfruttare anche librerie ottimizzate come numpy.
L'intero algoritmo si basa sui vincoli e sulle restrizioni già descritte in precedenza e i commitment si generano seguendo passaggi sequenziali.

4.3. Conversione dei log

Le prime righe del programma costituiscono gli import delle librerie necessarie.
Pandas serve per la sola lettura del file in input in formato csv.
Il resto degli import costituiranno tutti una parte della libreria Pm4PY.

```
import pandas as pd
from pm4py.objects.log.util import dataframe_utils
from pm4py.objects.conversion.log import converter as
log_converter
```

Successivamente abbiamo un blocco di comandi che costituisce la conversione di un file dal formato csv al formato xes, quest'ultimo necessario per l'analisi del log.
In un primo momento si carica il file in formato csv, grazie al comando di pandas read_csv, poi si converte la colonna del timestamp in un dataframe Pandas e si ordinano le righe in base al timestamp. Dopo, con il comando rename di Pandas si deve rinominare la colonna delle attività in "concept:name" dato che l'algoritmo di mining cercherà una colonna chiamata "concept:name" al momento dell'esecuzione e, nel caso in cui non la trovasse, restituirebbe un errore .
Nella variabile parameters si inseriscono gli eventuali parametri (vedere la documentazione per tutte le possibili opzioni), in questo caso si specifica qual è la colonna del numero del caso e infine si converte il log in formato XES con il comando "log_converter.apply".

```
log_csv = pd.read_csv('<file_name.csv>', sep=',')
log_csv = dataframe_utils.convert_timestamp_columns_in_df(log_csv)
log_csv = log_csv.sort_values('<name_of_the_timestamp_column>')
log_csv.rename(columns={'<name_of_the_activities_column>':
'concept:name'}, inplace=True)
parameters =
{log_converter.Variants.TO_EVENT_LOG.value.Parameters.CASE_ID_KEY:
'<name_of_the_case_number_column'}
log = log_converter.apply(log_csv, parameters=parameters,
variant=log_converter.Variants.TO_EVENT_LOG)
```

Con questa procedura il log viene memorizzato in una variabile; se invece volessimo salvare il log come file in formato XES, basterebbe eseguire il comando qui sotto riportato per memorizzare un file .xes nella cartella in cui si esegue il programma.

```
from pm4py.objects.log.exporter.xes import exporter as
xes_exporter
xes_exporter.apply(log, '<file_name>.xes')
```

4.4. Heuristic miner

Con tutti i parametri impostati correttamente, non ci resta che applicare l'algoritmo di mining.
L'esecuzione del comando "apply_heu" restituisce la rete euristica derivata dal log.

```
from pm4py.algo.discovery.heuristics import algorithm as
heuristics_miner
heu_net = heuristics_miner.apply_heu(log)
```

Anche in questo caso sono presenti vari parametri che è possibile modificare per ottenere una rete che sia più chiara possibile. Alcuni dei parametri sono:

- DEPENDENCY_THRESH; soglia sotto la quale le dipendenze causali tra attività non sono considerate (default: 0.5)
- MIN_ACT_COUNT; il numero minimo di volte che deve apparire un'attività per essere presa in considerazione (default: 1)
- MIN_DFG_OCCURRENCES; numero minimo di occorrenze di un arco per essere considerato (default: 1)
- LOOP_LENGTH_TWO_THRESH; soglia della dipendenza per i loop di lunghezza due (default: 0.5)

Una volta generata la rete è possibile visualizzarla grazie alla libreria GraphViz.
Per visualizzare a video la rete euristica con lo schema di tutte le attività e la frequenza con cui si transita tra esse, si eseguono i comandi:

```
from pm4py.visualization.petrinet import visualizer as
pn_visualizer
gviz = pn_visualizer.apply(net, im, fm)
pn_visualizer.view(gviz)
```

L'immagine ottenuta sarà una rete del tipo mostrato nella figura sottostante.

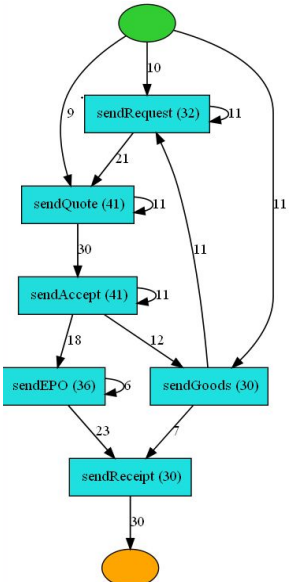


Figura 4.2 Visualizzazione di una rete euristica

Nel caso in cui invece volessimo visualizzare la rete di Petri, i comandi sono i seguenti (notare che c'è bisogno di salvare i tre oggetti net, im e fm anzichè la sola rete euristica net):

```
net, im, fm = heuristics_miner.apply(log)
from pm4py.visualization.petrinet import visualizer as
pn_visualizer
gviz = pn_visualizer.apply(net, im, fm)
pn_visualizer.view(gviz)
```

Il file .png ottenuto in questo caso sarebbe del tipo mostrato in figura.

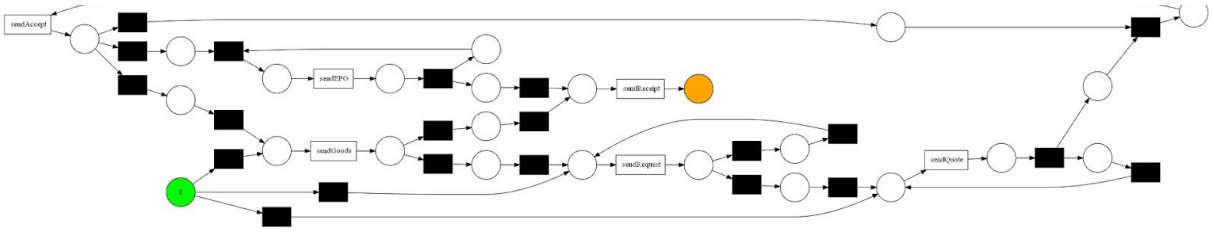


Figura 4.3 Visualizzazione di una rete di Petri

Visualizzare le reti, in particolare quella euristica, risulta in molti casi essenziale per capire il processo che si sta analizzando.

4.5. Dalle dipendenze ai commitment

Le dipendendenze causali, su cui si basa l'algoritmo, sono contenute in una struttura dati nell'oggetto restituito dall'algoritmo. La struttura in questione è un dizionario, cioè una specie di lista, che funziona tramite l'associazione di valori a delle chiavi. Per iterare attraverso il dizionario si usa un semplice ciclo for e tramite il comando .keys() è possibile ottenere una lista di tutte le chiavi presenti. Il dizionario contiene tutte le attività con associate a ciascuna di esse le altre attività come chiavi e i valori delle dipendenze come valori. Sono presenti esclusivamente le chiavi di attività con valori significativi, quelle cioè con valori alti.
Per esempio, con un semplice print della matrice si otterrebbe un dizionario del tipo:

{'postJob': {'apply': 0.9933333333333333}, 'apply': {'screenInterview': 0.9933333333333333}, 'screenInterview': {'makeOffer': 0.984375, 'rejectionNotice': 0.9886363636363636}, …

Osservando l'esempio si nota che, per la prima attività *'postJob'* è presente la sola chiave *'apply'*, ciò è dovuto al fatto che *'postJob'* abbia un'unica dipendenza causale significativa, che è *'apply'*; mentre altre attività possono avere più chiavi, *'screenInterview'* ne è la prova. Per ciascuna attività si può generalizzare lo schema:

'attività':{'altra_attività_1':'valore_dipendenza', 'altra_attività_2':'valore_dipendenza', ...}

e tramite gli appositi comandi si può navigare la struttura dati.

```
dep_list = []
for activity in heu_net.dependency_matrix:
    for act in heu_net.dependency_matrix[activity].keys():
        dep_list.append([activity.strip(), get_resource(activity),
act.strip(), get_resource(act),
heu_net.dependency_matrix[activity][act]])
        #dependency = (activity1, resource1, activity2, resource2,
dependency value)
```

Per ciascuna dipendenza si procede ad inserire in una lista le attività e le risorse che la costituiscono con il valore ad esse associato.

Per ottenere la risorsa che esegue una determinata attività, ho scritto una semplice funzione che cerca nel log la risorsa e la restituisce senza spazi bianchi e senza

l'ultimo carattere, siccome in molti casi le risorse sono composte dal ruolo e da un numero (esempio: "customer 2").

```
def get_resource(activity):
    for trace in log:
        for event in trace:
            if event["concept:name"] == activity:
                return event["RESOURCE"].strip()[:-1]
```

Ora giungiamo alla creazione dei commitment nella loro forma basilare.

```
ccs = []
for dep in dep_list:
    if  dep[4] > 0.80 and check_not_final(dep[0]) and dep[0] !=
dep[2]:
        ccs.append([dep[3], dep[1], dep[0], dep[2]])
        #cc(debtor, creditor, antecedent, consequent)
```

Nel codice si può notare che un commitment viene creato se la dipendenza tra due attività soddisfa tre vincoli:

1. La dipendenza deve essere abbastanza forte e superare un valore di soglia; i valori restituiti dall'algoritmo erano già stati discussi nel paragrafo 3.3 della tesi. È consigliabile adattare la soglia a proprio piacimento variando in un range da 0 a 1; una soglia bassa genererà molti commitment, però con il rischio di alcuni errori, mentre una soglia alta genererà meno commitment (quelli più consistenti) con la possibilità di tralasciare alcuni meno frequenti ma pur sempre corretti
2. Le attività non devono essere la stessa poiché vincolare un'attività con se stessa perderebbe di significato. Notare che, per ora, non sono presenti vincoli sulle risorse, quindi potrebbero essere presenti commitment di due attività eseguite dalla stessa risorsa
3. L'attività che costituisce l'antecedente non deve essere una delle attività finali del processo dato che le attività finali non devono essere seguite da nessun'altra attività. Per controllare le attività ho scritto una funzione qui sotto riportata

```
number_of_final_activities = 0
for key in heu_net.end_activities[0].keys():
    number_of_final_activities += heu_net.end_activities[0][key]
#used to check if an activity is not in more than 5% of the finals
```

```
ones
def check_not_final(activity):
    if " " + activity in heu_net.end_activities[0].keys():
        if (heu_net.end_activities[0][" " + activity] /
number_of_final_activities * 100) > 5:
            return False
    return True
```

La lista delle attività finali è salvata in un dizionario chiamato end_activities. Nella lista sono però presenti tutte le attività che, anche una sola volta, rappresentano l'ultima attività di un caso. Questo può risultare un problema, dato che in caso di rumore un'attività potrebbe essere considerata finale anche se non previsto che lo sia.
Per esempio, il ritardo nella comunicazione è un fattore comune da prendere in considerazione; tale ritardo potrebbe scaturire in un log incompleto, in cui l'ultima attività risulta essere un'attività intermedia dal momento che l'attività finale può non essere stata registrata nell'eventualità dello scadere di un timeout.
Un altro caso potrebbe essere l'ordine errato di registrazione delle richieste; nel caso di umani che gestiscono i log, questi ultimi potrebbero risultare a volte imprecisi, se magari una persona si dimentica di registrare un'attività appena eseguita; in maniera differente nel caso di log gestiti da sistemi informatici, l'esecuzione di attività potrebbe risultare talmente veloce da richiedere una precisione nel timestamp non indifferente per avere un ordine cronologico ben definito.
Per ovviare a queste avversità, si calcola prima il numero totale di attività finali e, successivamente un'attività verrà considerata finale solo nel caso in cui risulti l'ultima del caso in una percentuale significativa (in questo caso più del 5%). Anche questo valore può essere modificato a piacere per ottenere risultati migliori.

A questo punto abbiamo un'iniziale lista di commitment, che ricordiamo essere espressi nella forma: Commitment(Debitore, Creditore, Antecedente, Conseguente).

4.6. Perfezionamento dei commitment

Il passo successivo è quello di raffinare i commitment grazie al connettivo logico OR. Due commitment possono essere uniti se rispettano tre condizioni, nello specifico:

- L'antecedente deve essere comune ad entrambi i commitment, da ciò ne consegue che i creditori sono uguali
- I conseguenti devono essere diversi ma eseguiti dalla stessa risorsa; senza questo vincolo, il commitment risultante avrebbe poi due debitori differenti

Una situazione esempio in cui si creerebbe un commitment con *OR* è quando si hanno commitment come cc1 e cc2:

cc1('seller', 'buyer', 'make_offer', 'response_no')
cc2('seller', 'buyer', 'make_offer', 'response_yes')
in questo caso, dato che debitore e creditore sono uguali, gli antecedenti sono uguali e i conseguenti differenti si crea il commitment cc3:
cc3('seller', 'buyer','make_offer', 'response_no V response_yes')
I vincoli descritti e rappresentati nell'esempio sono sufficienti alla creazione del commitment con *OR* (*a V b*) ed escludono altre alternative. La presenza della disgiunzione è dovuta al fatto che cc1 e cc2 sono mutuamente esclusivi, cioè sono traiettorie alternative del processo e non possono mai verificarsi entrambe, occorre sempre o *a* (es. *yes*) o *b* (es. *no*).
Un'altra congiunzione come *AND* potrebbe invece verificarsi quando, a differenza dell'operatore OR, *a* e *b* occorrono nella stessa traiettoria, ma per catturare questo tipo di dipendenze ricorriamo all'uso dell'operatore "before".

```
add_ccs = []
for cc1 in ccs:
    for cc2 in ccs:
        if cc1 != cc2 and cc1[2] == cc2[2] and cc1[3] != cc2[3]
and cc1[0] == cc2[0]:
        #antecedent equal and consequent different and consequent
executed by same resource
            add_ccs.append([cc1[0], cc1[1], cc1[2], "(" + cc1[3] +
" V " + cc2[3] + ")"])
            ccs.remove(cc1)
            ccs.remove(cc2)

ccs = ccs + add_ccs
```

Nel codice si può notare la creazione di una lista di supporto nella quale verranno appesi i nuovi commitment con gli *OR* per far sì che la lista originale non venga compromessa. Una volta aggiunti i nuovi commitment alla lista di supporto, quelli che lo costituiscono possono essere eliminati. Infine si procede ad unire le due liste così da ottenere la lista completa con commitment semplici e commitment con OR.

Dopodiché possiamo procedere a concatenare i commitment attraverso il simbolo "·". La concatenazione risulta possibile quando due commitment rispettano le seguenti condizioni:

- Il conseguente del primo è uguale all'antecedente del secondo
- Il creditore del primo non è uguale al debitore del secondo. Nel caso in cui lo fossero, il commitment risultante avrebbe il debitore uguale al creditore e perderebbe quindi di senso

Se le condizioni vengono rispettate, si procede a controllare che nessuna attività del debitore venga inclusa nell'antecedente; se così fosse, tale attività si includerà nel conseguente.
Un esempio in cui si possono concatenare due commitment è con i seguenti cc1 e cc2:
cc1('seller', 'buyer', 'make_offer', 'response_yes')
cc2('shipper', 'seller', 'response_yes', 'send_goods')
il commitment concatenato cc3 sarà:
cc3('shipper', 'buyer', 'make_offer · response_yes ', ' send_goods')

Un altro esempio dove è presente un commitment con *OR* è:
cc1('evaluator', 'candidate', 'apply', 'screenInterview')
cc2('evaluator', 'evaluator', 'screeInterview', '(makeOffer V rejectionNotice)')
in questo caso si crea un commitment cc3:
cc3('evaluator', 'candidate', 'apply', 'screeInterview · (makeOffer V rejectionNotice)')

Una volta concatenate le attività si procede con l'eliminazione del commitment che è stato inglobato nella concatenazione. Questo passaggio non è sempre consigliato, siccome se non cancellassimo i due commitment, essi potrebbero risultare utili per altre concatenazioni; in questa implementazione si eliminano siccome si desiderano ottenere risultati semplici e compatti.

```
    for cc1 in ccs:
        for cc2 in ccs:
            if cc1 != cc2 and cc1[3] == cc2[2] and cc1[1] !=
cc2[0]:
                if cc2[0] == cc1[0]:
                    cc1[3] = cc1[3] + " · " + cc2[3]
                    ccs.remove(cc2)
                else:
                    cc2[1] = cc1[1]
                    cc2[2] = cc1[2] + " · " + cc2[2]
                    ccs.remove(cc1)
```

La presenza del ciclo for iniziale è dovuta al fatto che, in questo modo si possono creare concatenazioni lunghe a piacere. Il range della variabile *i* rappresenta la lunghezza massima possibile di un concatenamento siccome l'algoritmo controllerà i-volte la possibilità di creare concatenamenti. In questo caso si potranno concatenare fino a 5 attività in un solo commitment.

Come già descritto nel paragrafo 3.1, dato che l'intero algoritmo si basa sul principio di causalità presente nei processi di business, per rappresentare il vincolo di causalità tra antecedente e conseguente, il commitment finale sarà della forma C(x, y, p_y, $p_y \cdot q_x$).
Con i seguenti comandi, in ciascun commitment verrà anteposto l'antecedente, seguito dall'operatore before "·", al conseguente.

```
for cc in ccs:
    cc[3] = cc[2] + " . " + cc[3]
```

A questo punto non bisogna dimenticare che all'inizio si erano creati commitment senza controllare che debitore e creditore non fossero la stessa risorsa; perciò il passo finale del procedimento consiste nell'eliminare i commitment in cui debitore e creditore sono la stessa risorsa.

```
for cc in ccs:
    if cc[0] == cc[1]:
        ccs.remove(cc)
```

L'algoritmo risulta completo, nella lista ccs avremo tutti i commitment e tramite un semplice print potremo visualizzarli.

```
print("\nFINAL COMMITMENTS:")
for cc in ccs:
    print(cc)
```

4.7. Osservazioni finali

L'intero codice dell'algoritmo è disponibile su github all'indirizzo https://github.com/Jack748/CommitmentsMiner dove sono inoltre presenti alcuni file su cui è possibile testare l'algoritmo.
L'algoritmo proposto si focalizza sui processi di business ed è migliorabile sotto diversi aspetti però, come vedremo successivamente, offre risultati interessanti.

Per ottenere un algoritmo il più possibile adattabile e flessibile, sono presenti degli iper-parametri già citati in precedenza. I principali parametri modificabili sono:

- Il valore di soglia della dipendenza per la creazione dei commitment (consigliato a 0.9)
- Il valore di soglia per considerare un'attività finale o meno (consigliato a 5%)

- Il valore della lunghezza di concatenazione massima (consigliato a 5)

Inoltre, in aggiunta, si possono modificare tutti i parametri dell'heuristic miner già esposti nel capitolo 4.3.
La modifica degli iper-parametri avrà come risultato la creazione di commitment differenti, più precisamente se si vogliono ottenere una grande quantità di commitment sacrificando la precisione di essi, si può abbassare il valore di soglia delle dipendenze mentre per ottenere pochi commitment ma precisi si può alzare il valore.
Se sapessimo che il log che stiamo analizzando fosse perfetto e senza rumori, allora potremmo impostare il valore di soglia per le attività finali a zero dato che le sole attività finali sarebbero considerate tali; si consiglia comunque di mantenere un valore maggiore di zero.
Per quanto riguarda la lunghezza delle concatenazioni, la scelta dell'iper-parametro è completamente a discrezione dell'utente. In alcuni casi avere commitment con catene di attività lunghe può risultare scomodo e inconveniente mentre in altri casi risulta più ordinato e significativo.
Come vedremo successivamente nel case study, la modifica di questi parametri può influenzare enormemente l'esito dell'applicazione dell'algoritmo.

5. CASE STUDY

5.1. Il protocollo NetBill

Cominciamo le nostre prove con un protocollo relativamente semplice.
Il NetBill protocol è un sistema utilizzato nel mondo del commercio per eseguire uno scambio di beni con relativo pagamento.
Il NetBill è un protocollo e non un processo, quindi non si devono assumere relazioni di causalità e da ciò ne consegue che l'antecedente non verrà riportato nel conseguente, a differenza dei processi di business; di fatto la modifica al programma è costituita dall'eliminazione delle due righe di codice che copiano l'antecedente nel conseguente.
Questo protocollo è costituito da due agenti (cliente e commerciante) che interagiscono tramite un insieme di attività.
La sequenza standard di attività segue l'ordine:

A) Il cliente richiede un prezzo al commerciante
B) Il commerciante fa un'offerta al cliente
C) Il cliente accetta l'offerta
D) Il commerciante spedisce le merci al cliente
E) Il cliente invia l'Electronic Purchase Order (EPO) al commerciante
F) Il commerciante invia la ricevuta dell'acquisto al cliente

Nonostante sia presente un ordine standard per le attività, non sempre il flusso segue questo percorso. Ad esempio, il comunicante potrebbe presentare un'offerta anche senza che il cliente abbia richiesto un prezzo.
Analizzando un file di log generato da un programma che simula le esecuzioni di processi in cui gli agenti sottostanno al protocollo NetBill, la rete euristica ottenuta è la seguente mostrata in figura 5.1:

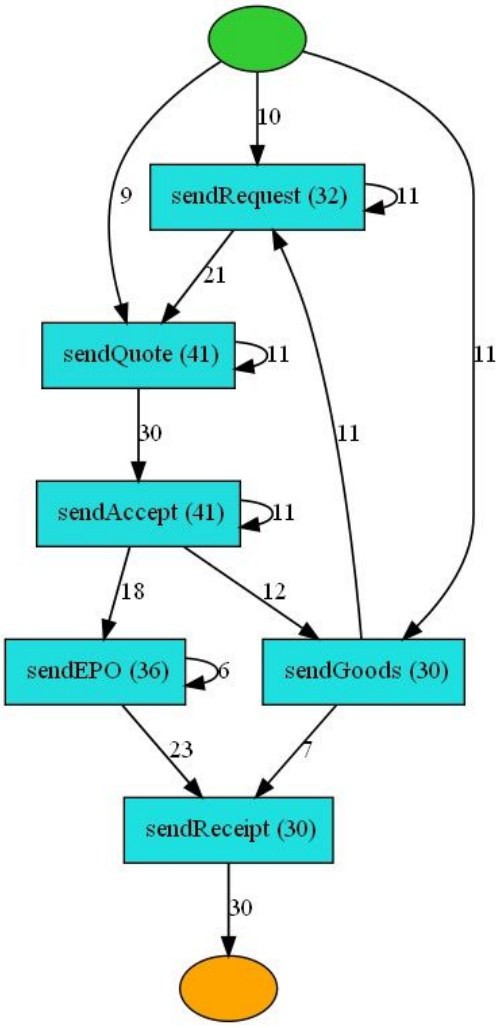


Figura 5.1 Rete euristica ricavata da un processo NetBill

Come possiamo notare, l'ordine predominante è quello standard, però quest'ordine non è sempre rispettato; basti guardare l'attività iniziale che solo in 10 casi su 30 è costituita da *sendRequest*. Ciò rappresenta la possibilità di un inizio con un'attività differente come può essere *sendQuote* o anche *sendGoods*.
Non essendo un processo, non è fondamentale seguire l'ordine prestabilito; l'importante è rispettare il protocollo.

In questo particolare case study la relazione presente tra gli agenti è una relazione uno a uno per cui anche la rete euristica risulta di facile lettura e i commitment che si genereranno non potranno essere molto complicati.

Eseguendo l'algoritmo con il valore di soglia delle dipendenze per la creazione dei commitment a 0.8 e gli altri parametri quelli consigliati, ciò che otteniamo sono i seguenti commitment:

1. *C('merchant', 'customer', 'sendRequest', 'sendQuote')*

2. *C('customer', 'merchant', 'sendQuote', 'sendAccept · sendEPO')*
3. *C('merchant', 'customer', 'sendAccept', 'sendGoods · sendReceipt')*
4. *C('merchant', 'customer', 'sendEPO', 'sendReceipt')*
5. *C('customer', 'merchant', 'sendGoods', 'sendRequest')*

Osservandoli possiamo notare che descrivono perfettamente il protocollo.
Il commitment numero 5 potrebbe sembrare errato dato che non sempre l'invio dei beni è seguito dall'invio di una richiesta però è giusto poiché quella sequenza di attività si può verificare; ciò non comporta che questa determinata sequenza debba sempre verificarsi. Alcune coppie di commitment potrebbero essere considerate mutuamente esclusive, nel senso che la realizzazione di un commitment rappresenta automaticamente l'impossibilità della realizzazione dell'altro. Per esempio, se il commitment numero 3 si realizzasse, quello numero 5 non potrebbe più verificarsi siccome l'attività sendReceipt costituisce sempre un'attività finale.

5.2. Il business process dell'Hiring

In questo caso trattiamo un processo di business, riguardante nello specifico l'assunzione di un impiegato.
La figura 5.2 è estratta dall'articolo [12].

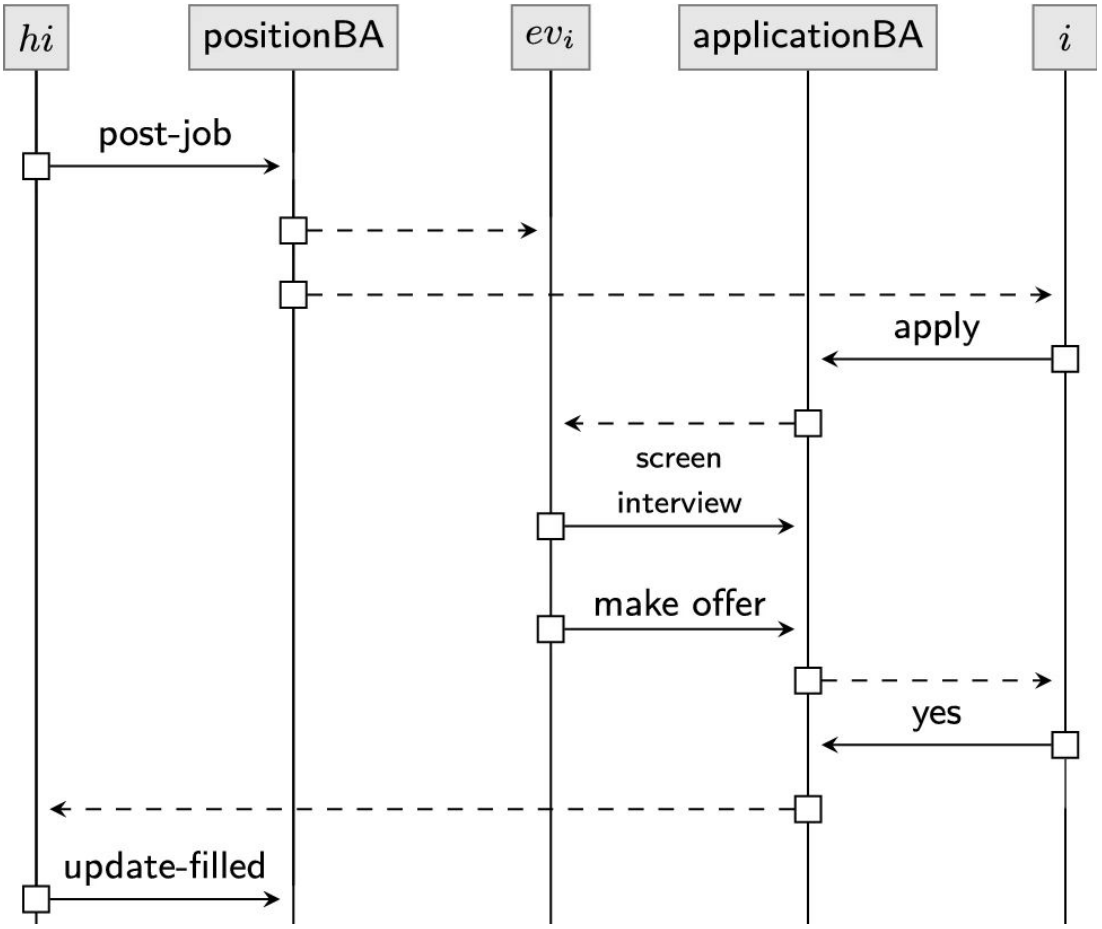


Figura 5.2 Rappresentazione del processo di Hiring

Osservando l'immagine possiamo intuire il processo che consiste in:

A) Un datore di lavoro (*hirer*) pubblica l'offerta per un posto di lavoro
B) Un candidato (*candidate*) si candida per ottenere il posto
C) Un valutatore (*evaluator*) effettua il colloquio con il candidato
D) Se la valutazione è negativa, il valutatore comunica al candidato che è stato respinto; se invece la valutazione è positiva, il valutatore procede ad effettuare un'offerta
E) Il candidato che riceve l'offerta, comunica il rifiuto o l'accettazione
F) Se il candidato ha accettato, il datore di lavoro procede alla chiusura del posto di lavoro

Per ciascuna esecuzione sono presenti un hirer, alcuni valutatori e vari candidati.
In questo processo entra in gioco la relazione uno-a-molti dato che un singolo valutatore può dover intervistare più candidati.
Questo è stato il processo su cui sono stati effettuati più test, e la relazione uno-a-molti ha costituito un ostacolo che però è stato abbattuto modificando il file di log generato in modo che fossero presenti tante istanze di *post-job* quante *apply*. Di norma, dovrebbe essere presente un *post-job* singolo per esecuzione, però in questo modo non si sarebbe potuta catturare bene la relazione causale tra post-job

ed apply, mentre riportando *post-job* per ogni candidato, la dipendenza causale tra le attività è molto più forte.
Inizialmente si sono analizzati processi in cui il numero di valutatori era limitato per semplificazione per arrivare poi a processi più completi e complessi.
Infine, eseguendo l'algoritmo su un file contenente l'unione di tutti i log riguardanti questo tipo di processo si sono ottenuti i seguenti risultati:

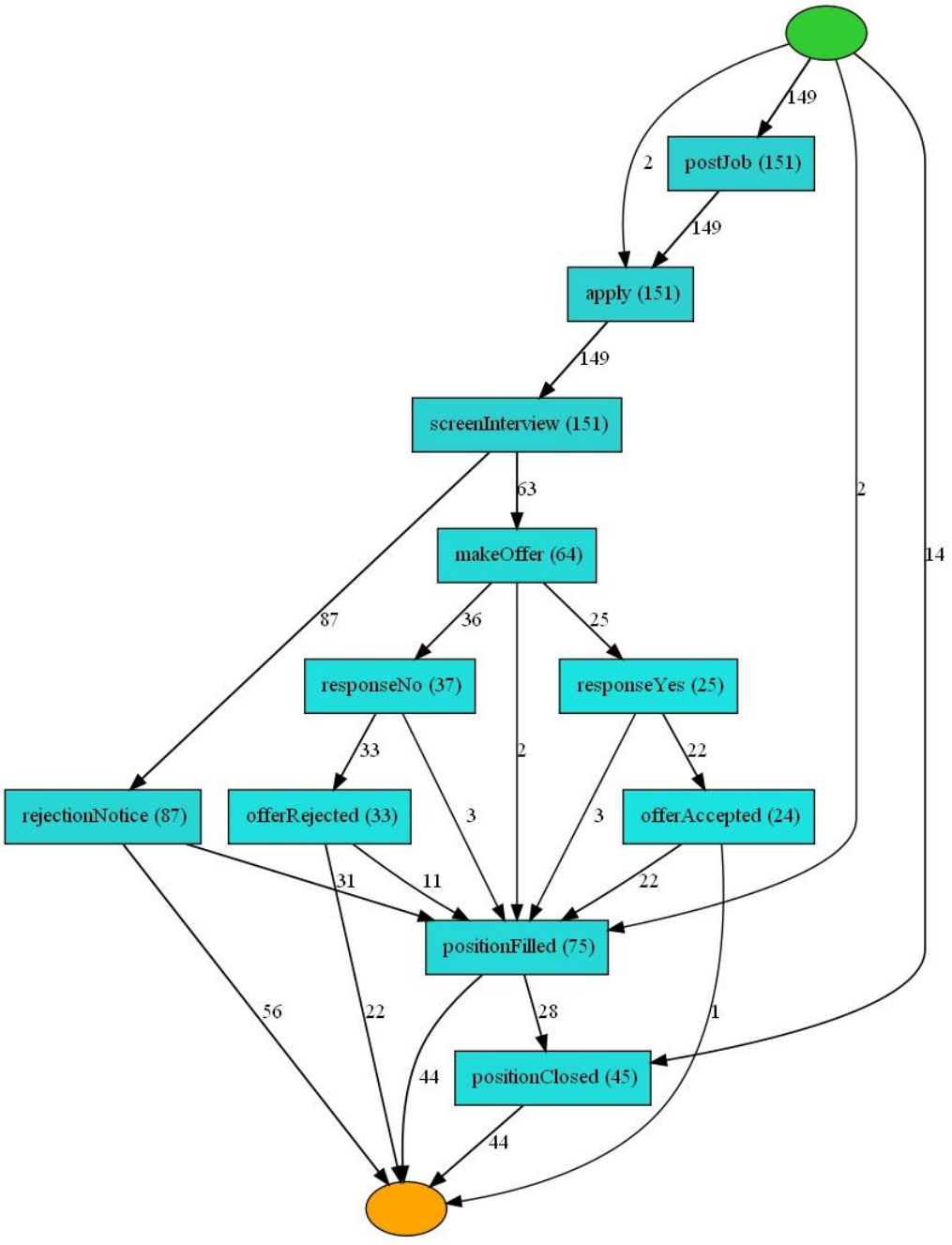


Figura 5.3 Rete euristica ricavata da un processo di Hiring

Questa è la rete euristica che rappresenta la frequenza dei percorsi di attività di ogni esecuzione.

Come possiamo notare, ci sono attività fortemente legate tra loro e che avvengono spesso in successione, per esempio *apply* e *screenInterview* o *responseNo* e *offerRejected*.
Altre coppie di attività invece, sono mutuamente esclusive, per esempio *rejectionNotice* e *makeOffer* o anche *responseYes* e *responseNo*.
Essendo gli agenti indipendenti l'uno dall'altro, può avvenire l'esecuzione di attività in un ordine apparentemente errato; per esempio vediamo che *positionFilled* può verificarsi in diversi momenti del processo; trattandosi di comunicazione tra agenti, tutto ciò è normale ed è anche per questo che nascono i commitment.
Ora passiamo ad analizzare i commitment estratti dall'esecuzione dell'algoritmo:

1. *C('evaluator', 'hirer', 'postJob · apply', 'postJob · apply · screenInterview . (makeOffer V rejectionNotice)')*

2. *C('candidate', 'evaluator', 'makeOffer', 'makeOffer · (responseYes V responseNo)')*

3. *C('evaluator', 'candidate', 'responseNo', 'responseNo · offerRejected')*

4. *C('hirer', 'candidate', 'responseYes · offerAccepted', 'responseYes · offerAccepted . positionFilled')*

Procediamo ad esaminare i commitment ottenuti. Ricordiamo che questo è un processo di business in cui vogliamo quindi vincolare le attività dell'antecedente al conseguente, per questo usiamo la tecnica che consiste nel ripetere l'antecedente nel conseguente.
Nell'articolo [12] troviamo una lista di commitment che descrivono il processo in questione, paragonandoli a quelli ottenuti dall'algoritmo, vedremo che sono molto simili e descrivono quasi identicamente il processo.
Il commitment numero 1 vincola l'evaluator a intervistare un eventuale candidato propostosi, tramite l'attività di *screenInterview* e successivamente a prendere una decisione, cioè *(makeOffer V rejectionNotice)*. Possiamo osservare che dentro al commitment è presente anche *apply*, un'attività del candidato; ciò è assolutamente normale, in questo modo in un certo senso si vincolano tutti e tre gli agenti. Inoltre, la presenza della congiunzione logica *OR* impone la mutua esclusione desiderata tra *makeOffer* e *rejectionNotice*.
Il secondo commitment obbliga il candidato a rispondere all'offerta del valutatore. Le due risposte sono mutuamente esclusive, non può mai accadere che il candidato risponda entrambe o che non risponda.
Il commitment numero 3 vincola l'annuncio del valutatore del rifiuto della offerta in seguito all'eventuale *responseNo* da parte del candidato.
Se invece il candidato accettasse l'offerta, avverrebbe la comunicazione dell'accettazione da parte del valutatore tramite *offerAccepted* e, grazie al commitment 4, il datore di lavoro sarebbe vincolato all'esecuzione dell'azione di *positionFilled* che comunica la fine della ricerca dell'impiegato così da non accettare ulteriori applicazioni di candidati o interviste da parte di valutatori.

Osserviamo i valori di tutte le dipendenze causali sotto riportate e successivamente discutiamo la scelta dei valori degli iper-parametri.

['postJob', 'hirer', 'apply', 'candidate', 0.9933333333333333]
['apply', 'candidate', 'screenInterview', 'evaluator', 0.9933333333333333]
['screenInterview', 'evaluator', 'makeOffer', 'evaluator', 0.984375]
['screenInterview', 'evaluator', 'rejectionNotice', 'evaluator', 0.9886363636363636]
['makeOffer', 'evaluator', 'responseNo', 'candidate', 0.972972972972973]
['makeOffer', 'evaluator', 'responseYes', 'candidate', 0.9615384615384616]
['makeOffer', 'evaluator', 'positionFilled', 'hirer', 0.6666666666666666]
['responseNo', 'candidate', 'offerRejected', 'evaluator', 0.9705882352941176]
['responseNo', 'candidate', 'positionFilled', 'hirer', 0.75]
['offerRejected', 'evaluator', 'positionFilled', 'hirer', 0.9166666666666666]
['responseYes', 'candidate', 'offerAccepted', 'evaluator', 0.9565217391304348]
['responseYes', 'candidate', 'positionFilled', 'hirer', 0.75]
['offerAccepted', 'evaluator', 'positionFilled', 'hirer', 0.9565217391304348]
['positionFilled', 'hirer', 'positionClosed', 'evaluator', 0.9655172413793104]
['rejectionNotice', 'evaluator', 'positionFilled', 'hirer', 0.96875]

In questa lista troviamo tutti i valori delle dipendenze tra le attività con associata la risorsa che le esegue, più precisamente con lo schema:
['Attività1, 'Risorsa1, 'Attività2, 'Risorsa2', Valore della dipendenza]

La scelta degli iper-parametri è ricaduta su:

- Valore minimo di dipendenza per la creazione di un commitment: **0.90**.
  In questo caso il valore è elevato perché le dipendenze sono, in generale, molto forti e con un valore più basso si sarebbero potuti generare commitment indesiderati.
- Valore di soglia per considerare un'attività finale: **5%.**
  In questo caso, la verifica sulle attività finali risulta fondamentale siccome, guardando i valori delle dipendenze causali, possiamo notare che, per esempio, esiste una forte dipendenza causale tra *rejectionNotice* e *positionFilled* ma questo non dovrebbe costituire un commitment dal momento che, facendo i calcoli, *rejectionNotice* risulta un'attività finale nel 33% dei casi.
- Lunghezza massima dei concatenamenti: **5**.
  Impostando questo valore a 0 otterremmo una lista di commitment finali senza concatenazioni, per la precisione:
  1. *C('candidate', 'hirer', 'postJob', 'postJob . apply')*
  2. *C('evaluator', 'candidate', 'apply', 'apply . screenInterview')*
  3. *C('evaluator', 'candidate', 'responseNo', 'responseNo . offerRejected')*
  4. *C('evaluator', 'candidate', 'responseYes', 'responseYes . offerAccepted')*
  5. *C('hirer', 'evaluator', 'offerAccepted', 'offerAccepted . positionFilled')*

6. *C('candidate', 'evaluator', 'makeOffer', 'makeOffer . (responseYes V responseNo)')*

Se notiamo, si perdono alcuni vincoli importanti dal momento che un commitment non può esistere se un solo agente viene coinvolto; per esempio il vincolo tra *screenInterview* e *(makeOffer V rejectionNotice)* non sussiste più siccome sono tutte azioni eseguite dal valutatore.

6. CONCLUSIONI E SVILUPPI FUTURI

In questo documento ho presentato un algoritmo per il mining dei commitment a partire dai log. Per ottenere una buona conoscenza della materia, ho studiato le peculiarità dei commitment in dettaglio, grazie ai vari articoli scritti a riguardo.
Per ottenere il risultato finale ho sfruttato tecniche di process mining che permettono di analizzare i log, con l'obiettivo di servirmi di risultati intermedi prodotti da strumenti di process mining all'avanguardia.
La materia del process mining ha un'importanza notevole al giorno d'oggi, dove cresce la necessità di analizzare i dati prodotti dai sistemi.
Dal lavoro eseguito è emerso che è possibile creare commitment dall'analisi dei log e l'utilizzo di veri e propri log per il case study ha permesso di dimostrare l'effettivo funzionamento dell'algoritmo creato. L'utilizzo di strumenti di process mining, come proM e PM4Py, è risultato fondamentale sia per la ricerca che per lo sviluppo. L'obiettivo principale della tesi era quello di trovare una metodologia che descrivesse come ricavare i commitment da un log, e si può dire che il fine è stato raggiunto.
Tutto ciò non significa che questo è un punto di arrivo, anzi, le ricerche in questa direzione devono sicuramente proseguire.
Alcune implementazioni future potrebbero essere:

- migliorare l'implementazione proposta aumentando la precisione dei criteri di selezione
- generalizzare l'algoritmo cosicchè sia possibile riconoscere commitment in qualsiasi tipologia di processi, non solo di business
- aggiungere un criterio di valutazione sistematica della bontà dei commitment generati, così da sapere quali commitment sono più forti e significativi, e quali meno
- implementare un metodo che, sfruttando i commitment, valuti la compliance dell'esecuzione dei processi rispetto al modello degli stessi; in altre parole determinare quanto differisce ciascuna esecuzione dal modello corrispondente, così da riuscire ad isolare le esecuzioni che potrebbero contenere errori

Le ricerche eseguite mi hanno permesso di scoprire un campo che prima avevo solamente sentito nominare e che ritengo abbia molte potenzialità, in aggiunta ho avuto modo di scoprire molti standard e strumenti nuovi per cui reputo lo stage effettuato valido e utile.

7. RINGRAZIAMENTI

Per concludere l’elaborato vorrei ringraziare tutte le persone che hanno contribuito a rendere possibile la sua realizzazione.

In primo luogo, un doveroso ringraziamento va al mio relatore, il professore Roberto Micalizio per avermi seguito passo a passo nella fase di ricerca e nella fase di stesura, offrendo sempre la sua disponibilità.

Vorrei inoltre ringraziare la mia famiglia per tutto il supporto offerto, nello specifico mio padre Tiziano, mia sorella Cristina e tutti i miei nonni.

La dedica va a te, mamma, so che ne saresti orgogliosa.

8. BIBLIOGRAFIA E SITOGRAFIA

1. Van der Aalst Wil (2016) *Process Mining: Data Science in Action*, 2nd edition, Springer-Verlag Berlin Heidelberg.
2. Van der Aalst Wil (2011) *Process Mining: Discovery, Conformance and Enhancement of Business Processes, Springer-Verlag Berlin Heidelberg*
3. http://www.processmining.org/prom/start
4. http://xes-standard.org
5. Van der Aalst Wil, Reijers Hajo A., Song M. (2005) *Discovering Social Networks from Event Logs,* Computer Supported Coop Work 14, 549–593
6. https://pm4py.fit.fraunhofer.de
7. https://www.fit.fraunhofer.de/en.html
8. Castelfranchi C. (1995) *Commitments: From individual intentions to groups and organizations*, Proceedings of the First International Conference on Multiagent Systems, AAAI (www.aaai.org).
9. Munindar P. Singh (1999) *An Ontology for Commitments in Multiagent Systems*, Artificial Intelligence and Law 7, 97–113
10. https://pandas.pydata.org
11. https://graphviz.org
12. Baldoni M., Baroglio C., Capuzzimati F., Micalizio R. (2019) *Process Coordination with Business Artifacts and Multiagent Technologies*, J Data Semant 8, 99–112
13. Kafalı Ö., Torroni P. (2012) *Exception diagnosis in multiagent contract executions*, Ann Math Artif Intell 64, 73–107