Alma Mater Studiorum · Università di Bologna

Dipartimento di Informatica - Scienza e Ingegneria

Corso di Laurea Magistrale in Ingegneria Informatica

UN ALGORITMO PER LA DETERMINAZIONE DI MODELLI DI PROCESSO DCR A PARTIRE DA ESEMPI POSITIVI E NEGATIVI

Relatore:
Prof. Federico Chesani

Presentata da:
Manuel Raneri

Correlatore:
Prof. Paola Mello

Sessione autunnale
Anno Accademico 2020/2021

Sommario

Al giorno d'oggi le aziende si trovano a dover fronteggiare la complessità dei processi di business e l'ottimizzazione degli stessi. Per ridurre tale complessità si sta investendo in un approccio che permetta di facilitare la gestione di un processo ed evitare di riscontrare problemi dal punto di vista gestionale e organizzativo all'interno dell'impresa.

Il processo di business è stato definito come un insieme di attività tese al raggiungimento di un obiettivo aziendale. Il processo deve prevedere input chiaramente ben definiti e un risultato finale. Gli input si basano sull'utilizzo dei modelli di processo che descrivono le attività svolte all'interno di un processo, sia per quanto riguarda i metodi di svolgimento, sia gli strumenti utilizzati. Alcune aziende hanno aderito a tali modelli rendendosi conto della facilità di utilizzo e dell'immediata leggibilità, in quanto utilizzano una notazione grafica molto semplice e comprensibile da tutti, anche da persone con un limitato background aziendale. 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. Grazie ai log è possibile ricostruire tutte le operazioni che sono state effettuate, quando e da chi. Questo garantisce una grande facilità nella misurazione ed un continuo monitoraggio delle attività coinvolte nel processo, permettendo, così, di individuare eventuali punti di dispersione, sia di tempo che di risorse, all'interno del processo. In questo modo sarà possibile individuare eventuali miglioramenti da applicare per rendere più performante il processo in modo tale da portare benefici sia all'azienda che agli attori coinvolti. Il controllo ed il monitoraggio di tali processi viene realizzato dal *Process Mining*, una disciplina di ricerca relativamente giovane che si colloca tra il data mining e la modellazione/analisi dei processi. L'idea

del Process Mining è analizzare i processi per come si svolgono nella realtà scoprendo i punti di forza e di debolezza e quindi migliorarli. Questo è possibile analizzando i dati che sono presenti nei log di eventi prontamente disponibili nei sistemi (informativi) odierni. La maggior parte degli algoritmi di discovery esistenti usa solo log composti da tracce positive, ovvero, una sequenza di eventi che rappresentano il processo dall'inizio alla fine. In questo elaborato, invece, saranno utilizzati dei log che contengono sia tracce positive che negative. Le tracce negative rappresentano una sequenza di azioni dal comportamento indesiderato.
L'obiettivo di questo progetto è creare un algoritmo, implementato attraverso il linguaggio di modellazione DCR, che sia capace di ottenere un modello di processo, costituito da vincoli DCR, estraendo informazioni da un log di eventi che rappresentano un particolare processo di business. Questo algoritmo prende inspirazione dall'articolo pubblicato da Mooney *Encouraging experimental results on learning CNF* [1] e avrà due differenti versioni.

Il lavoro è strutturato in sei capitoli. Il capitolo 1 fornisce un'introduzione sulle principali tecnologie che verranno utilizzate per raggiungere l'obiettivo. Nello specifico, verranno spiegati cosa sono i modelli di processo, i log e il process mining con tutte le sue possibili applicazioni, tecniche e strumenti. Il capitolo 2 è principalmente dedicato allo stato dell'arte di DCR, ai benefici dell'utilizzo di un approccio dichiarativo invece di uno imperativo; inoltre, verranno mostrati alcuni esempi di casi di studio reali e alcuni algoritmi di mining implementati in DCR, ma esclusivamente su tracce positive. Il capitolo 3 spiega l'algoritmo ideato da Mooney, nelle due versioni, e come tutto ciò è stato adattato e implementato nel mondo DCR. Saranno spiegate le principali funzioni implementate e alcuni esempi per chiarire meglio i concetti introdotti. Una parte di questo capitolo è dedicata al confronto tra l'algoritmo che lavora su Declare, quello che lavora su DCR e su alcuni algoritmi di discovery, imple-

mentati in DCR, che sono stati sviluppati per lavorare solo su tracce positive. Nel capitolo 4 verranno mostrati i risultati dei test che sono stati effettuati su tre differenti tipi di log di eventi in modo da monitorare le prestazioni dell'algoritmo al variare del numero di tracce del log. Il capitolo 5 mostra una possibile estensione dell'algoritmo di Mooney, implementata solo sulla versione DNF. Si vedranno, inoltre, alcuni casi di studio e un esempio che mostri come funziona l'algoritmo e il confronto con il modello di Mooney iniziale. Il sesto e ultimo capitolo descrive possibili lavori futuri per migliorare ed estendere il lavoro svolto fino a questo momento.

Indice

Elenco delle figure

1 Introduzione

1.1 Modelli di processo

Un modello di processo è una rappresentazione formale di un processo in cui vengono rappresentate le singole attività che lo compongono. Definire un processo significa specificare le attività, le relative procedure, gli strumenti che dovranno essere utilizzati, i documenti che verranno prodotti e modificati nel corso del processo.
I modelli di processo [2] sono importanti perché illustrano e documentano i processi. Possono anche essere utilizzati per formare nuovi dipendenti oppure per eseguire le analisi e le ottimizzazioni del processo. Essi vengono utilizzati anche per il controllo del processo, ad esempio per verificare se ne è stata rispettata la conformità. Tra le notazioni per i modelli di business le più diffuse attualmente sono la Business Process Model and Notation (BPMN), Petri nets (adottano un approccio procedurale), Dynamic Condition Response graphs (DCR) e Declare (adottano un approccio dichiarativo). Questo studio si concentra, in particolar modo, sul formalismo DCR.
La Fig. 1 mostra un esempio di un modello di processo BPMN composto solo da quattro attività. Il processo inizia con una persona che ha fame. Il passo successivo prevede il controllo del frigo, dove sono presenti due alternative: se il frigo è pieno la scelta ricade sul mangiare a casa, viceversa, la scelta sarà mangiare al ristorante. Il processo si conclude quando la persona risulta sazia.

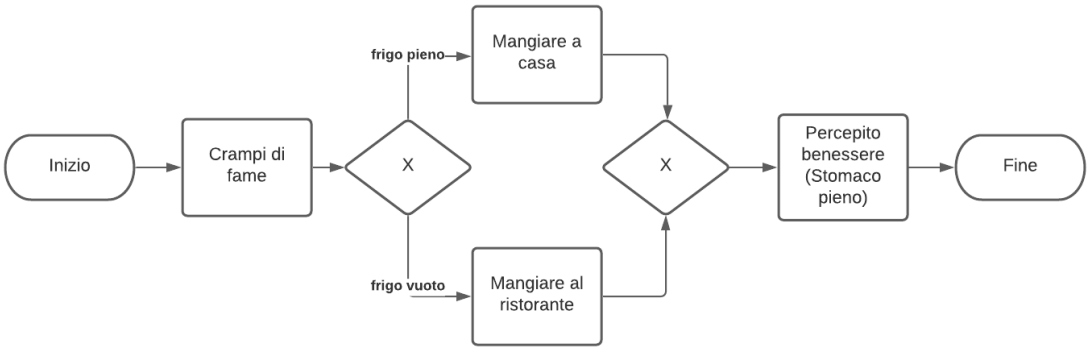


Figura 1: Modello di processo BPMN.

1.2 Log

Il log, in ambito informatico, è un file in cui vengono registrate le operazioni svolte dall'utente all'interno di un contesto con lo scopo di:

- Recuperare le informazioni precedenti in seguito ad un errore del sistema.
- Investigare su attività malevole.
- Analizzare le operazioni effettuate.
- Analizzare le modifiche su basi di dati.
- Verificare la presenza di errori.
- Estrapolare informazioni in un determinato arco di tempo.

Esistono diversi tipi di log [3] che differiscono dall'ambito in cui vengono utilizzati:

- **Log di registrazione, riconoscimento e accesso**: è il noto login o logon per l'accesso di un utente al sistema per l'esecuzione dei relativi servizi. I dati di accesso memorizzati sono registrati su appositi file per l'eventuale analisi. Anche l'uscita (o logout) può essere memorizzata.

- **Log di sistema**: usati frequentemente nei server, essi registrano gli eventi riguardanti il flusso di dati tra il sistema, come fornitori di servizi, e i clienti di tali servizi.

- **Log di base di dati**: in questo caso è il sistema gestore di base di dati (DBMS) che registra le operazioni fatte sulla base di dati: inserimento, aggiornamento, cancellazione di record. Nei DBMS evoluti, che forniscono servizi di tipo transazionale, il log è anche la base di riferimento per eseguire le funzioni di transazione completa (commit) o transazione annullata (rollback).

- **Log di sicurezza**: tipico dei sistemi informatici complessi o destinati ad ospitare dati particolarmente sensibili, memorizza tutte le operazioni che sono considerate critiche per l'integrità dei dati e per il sistema, nonché il controllo dei tentativi di accesso al sistema (autorizzati e non).

- **Log di applicazione**: molte applicazioni prevedono log propri su cui sono registrati eventi caratteristici dell'applicazione e che fungono, in certi casi, da veri e propri protocolli di entrata e di uscita. Ad esempio, in programmazione, il file di log evidenzia il tipo di errore e il punto in cui si trova nel codice, grazie al messaggio inviato dal canale di standard error (es. eccezioni).

- **Log di eventi (event log)**: in questa tipologia di log vengono registrate tutte le attività svolte all'interno di un processo aziendale. Questo è il tipo di log che verrà utilizzato all'interno di questo elaborato.

1.2.1 Log di eventi

I log di eventi sono dei particolari tipi di log che contengono informazioni relative a eventi che si sono verificati all'interno di un sistema (ad esempio chi ha eseguito una determinata attività, il timestamp di tale attività ecc.) e vengono utilizzati per

svolgere alcune funzioni all'interno di un'organizzazione, quali ottimizzazione delle prestazioni dei sistemi e investigazione sulle attività malevoli.

I log [4] possono essere memorizzati nelle tabelle dei database, nei registri dei messaggi o delle transazioni e negli archivi di posta. Una cosa molto importante da gestire è la qualità dei log di eventi. L'obiettivo è, infatti, quello di avere dei log che siano affidabili, cioè che memorizzino eventi realmente accaduti e i cui attributi siano corretti. Inoltre, essi devono essere completi, ovvero, dato un certo contesto, devono contenere tutti gli eventi inerenti ad esso. Un log di eventi è composto da una serie di record che vengono registrati in maniera sequenziale. Nella Fig. 2 [5] è possibile notare un esempio di un log di eventi.

Case ID	Task Name	Event Type	Originator	Timestamp
1	File Fine	Completed	Raja	22-08-10 14:00:00
2	File Fine	Completed	Raja	22-08-10 15:00:00
1	Send Bill	Completed	System	22-08-10 15:05:00
2	Send Bill	Completed	System	22-08-10 15:07:00
3	File Fine	Completed	Raja	23-08-10 10:00:00
3	Send Bill	Completed	System	23-08-10 14:00:00
4	File Fine	Completed	Raja	24-08-10 11:00:00
4	Send Bill	Completed	System	24-08-10 11:10:00
1	Process Payment	Completed	System	26-08-10 15:05:00
1	Close Case	Completed	System	26-08-10 15:06:00
2	Sent Reminder	Completed	Saran	22-09-10 10:00:00
3	Send Reminder	Completed	John	23-09-10 10:00:00
2	Process Payment	Completed	System	24-09-10 09:05:00
2	Close Case	Completed	System	24-09-10 09:06:00
4	Send Reminder	Completed	John	24-09-10 15:10:00
4	Send Reminder	Completed	Saran	24-09-10 17:10:00
4	Process Payment	Completed	System	30-09-10 14:01:00
4	Close Case	Completed	System	30-09-10 17:30:00
3	Send Reminder	Completed	John	21-10-10 10:00:00
3	Send Reminder	Completed	John	21-11-10 10:00:00
3	Process Payment	Completed	System	24-11-10 14:00:00
3	Close Case	Completed	System	24-11-10 14:01:00

Figura 2: Esempio di Log di Eventi [5].

1.3 Process Mining

1.3.1 Definizione

Il Process Mining [4] è una disciplina di ricerca relativamente giovane che si trova, da un lato, tra la computational intelligence e il data mining e, dall'altro, tra la modellazione e l'analisi dei processi. L'idea del Process Mining è scoprire, monitorare e migliorare i processi reali estraendo la conoscenza dai log di eventi disponibili nei sistemi informativi odierni. Il Process Mining fa un uso costante dei log di eventi durante le varie fasi. Esso include:

- Il rilevamento automatico dei processi, quindi l'estrazione di modelli di processo da un log (discovery).
- Il controllo della conformità (ovvero il monitoraggio delle discrepanze tra un modello e un log).
- L'individuazione di reti sociali (social network) e organizzative.
- La costruzione automatica di modelli di simulazione.
- L'estensione e la verifica dei modelli.
- La previsione delle possibili evoluzioni future di un'istanza di processo.
- L'attenzione sull'utilizzo dei dati storici.

In questi ultimi anni la tecnologia si è evoluta, favorendo l'allineamento tra l'universo digitale, che comprende tutti i dati archiviati e scambiati elettronicamente, e l'universo reale, che comprende quell'insieme di eventi che si verificano quotidianamente nei vari contesti.
Lo scopo del Process Mining è partire da questi dati (contenuti all'interno del log di eventi), studiarli e attuare delle strategie per migliorare e semplificare i processi. I log

di eventi (event log o semplicemente log) possono contenere anche altre informazioni, oltre agli eventi, come le persone o le risorse che eseguono o iniziano un'attività, i timestamp per determinare il sequenziamento e quindi l'ordine temporale delle attività o altri dati specifici di un evento. Per contestualizzare il Process Mining, utilizziamo il ciclo di vita del Business Process Management (BPM) mostrato in Fig. 3 [4].
Il ciclo di vita del BPM mostra sette fasi di un processo di business e il sistema di informazioni corrispondente. Nella fase di (ri)modellazione viene creato un nuovo modello di processo o viene adattato un modello di processo esistente. Nella fase di analisi vengono confrontati un modello candidato e una possibile alternativa. Conclusa la fase di (ri)modellazione, si implementa il modello (fase di implementazione) o si (ri)configura un sistema informativo esistente (fase di (ri)configurazione). Nella fase di esecuzione viene messo in atto il modello progettato e viene monitorato il processo. Inoltre, possono essere effettuati piccoli aggiustamenti senza riprogettare il processo (fase di adattamento). Nella fase di diagnosi viene analizzato il processo in esecuzione e l'output di questa fase può innescare una nuova fase di riprogettazione del processo.

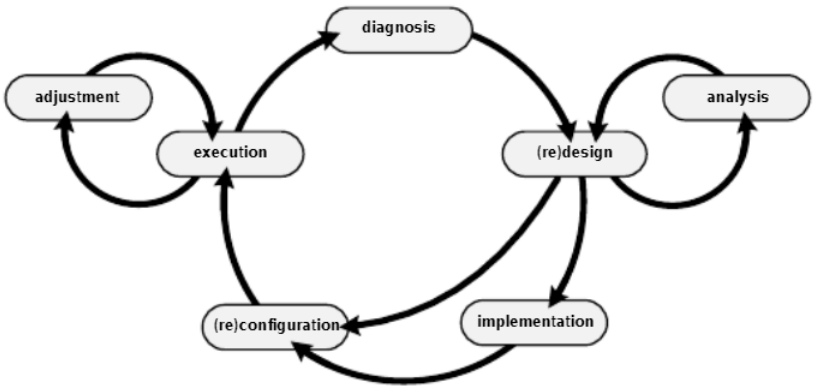


Figura 3: Il Ciclo di vita BPM [4].

Il Process Mining è uno strumento prezioso per la maggior parte delle fasi descritte

precedentemente. Ad esempio, nella fase di esecuzione o nella fase di diagnosi, le tecniche di Process Mining possono supportare le decisioni, quindi aiutare ad adattare il processo e guidare la rimodellazione dello stesso.
La Fig. 4 mostra le attività e i risultati del Process Mining, sono, infatti, riportate le possibili fasi attraversate durante un progetto di Process Mining:

- **Fase 0**: ogni progetto inizia con una identificazione dei bisogni che conducano alla sua apertura e alla successiva pianificazione delle varie attività.

- **Fase 1**: avviato il progetto, è necessario interrogare sistemi informativi, manager, stakeholders in generale, in modo da ottenere dati degli eventi, modelli, identificare obiettivi (KPI) e prendere nota delle domande a cui, successivamente, si potrà rispondere.

- **Fase 2**: si costruisce il modello del flusso di controllo e lo si collega al log. In questa fase si possono utilizzare tecniche di discovery automatico. Il modello di processo ottenuto potrebbe già rispondere ad alcune domande e potrebbe portare ad adattamenti e rimodellazioni. Inoltre, il log può essere modificato, rimuovendo o aggiungendo attività, per adattarlo in base al modello. Gli eventi rimanenti sono connessi all'entità del modello di processo.

- **Fase 3**: quando il processo raggiunge un'adeguata struttura, il modello del flusso di controllo può essere esteso con altre prospettive, ad esempio dati, tempo, risorse. Tutto ciò è possibile grazie al collegamento tra log e modello, realizzato nella fase precendente.

- **Fase 4**: i modelli ottenuti nella fase 3 possono essere utilizzati per il supporto operazionale. La conoscenza estratta, a partire dai dati storici sugli eventi, è combinata con informazioni riguardanti le istanze in esecuzione. In questo modo è possibile modificare, predire e consigliare.

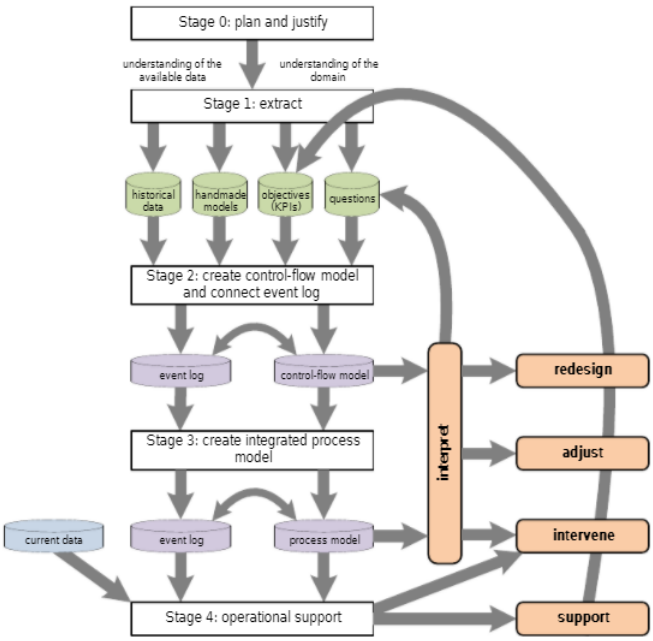


Figura 4: Il Ciclo di vita di un modello che descrive un progetto di Process Mining [4].

È opportuno evidenziare che le fasi 3 e 4 possono essere raggiunte solo se il processo risulta sufficientemente stabile e strutturato. Attualmente, ci sono tecniche e strumenti che possono supportare tutte le fasi mostrate nella Fig. 4. Tuttavia, il Process Mining è un paradigma relativamente nuovo e la maggior parte degli strumenti attualmente disponibili sono ancora immaturi; inoltre, gli utenti spesso non sono consapevoli del potenziale e dei limiti del Process Mining.

1.3.2 Tecniche di Process Mining

Come mostrato in Fig. 5, i log di eventi possono essere utilizzati per condurre tre tipi di Process Mining [4]:

1. **Discovery**: è la tecnica più importante del Process Mining. La tecnica di discovery prende un log di eventi e produce un modello senza utilizzare alcuna informazione a priori, quindi riesce a descrivere i processi reali basandosi, semplicemente, su esempi di esecuzione nei log.

2. **Conformance Checking**: un modello di processo esistente viene confrontato con un log di eventi dello stesso processo. Il conformance checking può essere utilizzato per verificare se ciò che accade nella realtà, così come risulta nel log, è conforme al modello e viceversa. Tale tecnica può essere applicata a tipi differenti di modelli: modelli procedurali, modelli organizzativi, modelli dichiarativi, regole di business, leggi, ecc.

3. **Enhancement**: consiste nel migliorare o estendere un modello di processo esistente utilizzando le informazioni relative al processo, contenute all'interno dei log. A differenza del conformance checking, che misura quanto un modello sia allineato con la realtà, l'enhancement propone di modificare o estendere il modello già esistente per apportare miglioramenti. Ad esempio, usando i timestamp in un log eventi possiamo identificare colli di bottiglia, livelli di servizio, tempi di produttività e frequenze.

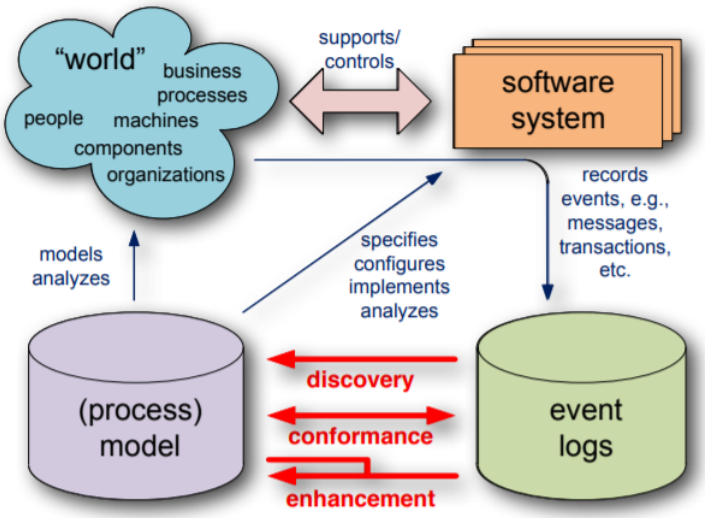


Figura 5: I principali tipi di Process Mining [4].

La Fig. 6 mostra i tipi di input e output delle tre tecniche di Process Mining. La tecnica di discovery prende in input un log di eventi e restituisce in output un modello, che solitamente è un modello di processo (per esempio una rete di Pedri, un modello BPMN, un grafo DCR). Tuttavia, il modello può descrivere anche altre prospettive, come ad esempio un social network. La tecnica di conformance checking prende in input un log di eventi e un modello e restituisce in output una serie di informazioni diagnostiche che mostrano come il log e il modello differiscono. Anche la tecnica di enhancement richiede un log e un modello in ingresso, ma in uscita restituisce un nuovo modello che è un'estensione del modello di input (esteso con informazioni riguardanti, ad esempio, il tempo delle attività).

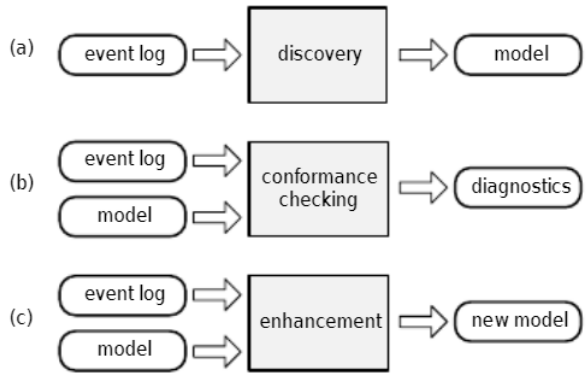


Figura 6: Input ed output delle tre tecniche di Process Mining [4].

1.3.3 Prospettive del Process Mining

Il Process Mining copre varie prospettive [4]:

- **Prospettiva del flusso di controllo**: si basa sul flusso di controllo, cioè ci si concentra sull'ordine delle attività. Lo scopo è trovare una buona caratterizzazione di tutti i possibili percorsi di un processo. L'output è espresso in termini di un qualunque modello di processo.

- **Prospettiva dell'organizzazione**: si occupa di individuare quelle risorse (per esempio persone, ruoli, dipartimenti e sistemi) contenute nei log, spesso non visibili, e come questi sono in relazione tra di loro. Lo scopo è quello di strutturare l'organizzazione classificando le persone in base al ruolo, oppure di creare una rappresentazione del social network dell'organizzazione.

- **Prospettiva della traccia**: si concentra sulle proprietà di una traccia (detta anche case). Una traccia può essere caratterizzata dal suo percorso nel processo oppure dagli attori che ci lavorano. Inoltre, possono dipendere anche dai valori degli elementi di dati corrispondenti. Ad esempio, se una traccia rappresenta

un ordine, potrebbe essere utile individuare l'azienda fornitrice, il numero di prodotti ordinati o il prezzo.

- **Prospettiva del tempo**: viene utilizzata per individuare quando un evento si è verificato e la sua frequenza di esecuzione. Se ad un evento è associato un timestamp, è possibile scoprire colli di bottiglia, misurare i livelli di un servizio, monitorare l'uso delle risorse e prevedere il tempo restante per il completamento di un case.

2 Approcci dichiarativi per modelli di processo

Le tecnologie di gestione dei processi aziendali (BPM) [6] supportano la digitalizzazione dei flussi di lavoro dei processi aziendali utilizzando modelli di processo espliciti (imperativi), seguendo diverse fasi: progettazione, convalida, esecuzione del processo e monitoraggio. Tali modelli di processo vengono identificati attraverso gli algoritmi di Process Mining. La maggior parte delle tecnologie BPM descrivono i processi attraverso dei diagrammi di flusso imperativi come BPMN (Business Process Model and Notation). I processi non seguono uno schema preciso, infatti, il più delle volte, subiscono delle variazioni, di conseguenza, mappare tali azioni in un diagramma di flusso risulta troppo rigido e complesso. Inoltre, i diagrammi di flusso descrivono solo come eseguire un processo, non occupandosi di aspetti, ad esempio, legali.

Consideriamo, ad esempio, un flusso di lavoro ospedaliero tratto da uno studio di vita reale negli ospedali danesi [7]. Inizialmente definiamo due eventi, *Prescribe* e *Sign*, che corrispondono rispettivamente alla prescrizione di una medicina da parte di un medico e la firma su tale prescrizione.
Una prima descrizione del processo imperativo è mostrata in Fig. 7(a) in cui vengono messe in sequenza le due azioni: in tale scenario il medico deve prescrivere un medicinale e successivamente apporre una firma sulla prescrizione. Tale soluzione non permette di aggiungere più prescrizioni prima o dopo la firma e di firmare più volte, nonostante sia lecito secondo i vincoli.
Una seconda descrizione è mostrata in Fig. 7(b) in cui l'evento *Prescribe* è seguito da cicli che consentono di prescrivere e firmare più prescrizioni. Il flusso di lavoro può terminare solo dopo l'evento *Sign*. Se l'esecuzione continua per sempre, è necessario imporre che ogni prescrizione sia alla fine seguita da un evento di firma.
Una terza descrizione, mostrata in Fig. 7(c), aggiunge un ulteriore evento (*Give*) che permette all'infermiera la somministrazione del medicinale al paziente, solo dopo che

il medico abbia firmato la prescrizione. L'evento *Give* viene aggiunto dopo l'evento di firma, all'interno del ciclo, modificando la terminazione del flusso di lavoro, solo dopo l'esecuzione di quest'ultimo. Tale esempio mostra come sia difficile gestire qualsiasi esecuzione di un flusso di lavoro che soddisfi i requisiti senza fare riferimento ad un caso specifico.

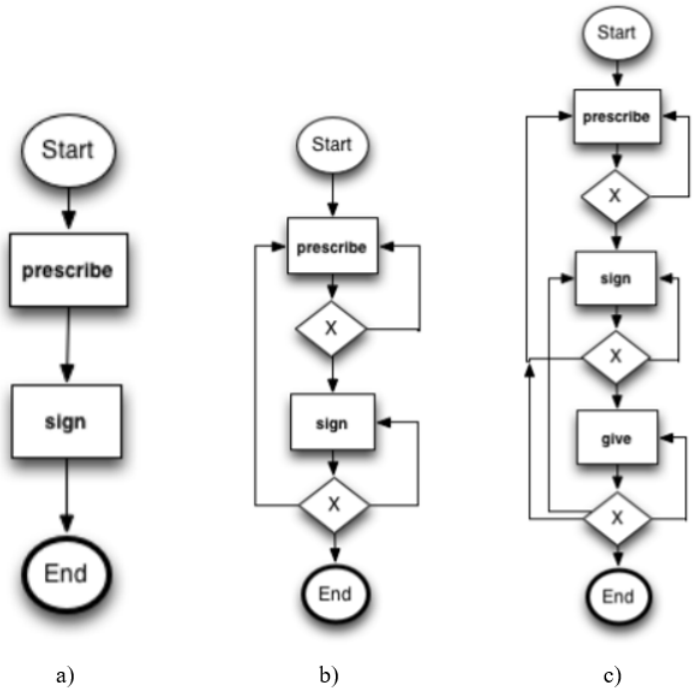


Figura 7: Esempio dei tre flussi di lavoro imperativo [7].

I linguaggi imperativi determinano i modelli dall'interno verso l'esterno, specificando il flusso di controllo che si vuole avere nel processo, concentrandosi, in particolar modo, nel soddisfare i requisiti. Al contrario, i modelli dichiarativi si concentrano sulla specifica di ciò che dovrebbe essere soddisfatto, offrendo tutti i comportamenti possibili e usando i vincoli per eliminare il comportamento non desiderato nel processo, come mostrato in Fig. 8.

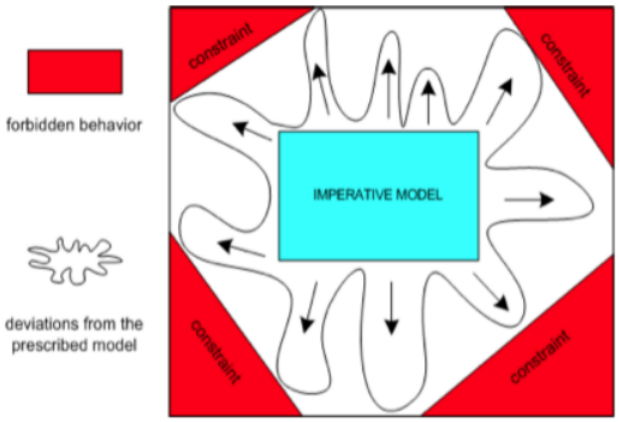


Figura 8: Approccio imperativo e dichiarativo a confronto [7].

Questo è il motivo per cui, in genere, si utilizzano i diagrammi di flusso solo per i processi di routine o per descrivere pratiche standard. Si sceglie, quindi, di adottare le notazioni dichiarative (implicite) come output del Process Mining e per il supporto del processo in fase di esecuzione. Le tecniche di implementazione, per la maggior parte dei modelli di processo dichiarativi, come Declare o DecSerFlow, si basano sulla conversione dei vincoli dichiarativi in un modello imperativo (ad esempio un automa) per consentirne l'esecuzione. Tale conversione comporta una maggiore difficoltà nell'adattare i vincoli durante la fase esecutiva, in quanto l'automa deve essere ricalcolato ogni volta che cambiano i vincoli.
Un'eccezione sono i grafi DCR (Dynamic Condition Response), infatti, creando direttamente il grafo di transizione e gestendo le modifiche dei vincoli durante la fase di esecuzione, non necessitano della traduzione in un modello imperativo. Tuttavia, uno svantaggio dell'approccio dichiarativo riguarda il fatto che la definizione implicita del flusso di controllo lo rende meno facile da percepire per l'utente o da calcolare dal motore di esecuzione. In ogni stato, si deve risolvere l'insieme di vincoli per capire quali sono i prossimi eventi possibili, pertanto, diventa ancora più difficile avere una

panoramica dell'intera esecuzione del processo.
La ricerca sulla modellazione dichiarativa ha suscitato un crescente interesse negli ultimi anni, infatti, sono stati sviluppati diversi linguaggi dichiarativi come Declare, DCR Graphs e SCIFF (Social Constrained IFF).

2.1 DCR: Dynamic Condition Response

DCR (Dynamic Condition Response) si rivela lo strumento più adatto per dare una visione del flusso di lavoro in cui l'ordine delle attività può variare notevolmente, come avviene nella realtà. Ad esempio, se la legislazione cambia, in poco tempo il cambiamento si propaga nel nostro sistema IT.
La caratteristica unica di DCR è che la legislazione e i requisiti possono essere digitalizzati direttamente e mantenuti localmente da esperti del settore utilizzando strumenti di progettazione e simulazione, evitando, dunque, costosi ritardi o errori di aggiornamento da parte degli sviluppatori IT. Esistono soluzioni software che digitalizzano i processi, ma tali applicazioni, molto spesso, comportano una maggiore complessità nel processo e nel lavoro dell'utente finale. Invece, con DCR la digitalizzazione diventa un processo facile, veloce ed efficiente.

Nello specifico, i benefici relativi a DCR sono [8]:

- **Ottimizzare i processi di business**: le organizzazioni e le aziende si occupano dell'esecuzione di processi critici in cui, spesso, i cambiamenti immediati sono di difficile esecuzione. DCR analizza le regole aziendali per implementare e migliorare i processi all'istante, riducendo il rischio, i tempi e i costi di realizzazione normalmente presenti durante lo sviluppo di nuove tecnologie.

- **Aiutare i lavoratori**: grazie a DCR i lavoratori sono in grado di progettare, modificare, testare e convalidare i processi dinamici in poco tempo.

- **Aumentare la soddisfazione dei clienti**: i clienti possono migliorare la loro soluzione esistente.
- **Supportare i consulenti aziendali**: i consulenti aziendali e di processo possono collaborare con i clienti per modellare i processi di business e crearli in modo più veloce.

Un linguaggio di modellazione dei processi sono i grafi DCR.

2.2 Dalla struttura a eventi a DCR distribuito

In questa sezione verrà discusso come ottenere DCR distribuito, partendo da una struttura a eventi [9] come mostrato in Fig. 9 [7].

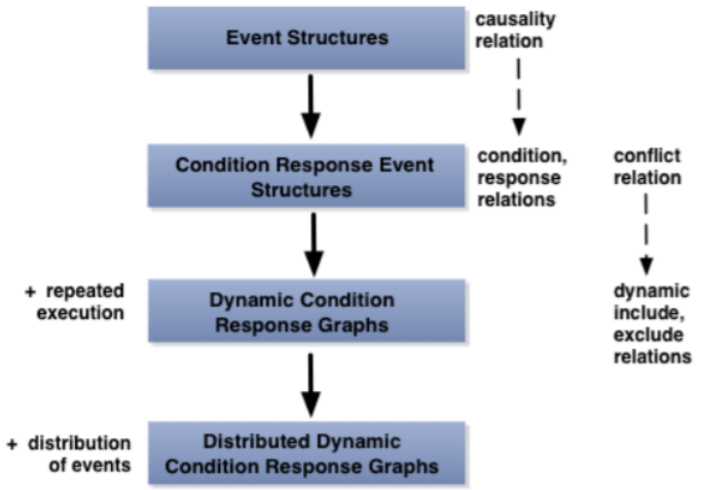


Figura 9: Panoramica da Struttura a Eventi ai grafi DCR [7].

2.2.1 Struttura ad eventi principali

Una struttura ad eventi principali (ES) è una quadrupla *(E, Act, ≤, #, l)* dove:

- E è un insieme (possibilmente infinito) di eventi.

- Act è un insieme di azioni .
- $\leq \subseteq E \times E$ è la relazione di causalità tra eventi il cui ordine è parziale.
- $\# \subseteq E \times E$ è una relazione di conflitto binaria tra eventi, simmetrica e non riflessiva.
- $l : E \to Act$ è la funzione di etichettatura che associa gli eventi alle azioni.

L'azione $a \in Act$ rappresenta l'azione che il sistema potrebbe eseguire, mentre un evento $e \in E$ etichettato con a rappresenta il verificarsi dell'azione a durante l'esecuzione del sistema. La relazione di causalità $e \leq e'$ significa che l'evento e deve essere eseguito prima dell'evento e' e la relazione di conflitto $e \# e'$ implica che il verificarsi di uno esclude il verificarsi dell'altro. Le relazioni di causalità e di conflitto soddisfano le condizioni:

- $e \# e' \leq e'' \Rightarrow e \# e''$, cioè se l'evento e', in conflitto con l'evento e, deve essere eseguito prima di e'', allora l'evento e è in conflitto con l'evento e''.
- l'insieme $\{e' \mid e' \leq e\}$ è finito per ogni $e \in E$.

Una configurazione c è un insieme di eventi tale che:

- Non ci sono conflitti tra eventi: $\forall\ e, e' \in c \Rightarrow \neg e \# e'$.
- Chiuso verso il basso: $\forall\ e \in c, e' \in E \Rightarrow e' \leq e \Rightarrow e' \in c$.

Come passo intermedio verso la struttura DCR, generalizziamo le strutture ad eventi principali con la struttura ad eventi delle risposte alle condizioni (CRES), sostituendo la relazione di causalità con due relazioni: la relazione di condizione e di risposta.

2.2.2 Struttura ad eventi delle risposte alle condizioni

Una struttura ad eventi delle risposte alle condizioni (CRES) è una tupla $D = (E, Act, Pe, \leq_C, \leq_R, \#, l)$ dove:

- E è un insieme (possibilmente infinito) di eventi.
- Act è un insieme di azioni.
- Pe è un insieme di risposte iniziali.
- $\leq_C \subseteq E \times E$ è la relazione di condizione tra eventi che è di ordine parziale.
- $\leq_R \subseteq E \times E$ è la relazione di risposta tra eventi, dove $\leq = \leq_C \cup \leq_R$ è di ordine parziale.
- $\# \subseteq E \times E$ è una relazione binaria di conflitto tra eventi, simmetrica e non riflessiva.
- $l : E \to Act$ è la funzione di etichettatura che associa gli eventi alle azioni.

La relazione di condizione impone una relazione di precedenza tra gli eventi, ad esempio, se due eventi sono correlati dalla relazione di condizione $e \leq_C e'$, l'evento e deve accadere prima che si verifichi l'evento e'.
Una relazione di risposta $e \leq_R e'$ impone che ogni volta che e viene eseguito, allora, ad un certo punto, dovrà verificarsi e'. Anche in CRES le relazioni di causalità e di conflitto, soddisfano le condizioni:

- $e \# e' \leq e'' \Rightarrow e \# e''$.
- L'insieme $\{e' \mid e' \leq e\}$ è finito per ogni $e \in E$.

Definiamo una configurazione come per ES, tranne per il fatto che una configurazione di un CRES deve essere chiusa verso il basso rispetto alla relazione di condizione, cioè una configurazione c di un CRES è un insieme di eventi tale che:

- Non ci sono conflitti tra eventi: $\forall\ e,\ e' \in c \Rightarrow \neg e\ \#\ e'$.
- Chiuso verso il basso: $\forall\ e \in c,\ e' \in E \Rightarrow e' \leq_C e \Rightarrow e' \in c$.

2.2.3 Struttura dei grafi DCR

Una struttura di risposta alle condizioni dinamiche (DCR) è una tupla $D = (E,\ Act,\ M,\ \rightarrow\bullet,\ \bullet\rightarrow,\ \rightarrow\Diamond,\ \pm,\ l)$ dove:

- E è un insieme di eventi.
- Act è un insieme di azioni.
- M è una tripla (Ex, Pe, In) di insiemi di eventi, dove Ex è l'insieme di attività atomiche che sono state eseguite almeno una volta durante l'esecuzione, Pe è l'insieme di attività atomiche che, se incluse, dovranno essere eseguite almeno un'altra volta in futuro come risultato di un vincolo di risposta (cioè sono risposte in sospeso), In indica le attività attualmente incluse.
- $\rightarrow\bullet \subseteq E \times E$ è la relazione di condizione.
- $\bullet\rightarrow \subseteq E \times E$ è la relazione di risposta.
- $\rightarrow\Diamond \subseteq E \times E$ è la relazione milestone.
- $\pm : E \times E \rightarrow \{+,\ \%,\ *\}$ è la relazione dinamica di inclusione/esclusione.
- $l : E \rightarrow Act$ è la funzione di etichettatura che associa gli eventi alle azioni.

Le relazioni di condizione e di risposta in DCR sono le stesse delle relazioni corrispondenti di CRES, tranne per il fatto che non sono vincolate in alcun modo, cioè è possibile avere relazioni cicliche.
I grafi di risposta alle condizioni dinamiche sono una rappresentazione dichiarativa di un modello che descrive contemporaneamente un processo e il suo stato di esecuzione. La notazione è composta da attività, stati di attività e cinque relazioni tra attività. Lo stato di attività può assumere uno tra tre possibili valori booleani per indicare se l'attività è stata eseguita, è inclusa, è in attesa di risposta (in sospeso). Se un'attività non è inclusa allora è momentaneamente assente dal flusso di lavoro, se è in sospeso allora deve essere eseguita o esclusa per permettere che il flusso di lavoro termini. Le relazioni tra le attività indicano se un'attività possa essere attualmente eseguita e come tale esecuzione modifichi lo stato di un'altra attività.
Una relazione di condizione A $\rightarrow\bullet$ B impone che l'attività B non possa essere eseguita se non dopo l'esecuzione dell'attività A. L'esecuzione di un'attività cancella il suo stato in sospeso e imposta il suo stato di esecuzione.
Una condizione di risposta A $\bullet\rightarrow$ B impone che ogni volta che l'attività A viene eseguita, allora, ad un certo punto, dovrà verificarsi l'attività B affinché il flusso venga accettato. Se l'attività A non venisse eseguita, la relazione sarebbe comunque soddisfatta senza effettuare il controllo sulla presenza o meno di B.
Un'inclusione A $\rightarrow+$ B impone che ogni volta che A viene eseguito, B viene incluso; al contrario, un'esclusione A $\rightarrow\%$ B impone che ogni volta che A viene eseguito, B viene escluso (esce momentaneamente dal flusso di lavoro). L'esclusione di un'attività annulla le relazioni di condizione e risposta.
La relazione milestone tra un'attività A e un'attività B (A $\rightarrow\diamondsuit$ B) nasce con l'introduzione del grafo DCR annidato per indicare un blocco simile alla relazione di condizione; indica che l'attività B non può iniziare se A è contenuto nell'insieme delle risposte in sospeso (Pe). La differenza tra la relazione di condizione e la relazione

milestone fra due attività A e B risiede nel fatto che nella relazione di condizione l'attività B è bloccata solo la prima volta, fin quando l'attività A non viene eseguita, dopo di che potrà essere eseguita più volte. Nella relazione milestone, invece l'attività B può essere bloccata più volte se l'attività A è richiesta come risposta da un'altra attività C.

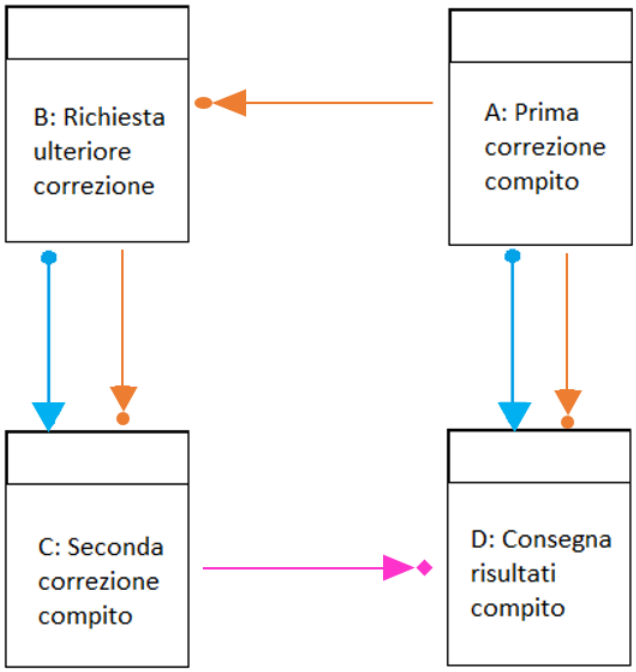


Figura 10: Differenza tra relazione di condizione e relazione milestone.

La Fig. 10 mostra un esempio per chiarire meglio la differenza tra la relazione di condizione e la milestone. Questo semplice flusso di lavoro consiste in una serie di attività che un insegnante deve svolgere dopo aver testato le conoscenze dei suoi alunni con una verifica. In questo caso, si decide, dopo la correzione del compito (A), di non consegnare subito i risultati (D) agli alunni, ma viene richiesta un'ulteriore correzione (B). L'esecuzione di quest'ultima attività implica una seconda correzione

del compito (C), cioè l'attività C dovrà essere eseguita ad un certo punto come conseguenza della relazione di risposta tra B e C. Questo significa che l'attività *Consegna risultati compito* non potrà essere eseguita se prima non avviene la seconda correzione (C), a causa della relazione milestone.

Ricapitolando, sono due i possibili scenari che si possono verificare: o dopo la prima correzione vengono consegnati i risultati agli studenti oppure si richiede un'ulteriore correzione del compito e sarà, quindi, necessario completare la seconda correzione per permettere la consegna finale dei risultati.
La relazione milestone impedisce che la consegna dei risultati (D) possa avvenire dopo l'esecuzione dell'attività B e prima dell'esecuzione dell'attività C. Infatti, se avessimo inserito una relazione di condizione, invece di una milestone, con l'ipotesi che l'attività *C* fosse stata eseguita precedentemente, la *Consegna risultati compito* sarebbe potuta avvenire dopo l'esecuzione dell'attività B.
Un grafo DCR annidato [7] può essere formalmente rappresentato come una tupla $D = (E, Act, \triangleright, M, \rightarrow\bullet, \bullet\rightarrow, \rightarrow\Diamond, \pm, l)$ dove:

- $(E, Act, M, \rightarrow\bullet, \bullet\rightarrow, \rightarrow\Diamond, \pm, l)$ è un grafo DCR.
- $\triangleright$: E $\rightarrow$ E è una funzione parziale che associa un evento al suo super-evento (se definito).

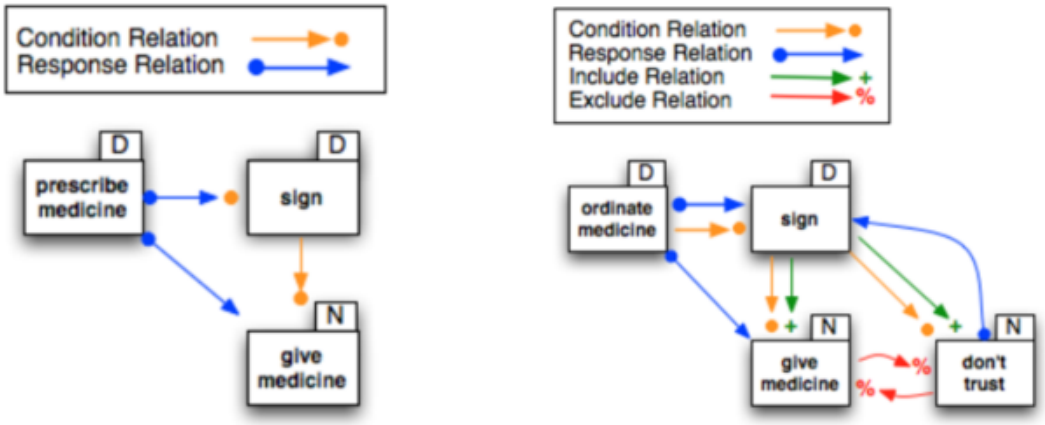
2.2.4 Struttura dei grafi DCR distribuiti

Una struttura di risposta alle condizioni dinamiche distribuite (DDCR) è una tupla $D = (E, Act, M, \rightarrow\bullet, \bullet\rightarrow, \rightarrow\Diamond, \pm, l, R, P, as)$ dove $(E, Act, M, \rightarrow\bullet, \bullet\rightarrow, \rightarrow\Diamond, \pm, l)$ è una struttura DCR, R è un insieme di ruoli, P è un insieme di risorse (ad esempio persone) e $as \subseteq (P \cup Act) \times R$ è la relazione che assegna il ruolo agli esecutori delle azioni.

2.2.5 Esempio

Consideriamo un flusso di lavoro ospedaliero estratto da uno studio di vita reale sul flusso di lavoro oncologico cartaceo negli ospedali danesi. La figura 11(a) mostra il flusso di lavoro di tale studio. Sono presenti tre eventi: *Prescribe medicine* (il medico scrive la dosa di medicinale), *Sign* (il medico firma la prescrizione per confermare la correttezza dei calcoli), *Give medicine* (l'infermiera dà il medicinale al paziente). Ogni evento è etichettato dal ruolo del soggetto di tale evento (D per il medico, N per l'infermiera).

La freccia $\bullet\rightarrow\bullet$ tra *Prescribe medicine* e *Sign* indica che i due eventi sono correlati sia dalla relazione di condizione che dalla relazione di risposta. La relazione di condizione significa che l'evento *Prescribe medicine* deve verificarsi almeno una volta prima dell'evento *Sign*. La relazione di risposta impone che, se si verifica l'evento di *Prescribe medicine*, successivamente, dovrà verificarsi l'evento *Sign* affinché il flusso venga accettato. Allo stesso modo, la relazione di risposta tra *Prescribe medicine* e *Give medicine* impone che, dopo l'esecuzione dell'evento *Prescribe medicine*, dovrà andare in esecuzione anche l'evento di *Give medicine*. Infine, la relazione di condizione tra *Sign* e *Give medicine* richiede che l'evento di firma debba accadere prima che il medicinale possa essere somministrato.
Si noti che l'infermiera può somministrare medicine molte volte e che il medico può, in qualsiasi momento, scegliere di prescrivere una nuova medicina e firmare nuovamente. Ciò non impedirà all'infermiere di continuare a somministrare farmaci. L'interpretazione è che l'infermiera potrebbe dover continuare a somministrare farmaci secondo la prescrizione precedente.



(a) Esempio Prescribe medicine.

(b) Esempio Prescribe medicine esteso.

Figura 11: Flusso di lavoro ospedaliero [9].

L'inclusione e l'esclusione dinamica degli eventi è illustrata da un'estensione dello scenario (tratta anche dal caso di studio reale): se l'infermiere diffida (*Don't trust*) dalla prescrizione del medico dovrebbe essere possibile indicarlo, questa azione dovrebbe forzare una nuova prescrizione seguita da una nuova firma o solo da una nuova firma. Finché la nuova firma non è stata aggiunta, il medicinale non deve essere dato al paziente. Questo scenario può essere modellato come mostrato nella Fig. 11(b), dove viene aggiunto un altro evento etichettato come non affidabile (*Don't trust*). Ora, l'infermiera può scegliere o di dare il medicinale al paziente o di non fidarsi della prescrizione. L'esecuzione dell'azione *Don't trust* comporterà una successiva esecuzione di *Sign*. Quindi, l'unico modo per raggiungere lo stato di accettazione è rieseguire *Sign* che includerà *Prescribe medicine*. Il medico può scegliere di ripetere la prescrizione del medicinale seguita da una firma (se la diffidenza dell'infermiera aveva fondamento) o semplicemente ripetere la firma.

La Fig. 12 mostra una notazione grafica che illustra le informazioni a runtime durante due diverse esecuzioni dello scenario esteso della Fig. 11(b). Le icone (Ø, √,

!) indicano rispettivamente che l'evento non è abilitato, è stato eseguito, è richiesto come risposta. Un evento escluso è indicato con un rettangolo tratteggiato.
La Fig. 12(a) mostra il flusso di lavoro, composto da quattro stati, secondo cui l'infermiera si fida della prescrizione fatta dal medico:

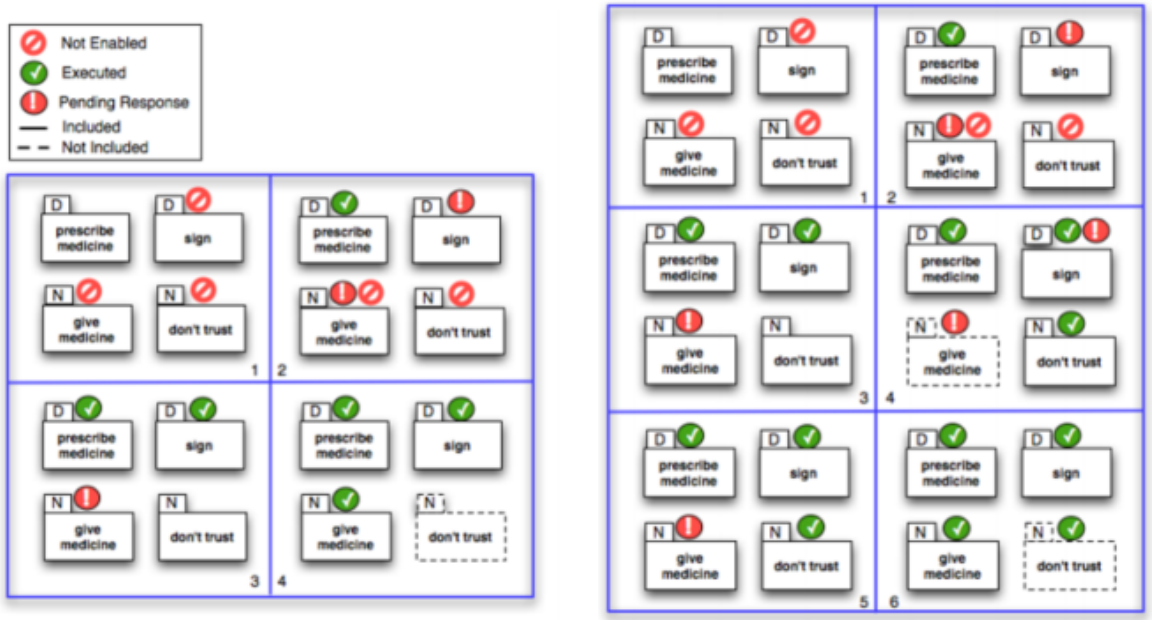
1. Nella condizione iniziale tutti gli eventi, tranne *Prescribe medicine*, sono bloccati.

2. *Prescribe medicine* viene eseguito, di conseguenza l'evento *Sign* e *Give medicine* sono richiesti come risposta. *Sign* non è più bloccato.

3. Il terzo stato è il risultato dell'esecuzione dell'evento *Sign*, che consente o di somministrare la medicina, oppure di non fidarsi.

4. *Give medicine* viene eseguito, l'evento *Don't trust* diventa escluso in quanto non può più verificarsi.

Allo stesso modo, la Fig. 12(b) mostra i sei stati di un flusso di lavoro in cui l'infermiera esegue, nella fase 3, *Don't trust*, che porta a un quarto stato, diverso dal precedente, in cui *Give medicine* è esclusa e *Sign* è richiesto come risposta.
Il quinto stato mostra che il medico esegue la firma (*Sign*), che include nuovamente la somministrazione della medicina, che viene quindi eseguita, portando allo stato finale in cui tutti gli eventi sono stati eseguiti e *Don't trust* è escluso.



(a) Esempio con *Give medicine* in esecuzione.

(b) Esempio con *Don't trust* in esecuzione.

Figura 12: Notazione grafica del flusso di Fig 11(b) [9].

In questo esempio non sono presenti attività collegate da una relazione milestone. La Fig. 13 mostra un'estensione dell'esempio appena visto, in cui possiamo notare la presenza di tale relazione all'interno di un grafo DCR annidato.

Una relazione milestone fra un'attività nidificata A e un'attività B, A $\rightarrow\Diamond$ B, consiste nel permettere l'esecuzione dell'attività B solo dopo che tutte le attività contenute in A sono state eseguite. Un'attività nidificata è considerata eseguita quando tutte le sotto attività che la compongono sono state eseguite. Vediamo solo alcuni vincoli in questo flusso di lavoro. La relazione di condizione da *sign doctor* all'attività *don't trust prescription* significa che l'infermiera può scegliere di non fidarsi della prescrizione solo se, precedentemente, il medico ne abbia apposto la firma. La relazione di risposta dall'attività *prescribe medicine* all'attività *give medicine* significa che quest'ultima deve essere eseguita dopo l'attività *prescribe medicine*.

Un flusso di lavoro è in uno stato completato se tutti questi vincoli di risposta risultano soddisfatti (o se l'attività richiesta è stata esclusa). La relazione milestone tra *medicine preparation* e *administer medicine* significa che nessuna delle attività secondarie di *administer medicine* (*give medicine...*) può essere eseguita se una qualsiasi delle attività secondarie di *medicine preparation* è richiesta come risposta. La relazione di esclusione viene utilizzata tra l'attività *cancel* e l'attività *treatment*. Poiché tutte le altre attività nel flusso di lavoro sono attività secondarie dell'attività *treatment*, tutte le attività vengono escluse se viene eseguita l'attività *cancel*. La relazione di inclusione viene utilizzata tra l'attività *precribe medicine* e la macro attività *manage prescription* per indicare che, dopo l'esecuzione di *prescribe medicine*, verrà inclusa l'attività *manage prescription* per la gestione della prescrizione.

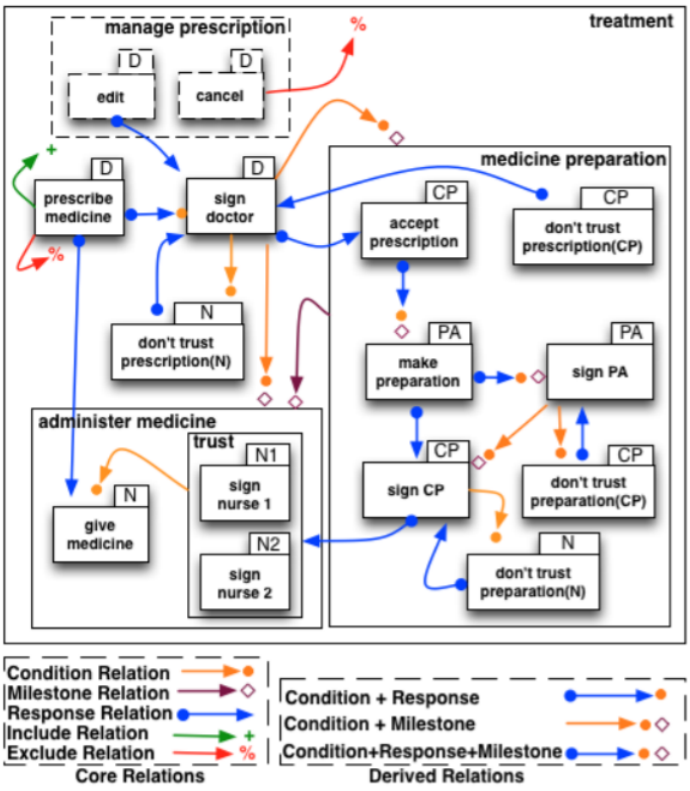


Figura 13: Estensione flusso di lavoro ospedaliero con grafo DCR annidato [7]

2.3 Metrica del Modello

Un algoritmo di discovery del processo [10] implementa un modello di processo partendo da un log di eventi. Per evento si intende il verificarsi di un'attività di un processo, mentre una traccia è una sequenza finita, non vuota, di eventi registrati durante l'esecuzione di tale processo. Quindi, un log di eventi è un insieme di tracce appartenenti a diverse esecuzioni dello stesso processo.

Gli algoritmi di discovery devono ottenere un modello di qualità. Esistono varie metriche e approcci per stabilire la qualità di un modello di processo. In questa sezione verranno presentate le misure di qualità che quantificano l'appropriatezza di un grafo

DCR, indicato con G, per un dato log di eventi l, formato da sole tracce positive. Si farà riferimento alle metriche di fitness, precisione (Precision), semplicità (Simplicity) e generalità (Generality) [6], nonché ad una metrica che misura la qualità di un modello di processo ottenuto da un log di tracce positive e negative [11].

2.3.1 Metriche di replay fitness

Replay fitness è definito come la capacità del modello DCR G di riprodurre accuratamente le tracce del log di input l. Nel concetto di logica a due valori, una traccia può essere conforme al modello (la traccia soddisfa tutti i vincoli del modello) o non conforme (la traccia non soddisfa almeno un vincolo del modello).
Queste metriche non usano il concetto di logica a due valori ma lo estendono in quanto considerano il fatto che una traccia possa non essere completa. La metrica di replay consiste nel calcolare quanto la traccia si discosta dal modello. Supponiamo di avere la seguente traccia:

```
traccia(p1, [evento(a,1),evento(c,3),evento(d,4)])
```

Il modello in questione impone che l'evento a si verifichi prima di b, l'evento b prima di c e l'evento c prima di d. Secondo la logica a due valori questa traccia non sarebbe conforme al modello in quanto b non è presente. La metrica di replay, in questo caso, aggiunge l'attività b alla traccia, rendendola conforme e considera un *token* dovuto a tale modifica. Un evento può sia essere aggiunto ad una traccia, sia essere rimosso; l'importante è considerare un token per ogni modifica.
Per calcolare il replay fitness di un modello si fa la differenza tra il numero totale di tracce e i token e si divide il tutto per il numero totale di tracce. Tale valore sarà conpreso tra 0 e 1.

$$fitness(G, l) = \frac{\#Replayabletraces(G, l)}{\#Traces(l)}$$

L'algoritmo tipico che permette una perfetta riproduzione è l'algoritmo UlrikHovsgaard.

2.3.2 Metriche di precisione

La precisione è definita come la somma delle quantità di azioni comportamentali non utilizzate dal log. Nei grafi DCR questo si traduce nel riprodurre il log e registrare, per ogni stato raggiunto nel grafo, le attività eseguibili in generale in quello stato e quante di queste sono state eseguite fino ad un certo punto.

$$precision(G, l) = \frac{\sum_{S \in VisitedStates(l)} \#ActivitiesExecuted(G, s, l)}{\sum_{S \in VisitedStates(l)} \#ExecutableActivities(G, s, l)}$$

L'attività eseguibile in un determinato stato "#ActivitiesExecuted" viene conteggiata solo la prima volta, quindi se si osserva che tale attività viene eseguita più volte dallo stesso stato, si conta solo la prima esecuzione. La metrica in questione conta le attività che sono possibili nel modello, ma mai considerate nel log. Se non ci sono tali attività, la precisione è perfetta. Un modello è preciso se non ammette comportamenti troppo generici, quindi un modello non preciso è in "underfitting".

2.3.3 Metriche di semplicità

La semplicità è definita come il rapporto tra la dimensione dell'albero di processo interno e il numero di attività nel log di input. Nei modelli di processo dichiarativi, i principali ostacoli alla comprensibilità sono il numero di vincoli rispetto al numero di attività. Di conseguenza, si misura la semplicità di un grafo DCR dal numero di coppie di attività correlate (RP) e dal numero totale delle relazioni (RT). Naturalmente, RT è maggiore di RP quando alcune attività sono correlate da più di una relazione. Basandoci su quanto appena visto, possiamo affermare che un grafo più semplice possibile è un grafo senza relazioni.

$$simplicity(G) = \frac{(1 - \frac{\#Relations}{\#PossibleRelations})}{2} + \frac{(1 - \frac{\#RPs}{\#PossibleRPs})}{2}$$

Queste tecniche di discovery, generalmente, danno come risultato modelli semplici, ma con scarso replay fitness e precisione.

2.3.4 Metriche di generalità

La generalità ha lo scopo di valutare la misura in cui il modello risultante sarà in grado di riprodurre comportamenti futuri. Tale metrica considera la frequenza con cui ogni nodo dell'albero deve essere visitato se il modello deve produrre il dato log. Se un nodo viene visitato spesso, allora si è certi che il suo comportamento sia corretto. Se alcune parti dell'albero sono visitate molto di rado, la generalizzazione assume un valore molto basso.
Un modello che non generalizza è in "overfitting", cioè è troppo specifico; di conseguenza, se tale modello spiega un particolare log, è improbabile che un altro log dello stesso processo possa essere spiegato bene dallo stesso modello. Questa è una proprietà estremamente importante per i modelli dichiarativi. Tuttavia, non può essere ragionevolmente misurata senza fare appello alla conoscenza del dominio: non è possibile solo dai log determinare quali sono generalizzazioni utili e quali no.

2.3.5 Metrica di un modello ottenuto da tracce positive e negative

Questa metrica è utilizzata nella logica a due valori e consiste nel calcolare l'accuratezza di un modello di processo contando le tracce conformi o meno al modello. In particolare, l'accuratezza è data dalla somma delle tracce conformi che sono classificate come conformi e delle tracce non conformi che sono classificate come non conformi, diviso il numero totale di tracce.

$$accuratezza(G, l) = \frac{Tracce\_Conformi + Tracce\_Non\_Conformi}{Tracce\_Totali}$$

2.4 Algoritmi di discovery per DCR graphs

Lo studio degli algoritmi di discovery per i grafi DCR è ancora nelle sue fasi iniziali, a causa della sua recente scoperta, ma ha compiuto passi da gigante attraverso lo sviluppo di due algoritmi di discovery: gli algoritmi UlrikHovsgaard e ParNek [12]. Entrambi gli algoritmi adottano un approccio euristico che è lineare nella dimensione del log.

2.4.1 UlrikHovsgaard

L'algoritmo UlrikHovsgaard è stato il primo algoritmo di discovery del processo dichiarativo per i grafi DCR. Si parte da un modello totalmente vincolato, in cui ognuna delle quattro relazioni di base (la milestone non è stata implementata) si trova tra ogni attività osservata nel log. L'algoritmo analizza le tracce del log per determinare quale delle relazioni può essere violata.
Tutto ciò funziona ma è incompleto: il modello risultante ha una fitness perfetta (cioè il modello riproduce perfettamente le tracce nel log), ma esiste la possibilità che alcune relazioni possano essere rimosse nonostante siano in grado di fornire una maggiore precisione. Poiché l'algoritmo parte da un modello completamente vincolato, tende a produrre un numero elevato di vincoli, portando, spesso, a modelli eccessivamente complessi e di difficile lettura, riducendo, pertanto, la loro utilità per gli utenti finali.

2.4.2 ParNek

L'algoritmo di discovery ParNek nasce dall'osservazione dell'algoritmo UlrikHovsgaard, in quanto, essendo completamente vincolato, non risultava conforme al paradigma dichiarativo. L'obiettivo dei modelli dichiarativi è, infatti, quello di catturare le regole di un processo e consentire qualsiasi comportamento che non sia vietato, di conseguenza adotta un approccio differente rispetto all'algoritmo UlrikHovsgaard: esso

parte da un modello non vincolato e cerca i vincoli osservando gli eventi presenti nel log.

Questo viene fatto attraverso una serie di euristiche e con due alternative possibili: la prima che tenta di trovare solo i vincoli più rilevanti, sacrificando la precisione del modello a favore della sua semplicità, mentre la seconda trova quanti più vincoli possibili, sacrificando la semplicità del modello a favore della precisione.

La prima alternativa ha migliorato notevolmente la semplicità, sacrificando poca precisione, mentre la seconda ha avuto performance superiori rispetto all'algoritmo UlrikHovsgaard; infatti, essendo in grado di trovare combinazioni di relazioni più efficaci per descrivere il processo mostrato dal log, è stato possibile ottenere modelli altrettanto precisi con una quantità minore di vincoli.

3 L'algoritmo

Gli algoritmi menzionati nel capitolo precedente (Parnek e UlrikHovsgaard), così come la maggior parte degli algoritmi di discovery, usano solo tracce positive, cioè una sequenza di eventi dal comportamento desiderato.
L'algoritmo implementato nel presente lavoro fa uso, invece, di un log di eventi che contiene sia tracce positive che negative. Quest'ultime rappresentano quell'insieme di eventi che non si dovrebbero mai presentare, in quanto sono comportamenti che si vuole evitare di avere nel modello.
Il vantaggio di utilizzare un log che contiene sia tracce positive che negative è sicuramente quello di ottenere un modello che si avvicini maggiormente alla realtà; infatti si andranno a considerare sia gli eventi ammissibili, che quelli proibiti, ignorando le parti non significative e semplificando, conseguentemente, il modello finale. Lo svantaggio risiede, tuttavia, nella difficoltà di discriminare, all'interno dei log di eventi, gli esempi positivi da quelli negativi; per tale motivo, solitamente, si lavora solo su tracce positive.
Al fine di chiarire il concetto, di seguito è mostrato un esempio di un log di una pizzeria d'asporto con un numero di tracce estremamente basso. Questo log potrebbe contenere le seguenti tracce:

```
traccia(p1, [evento(PrenotazioneEffettuata,1), evento(PagamentoRicevuto,2),
        evento(PreparazionePizzaConIngredientiCorretti,3),
        evento(ConsegnaInOrario,4)]).

traccia(n1, [evento(PrenotazioneEffettuata,1), evento(PagamentoRicevuto,2),
        evento(PreparazionePizzaConIngredientiSbagliati,3),
        evento(ConsegnaInOrario,4)]).
```

```
traccia(n2, [evento(PrenotazioneEffettuata,1), evento(PagamentoRicevuto,2),
        evento(PreparazionePizzaConIngredientiCorretti,3),
        evento(ConsegnaInRitardo,4)]).
```

La traccia p1 è chiaramente una traccia positiva in quanto composta da un insieme di attività che ci si aspetta si verifichino; contrariamente, le tracce n1 e n2 rappresentano tracce negative in quanto nel primo caso la pizzeria ha consegnato una pizza che il cliente non voleva, nel secondo caso la consegna è avvenuta in un orario successivo rispetto a quello preventivato.

3.1 Mooney

L'algoritmo che verrà discusso si ispira a quello presentato da Mooney, in quanto fa uso sia di tracce positive che negative, ma viene adattato in modo tale da restituire un modello i cui vincoli sono espressi in DCR. Mooney presenta due algoritmi [1] per l'apprendimento delle clausole Horn del primo ordine, uno che restituisce come risultato finale un'espressione DNF (Disjunctive Normal Form), cioè una formula logica costituita da disgiunzioni di congiunzioni (OR di AND), l'altro che restituisce come risultato finale un'espressione CNF (Conjunctive Normal Form), cioè una formula logica costituita da congiunzioni di disgiunzioni (AND di OR). Tali algoritmi sono uno il duale dell'altro e verranno adattati per elaborare il mining.
Nei successivi paragrafi verranno mostrate le implementazioni in pseudo-codice. Si introducono alcuni elementi che saranno menzionati in questo studio:

- *Pos* è l'insieme di esempi positivi.
- *Neg* è l'insieme di esempi negativi.
- *DNF*(*CNF*) è l'espressione DNF(CNF) finale, cioè il modello risultante.

- *Term* rappresenta il termine che andrà a comporre il modello.
- C è un letterale, rappresenta un vincolo del nostro modello.

3.2 Algoritmo DNF

Lo pseudo-codice dell'algoritmo DNF è mostrato in Fig. 14. Esso è costituito, essenzialmente, da due cicli:

- **Ciclo esterno**: partendo da una copia dei due insiemi, *Pos* e *Neg*, il ciclo si occupa di comporre l'espressione DNF finale aggiungendo i termini, prodotti nel ciclo interno, in OR fin quando l'insieme degli esempi positivi non risulterà vuoto. Quando l'insieme *Pos* diventa vuoto l'algoritmo finisce la sua esecuzione restituendo l'espressione DNF finale (modello).
- **Ciclo interno**: dagli insiemi dei positivi e dei negativi si sceglie il vincolo C che massimizza la funzione di guadagno (*DNF-gain()*). Dopo aver aggiunto tale vincolo in AND a *Term*, l'algoritmo procede eliminando dalle copie degli insiemi *Pos* e *Neg* tutte le tracce che non soddisfano il vincolo C scelto. Queste operazioni continuano fin quando l'insieme degli esempi negativi diventa vuoto, che si traduce nell'esclusione di tutte le tracce negative. In tal caso, si esce dal ciclo e il termine, appena formato, viene aggiunto, in OR, all'espressione DNF finale.

Nel ciclo interno l'algoritmo chiama una funzione (*DNF-gain*) per scegliere il vincolo più conveniente e lo aggiunge, in AND, al termine che sta costruendo. Il criterio di scelta adottato consiste nel prendere il vincolo che copra più tracce positive, nonostante l'esclusione dello stesso numero di tracce negative. Quindi, l'algoritmo rimuove dalla copia degli esempi positivi e negativi quelli che non soddisfano il nuovo vincolo.

Dopo aver trovato un termine contenente vincoli che escludono tutte le tracce negative, il ciclo interno termina; il termine viene aggiunto, in OR, ai precedenti (DNF) e tutte le tracce positive che soddisfano il nuovo termine vengono rimosse dall'insieme originale dei positivi. Quando questo elenco diventa vuoto, l'algoritmo termina e restituisce il modello. Poiché l'obiettivo del ciclo interno è trovare un termine che escluda tutte le tracce negative, probabilmente, verranno escluse anche alcune di quelle positive. Il ciclo esterno elimina le tracce coperte dall'insieme delle tracce positive per consentire al ciclo interno di trovare un secondo termine che si concentrerà sulla ricerca di vincoli non ancora selezionati.

DCR non contiene l'AND e l'OR nel suo linguaggio, di conseguenza, occorre utilizzare una rappresentazione specifica del modello che sia conforme a DCR. Tale rappresentazione consiste nell'implementare una lista di liste, dove le varie liste sono considerate in OR, mentre gli elementi interni alle liste sono in AND.

Algorithm 1 DNF learner by Mooney

```
Let Pos be all the positive examples.
Let DNF be empty.
Until Pos is empty do:
    Let Neg be all the negative examples.
    Set Term to empty and Pos2 to Pos.
    Until Neg is empty do:
        Choose the constraint that maximizes the DNF-gain.
        Add the constraint C to Term.
        Remove from Neg all the examples that do not satisfy C.
        Remove from Pos2 all the examples that do not satisfy C.
    Add Term as one term of DNF
    Remove from Pos all the examples that satisfy Term.
Return DNF.
```

Figura 14: Pseudo-codice dell'algoritmo DNF di Mooney [13]

3.2.1 Implementazione

L'algoritmo DNF restituisce un'espressione DNF costituita da disgiunzioni di congiunzioni. La generazione di un termine inizia dalla clausola più generale, specializzandola nelle successive iterazioni fin quando nessuna traccia negativa risulterà coperta. La specializzazione può avvenire in tre modi:

1. Aggiungendo vincoli in AND al termine che si sta costruendo.

2. Rimuovendo un vincolo appartenente al termine e sostituirlo con uno più specifico.

3. Facendo uso di variabili. Questo tipo di specializzazione dipende dalla semantica del vincolo che si sta considerando, ad esempio, dato un vincolo iniziale *respon-*

se(a,b), una specializzazione di tale vincolo potrebbe essere *response(X,b)*, cioè X è una variabile che può unificare con diverse attività. Questo si traduce nel fatto che *b* non possa che essere l'ultima attività della traccia in quanto ogni attività che unificherà con X dovrà essere eseguita prima di *b*. Invece, dato un vincolo iniziale *condition(a,b)*, una possibile specializzazione potrebbe essere *condition(a,X)*. In questo caso ogni attività che unifica con X, se presente nella traccia, dovrà essere preceduta dall'attività *a*, cioè *a* deve essere la prima attività.

DCR utilizza il primo approccio in quanto, a differenza di Declare, non si ha una gerarchia di vincoli, cioè non esistono vincoli che siano combinazione logica di altri vincoli più semplici [14]. La funzione chiamata nel ciclo interno che sceglie il vincolo successivo da aggiungere al termine si chiama *choose_constraint* ed è mostrata in Fig. 15.

```
choose_constraint(ListOfPosEx, ListOfNegEx, Term, NewConstraint) :-
    get_list_of_activities(ListOfPosEx, ListOfNegEx, Activities),
    get_constraints(Constraints),
    combine(Constraints, Activities, ListOfPossibleCandidates),
    get_best(ListOfPossibleCandidates, ListOfPosEx, ListOfNegEx,
    Activities, [], -9999, NewConstraint).
```

Figura 15: Codice della funzione *Choose_constraint* dell'algoritmo DNF

Questa funzione prende in ingresso tre parametri: l'insieme degli esempi positivi, l'insieme degli esempi negativi e l'elenco dei vincoli già aggiunti al termine e restituisce il nuovo vincolo. Per prima cosa, vengono estrapolate le attività presenti all'interno degli insiemi di esempi positivi e negativi, dopo di che, vengono combinate con i vincoli DCR in modo tale da ottenere una lista di tutti i possibili vincoli che possono appartenere al modello.

Ad esempio, dati i seguenti insiemi di tracce:

```
Insieme_Esempi_Positivi = [[p1,[evento(a,3),evento(b,6)]],[p2,[evento(b,3)]]].
Insieme_Esempi_Negativi = [[n1,[evento(b,2),evento(a,3)]]].
```

la lista di tutti i possibili vincoli è:

```
[condition(a,b),condition(b,a),exclusion(a,a),exclusion(a,b),
exclusion(b,a),exclusion(b,b),inclusion(a,b),inclusion(b,a),
milestone(a,b),milestone(b,a),response(a,b),response(b,a)]
```

Si noti che solo il vincolo *exclusion* può essere applicato allo stesso evento, questo è dovuto al fatto che l'esecuzione di un'attività può escludere la stessa. Tale vincolo è usato quando si necessita che un'attività venga eseguita solo la prima volta all'interno del flusso. Inoltre, eventuali duplicati vengono rimossi.

```
Function DNF-gain(L,Pos,Neg)
    Let P be the number of examples in Pos and N the number of examples in Neg
    Let p be the number of examples in Pos that satisfy L.
    Let n be the number of examples in Neg that satisfy L.
    Return p*(log(p/(p+n)) - log(P/(P+N)))
```

Figura 16: Codice della funzione *DNF_gain* [1]

Adesso, l'algoritmo deve scegliere, da tale lista, un vincolo; questo viene fatto attraverso la funzione *get_best*, in cui viene calcolato, per ogni vincolo presente nella lista, un valore numerico, detto guadagno, che stabilisce la bontà di tale vincolo rispetto alle tracce positive e negative. La funzione che si occupa di fare questo, descritta da Mooney [1], è la *dnf_gain* mostrata in Fig. 16. Questa funzione calcola il guadagno di ogni vincolo prendendo in considerazione il numero di tracce positive P, il numero di tracce negative N, il numero di tracce positive soddisfatte p e il numero di tracce negative soddisfatte n. Nell'algoritmo in questione, se p è 0, al vincolo

viene assegnato un guadagno molto basso che assicura che il vincolo non venga scelto. Questa assegnazione è necessaria perché il logaritmo di 0 è -∞, il che genera un errore in Prolog. In ogni caso, un vincolo che non copre nessuna traccia positiva non deve essere aggiunto nel termine in quanto, essendo in AND con tutti gli altri vincoli, farebbe sì che l'intero termine non copra alcuna traccia positiva e verrebbe scartato. La *get_best* ritorna il vincolo con il valore di guadagno più alto.
Una volta scelto il vincolo da aggiungere al termine, la funzione *remove* rimuove dall'insieme degli esempi positivi e dall'insieme degli esempi negativi le tracce che non soddisfano il nuovo vincolo. Adesso, se nell'insieme degli esempi negativi è ancora presente almeno una traccia si andrà alla ricerca di un altro vincolo da aggiungere, in AND, al precedente per escludere le tracce rimanenti; altrimenti, se l'insieme è vuoto, il ciclo interno termina e il termine arriverà al ciclo esterno venendo aggiunto, in OR, al modello. In quest'ultimo caso si rimuovono, dall'insieme dei positivi, quelle tracce che soddisfano il termine, iterando più volte fin quando non si andranno a coprire tutte le tracce positive.
Quando l'insieme degli esempi positivi risulterà vuoto, l'algoritmo termina e restituisce l'espressione DNF finale. Si noti che è possibile che qualche traccia non venga coperta, in tal caso si ritornano le tracce non coperte.
Un problema che si potrebbe verificare è la presenza di una traccia nell'intersezione dei due insiemi. In tal caso l'algoritmo non riesce a trovare un vincolo che copra tale traccia e contemporaneamente la escluda. Dal punto di vista logico tale situazione non può verificarsi, in quanto inconsistente, pertanto, deve essere l'utente a fornire all'algoritmo degli insiemi di tracce che siano corretti. Nonostante ciò, dopo la lettura degli insiemi di tracce positive e negative, viene effettuato un controllo per verificare se i due insiemi sono disgiunti e in tal caso l'algoritmo prosegue. In caso contrario, viene lanciato un messaggio di errore e termina.

3.2.2 Esempio

Per capire meglio come funziona l'algoritmo, viene mostrato un esempio su un numero limitato di tracce. Dato il seguente log di eventi:

Tracce positive:

```
traccia(p1, [evento(a,1),evento(b,2),evento(c,3)]).
traccia(p2, [evento(c,3),evento(a,4)]).
traccia(p3, [evento(a,3),evento(c,4),evento(b,6)]).
```

Tracce negative:

```
traccia(n1, [evento(a,1),evento(b,2)]).
traccia(n2, [evento(c,3),evento(b,4),evento(a,5),evento(c,6)]).
```

L'algoritmo inizia con la lettura degli insiemi di esempi positivi e negativi, dopo di che entra nel ciclo interno in cui la prima funzione chiamata è la *choose_constraint* che trova il miglior vincolo da aggiungere al termine. Tale funzione ottiene dal log di eventi tutte le attività presenti e genera i vincoli DCR da analizzare. Successivamente, viene chiamata la *get_best* che calcola, per ciascun vincolo, il guadagno e seleziona quello con il valore più alto:

```
response(a,b)      0.09151498112135026
response(a,c)     0.09151498112135026
response(b,a)      -0.0791812460476248
response(b,c)     0.09151498112135026
response(c,a)     -0.255272505103306
response(c,b)     -0.255272505103306
condition(a,b)    0.29073003902416933
condition(a,c)    -0.1583624920952496
```

```
condition(b,a)   -9999
condition(b,c)   -0.255272505103306
condition(c,a)   -0.0791812460476248
condition(c,b)     0.09151498112135026
milestone(a,b)   0.29073003902416933
milestone(a,c)     -0.1583624920952496
milestone(b,a)     -9999
milestone(b,c)   -0.255272505103306
milestone(c,a)   -0.0791812460476248
milestone(c,b)   0.09151498112135026
inclusion(a,b)   0.09151498112135026
inclusion(a,c)   0.09151498112135026
inclusion(b,a)   -9999
inclusion(b,c)   -0.0791812460476248
inclusion(c,a)   -0.0791812460476248
inclusion(c,b)   -0.0791812460476248
exclusion(a,a)   0.0
exclusion(a,b)   -0.0791812460476248
exclusion(a,c)   -0.0791812460476248
exclusion(b,a)   0.29073003902416933
exclusion(b,b)   0.0
exclusion(b,c)   0.09151498112135026
exclusion(c,a)   0.09151498112135026
exclusion(c,b)   0.09151498112135026
exclusion(c,c)   0.29073003902416933
```

Il vincolo scelto è *condition(a,b)*. Adesso, il ciclo interno deve valutare quante tracce negative non soddisfano questo vincolo. In questo caso, il vincolo esclude solo una delle due tracce negative, più precisamente, la traccia *n2* in quanto l'evento *b* viene eseguito nonostante, in precedenza, non venga eseguito l'evento *a*. Di conseguenza, bisogna trovare un altro vincolo da aggiungere, in AND, al termine per escludere la restante traccia negativa. Viene, dunque, richiamata la *choose_constraint* che ritorna *response(a,c)* dalla seguente lista di vincoli:

```
response(a,b)     -0.10230504489476264
response(a,c)    0.2498774732165999
response(b,a)    0.12493873660829995
response(b,c)    0.2498774732165999
response(c,a)    -0.17609125905568124
response(c,b)    -0.17609125905568124
condition(a,b)    0.0
condition(a,c)   -0.10230504489476264
condition(b,a)   -9999
condition(b,c)   -0.17609125905568124
condition(c,a)    0.12493873660829995
condition(c,b)    0.2498774732165999
milestone(a,b)   0.0
milestone(a,c)   -0.10230504489476264
milestone(b,a)   -9999
milestone(b,c)   -0.17609125905568124
milestone(c,a)   0.12493873660829995
milestone(c,b)   0.2498774732165999
```

```
inclusion(a,b)   -0.10230504489476264
inclusion(a,c)   0.2498774732165999
inclusion(b,a)   -9999
inclusion(b,c)   0.12493873660829995
inclusion(c,a)   0.12493873660829995
inclusion(c,b)   0.12493873660829995
exclusion(a,a)   0.0
exclusion(a,b)   0.12493873660829995
exclusion(a,c)   -0.17609125905568124
exclusion(b,a)   0.0
exclusion(b,b)   0.0
exclusion(b,c)   -0.10230504489476264
exclusion(c,a)   -0.10230504489476264
exclusion(c,b)   -0.10230504489476264
exclusion(c,c)   0.0
```

Tale vincolo esclude l'unica traccia negativa rimasta *n1*, in quanto, dopo l'esecuzione dell'evento *a*, non viene eseguito l'evento *c*. L'insieme degli esempi negativi, dunque, diventa vuoto e il termine *t1 = response(a,c) AND condition(a,b)* farà parte del modello.

Adesso l'algoritmo verifica se tutte le tracce positive vengono coperte dal modello. La risposta è negativa in quanto solo due tracce su tre vengono coperte, infatti, la traccia *p2* non viene coperta dal vincolo *response(a,c)* in quanto, dopo l'esecuzione dell'evento *a*, non viene eseguito *c*. Di conseguenza, si cerca un altro termine da aggiungere in OR al modello per provare a coprire la restante traccia positiva. Dopo una nuova esecuzione del ciclo interno viene restituito il termine *t2 = condition(a,b)*

AND response(b,a) che, aggiunto al modello, darà come risultato finale la seguente espressione DNF:

```
(condition(a,b) AND response(b,a)) OR (response(a,c) AND condition(a,b))
```

Come vediamo questo modello copre tutte le tracce positive ed esclude tutte quelle negative.

3.3 Algoritmo CNF

L'algoritmo CNF è duale rispetto a quello DNF descritto precedentemente. Tale algoritmo restituisce un modello nella forma normale congiuntiva, cioè un AND di OR. Per comprendere meglio le differenze tra i modelli generati dai due algoritmi, se il log in questione è composto dalle seguenti tracce:

Tracce positive:

```
traccia(p1, [evento(a,1),evento(b,2),evento(c,3)]).
traccia(p2, [evento(c,3),evento(a,4)]).
traccia(p3, [evento(a,3),evento(c,4),evento(b,6)]).
```

Tracce negative:

```
traccia(n1, [evento(a,1),evento(b,2)]).
traccia(n2, [evento(c,3),evento(b,4),evento(a,5),evento(c,6)]).
```

Il modello restituito dall'algoritmo DNF è:

```
(condition(a,b) AND response(b,a)) OR  (response(a,c) AND condition(a,b))
```

mentre quello restituito dall'algoritmo CNF è:

```
(response(b,a) OR response(a,c)) AND  condition(a,b)
```

Si noti come i due modelli risultanti sono uno il duale dell'altro.

Lo pseudo-codice dell'algoritmo CNF è mostrato in Fig. 17. In questo caso il ciclo esterno termina quando tutte le tracce negative sono escluse dal modello e il ciclo interno quando tutte quelle positive soddisfano la clausola. Ciò significa che ogni clausola dell'algoritmo, composta da vincoli in OR, dovrà necessariamente coprire tutte le tracce positive nel log.
Come per l'algoritmo DNF, anche in CNF l'AND e l'OR non fanno parte del linguaggio DCR. Le clausole sono, quindi, rappresentate come una lista di liste di vincoli, in cui ogni lista contiene i vincoli in OR, mentre le varie liste rappresentano l'AND. Nell'esempio di cui sopra, il modello restituito da CNF è nella forma:

```
[[response(b,a), response(a,c)],[condition(a,b)]]
```

La precedente è solo una riscrittura di tale lista di liste per rendere il modello più leggibile.

Algorithm 3 CNF learner by Mooney

```
Let Neg be all the negative examples.
Let CNF be empty.
Until Neg is empty do:
    Let Pos be all the positive examples.
    Set Clause to empty and Neg2 to Neg.
    Until Pos is empty do:
        Choose the constraint C that maximizes the CNF-gain.
        Add the constraint C to Clause.
        Remove from Pos all the examples that satisfy C.
        Remove from Neg2 all the examples that satisfy C.
    Add Clause as one clause of CNF
    Remove from Neg all the examples that do not satisfy
     Clause.
Return CNF.
```

Figura 17: Pseudo-codice dell'algoritmo CNF di Mooney [13]

Dagli insiemi dei positivi e dei negativi, il ciclo interno sceglie il vincolo C che massimizza la funzione di guadagno (*CNF-gain()*). Dopo aver aggiunto in OR tale vincolo a *Clause*, l'algoritmo rimuove dalle copie degli insiemi di esempi positivi e negativi, *Pos* e *Neg*, tutte le tracce che soddisfano il nuovo vincolo (l'algoritmo DNF, invece, rimuove quelli che non lo fanno).

Questo perché ogni clausola deve coprire tutte le tracce positive e poiché ogni vincolo in esse è in OR, anche le tracce coperte da un solo vincolo saranno coperte dall'intera clausola. Rimuovendoli dall'elenco, le iterazioni successive si concentrano sulle tracce positive che non sono ancora state coperte, quindi la clausola non avrà vincoli ridondanti che, probabilmente, includendo tracce negative, renderebbero il modello più ampio.

Il ciclo itera fin quando l'insieme degli esempi positivi diventa vuoto, in tal caso, si

esce dal ciclo interno e la clausola appena creata viene aggiunta in AND al modello. Adesso, nel ciclo esterno, si andranno ad escludere le tracce negative che non soddisfano il nuovo modello: se l'insieme risulta vuoto, allora sono state escluse tutte le tracce negative, viene restituito il modello e l'algoritmo termina; altrimenti, si andrà alla ricerca di un'altra clausola da aggiungere in AND al modello per escludere le tracce negative rimaste.

3.3.1 Implementazione

L'algoritmo CNF restituisce un'espressione CNF costituita da congiunzioni di disgiunzioni. La generazione di una clausola inizia dalla più specifica e la generalizza nelle iterazioni successive fin quando tutte le tracce positive saranno coperte.
La generalizzazione può avvenire in tre modi:

1. Aggiungendo vincoli in OR alla clausola che si sta costruendo.

2. Rimuovendo un vincolo appartenente alla clausola e sostituirlo con uno più generale.

3. Tramite uso di variabili. Questo tipo di generalizzazione dipende dalla semantica del vincolo che si sta considerando, ad esempio, dato un vincolo iniziale *response(a,b)*, una generalizzazione di tale vincolo potrebbe essere *response(a,X)*, dove X è una variabile che può unificare con diverse attività. Questo si traduce nel fatto che a deve essere eseguita prima di qualsiasi altra attività che unifica con X. Invece, dato un vincolo iniziale *condition(a,b)*, una possibile generalizzazione potrebbe essere *condition(X,b)*. In questo caso b, se presente nella traccia, dovrà essere preceduta dall'attività che unifica con X, cioè b deve essere l'ultima attività della traccia.

Data l'assenza di gerarchia di vincoli in DCR, anche CNF utilizza il primo approccio.

La funzione chiamata nel ciclo interno, che sceglie il vincolo migliore da aggiungere alla clausola, è la *choose_constraint*, la stessa usata per l'algoritmo DNF. La differenza sta nella funzione che si occupa di calcolare il guadagno di ogni vincolo, cioè nella *CNF_gain*.

Questa funzione, descritta da Mooney [1], calcola il guadagno di ogni vincolo prendendo in considerazione il numero di tracce positive P, il numero di tracce negative N, il numero di tracce positive che non soddisfano il nuovo vincolo p e il numero di tracce negative che non soddisfano il nuovo vincolo n (p e n, nell'algoritmo DNF, erano rispettivamente il numero di tracce positive e negative che soddisfavano il nuovo vincolo). Se n è 0, allora il vincolo copre ogni traccia negativa e questa è una situazione indesiderata, di conseguenza, verrà assegnato al vincolo un valore di guadagno molto basso per evitare che venga scelto. Occorre, quindi, considerare quei vincoli che escludono un numero alto di tracce negative.

Dopo aver scelto il vincolo che possiede il valore di guadagno più alto, la funzione *remove* rimuove dall'insieme degli esempi positivi e dall'insieme degli esempi negativi le tracce che soddisfano il nuovo vincolo. Se nell'insieme degli esempi positivi è ancora presente almeno una traccia, si andrà alla ricerca di un altro vincolo da aggiungere in OR al precedente per escludere le tracce rimanenti; altrimenti, se l'insieme è vuoto, il ciclo interno termina e restituisce la clausola al ciclo esterno, che andrà aggiunto in AND al modello. In quest'ultimo caso si rimuovono, dall'insieme degli esempi negativi, quelle tracce che non vengono coperte dal vincolo. Quando l'insieme degli esempi negativi risulta vuoto, l'algoritmo termina e restituisce l'espressione CNF finale.

3.3.2 Esempio

Al fine di chiarire il funzionamento dell'algoritmo CNF, di seguito è mostrato un esempio.
Il log preso in considerazione è lo stesso utilizzato per l'algoritmo DNF:
Tracce positive:

```
traccia(p1, [evento(a,1),evento(b,2),evento(c,3)])
traccia(p2, [evento(c,3),evento(a,4)])
traccia(p3, [evento(a,3),evento(c,4),evento(b,6)])
```

Tracce negative:

```
traccia(n1, [evento(a,1),evento(b,2)])
traccia(n2, [evento(c,3),evento(b,4),evento(a,5),evento(c,6)])
```

L'algoritmo inizia con la lettura degli insiemi di esempi positivi e negativi, successivamente, si entra nel ciclo interno in cui la prima funzione chiamata è la *choose_constraint* che trova il miglior vincolo da aggiungere alla clausola. Nello specifico, tale funzione, ottiene dal log di eventi tutte le attività presenti e genera i vincoli DCR da analizzare. A questo punto interviene la *get_best* che calcola, per ciascun vincolo, il guadagno, selezionando quello con il valore più alto:

```
response(a,b)      0.0969100130080564
response(a,c)     0.0969100130080564
response(b,a)     -0.07918124604762483
response(b,c)     0.0969100130080564
response(c,a)     -9999
response(c,b)     -9999
condition(a,b)    0.3979400086720376
```

```
condition(a,c)   -9999
condition(b,a)   -0.2041199826559248
condition(b,c)   -9999
condition(c,a)     -0.07918124604762483
condition(c,b)   0.0969100130080564
milestone(a,b)   0.3979400086720376
milestone(a,c)   -9999
milestone(b,a)   -0.2041199826559248
milestone(b,c)   -9999
milestone(c,a)   -0.07918124604762483
milestone(c,b)   0.0969100130080564
inclusion(a,b)   0.0969100130080564
inclusion(a,c)   0.0969100130080564
inclusion(b,a)   -0.2041199826559248
inclusion(b,c)     -0.07918124604762483
inclusion(c,a)     -0.07918124604762483
inclusion(c,b)   -0.07918124604762483
exclusion(a,a)   -9999
exclusion(a,b)   -0.07918124604762483
exclusion(a,c)   -0.07918124604762483
exclusion(b,a)     0.3979400086720376
exclusion(b,b)     -9999
exclusion(b,c)   0.0969100130080564
exclusion(c,a)   0.0969100130080564
exclusion(c,b)   0.0969100130080564
exclusion(c,c)   0.3979400086720376
```

Il vincolo scelto è *condition(a,b)* in quanto ha un valore di guadagno di 0.3979400086720376, che risulta maggiore o uguale rispetto ai valori dei restanti vincoli. Adesso il ciclo interno escluderà le tracce positive che vengono coperte da tale vincolo.
Questo è un caso particolarmente fortunato in quanto tre tracce su tre vengono coperte da *c1 = condition(a,b)*, pertanto, si ottiene, in maniera immediata, la clausola da aggiungere in AND al modello.
Nel ciclo esterno, quindi, l'algoritmo verifica se tutte le tracce negative dell'insieme degli esempi negativi vengono escluse dal modello. La risposta è negativa in quanto solo una traccia (*n2*) è esclusa, infatti, la traccia *n1* soddisfa *condition(a,b)* in quando l'evento *b* viene eseguito e, in precedenza, è stato eseguito l'evento *a*.
In questo caso, l'algoritmo andrà alla ricerca di un'altra clausola da aggiungere, in AND, al modello attualmente presente per escludere la traccia negativa rimasta. Dopo una nuova esecuzione del ciclo interno viene restituita la clausola *c2 = response(b,a) OR response(a,c)*. La presenza di *response(b,a)* è dovuta al fatto che il vincolo *response(a,c)*, da solo, non è sufficiente per coprire tutte le tracce positive, infatti, nella traccia *p2* non è presente un'evento *c* dopo l'esecuzione di *a*. Con l'aggiunta di *response(b,a)*, in OR, alla clausola parziale, la traccia positiva *p2* risulta coperta in quanto l'evento *b* non viene eseguito e quindi la clausola *c2 = response(b,a) OR response(a,c)* può essere inoltrata al ciclo esterno e aggiunta al modello precedente. Adesso tutte le tracce negative sono escluse dal modello, quindi, l'algoritmo termina e restituisce la seguente espressione CNF:

```
(response(b,a) OR response(a,c)) AND condition(a,b)
```

3.4 Confronto tra algoritmi di mining

In questa sezione viene presentato un confronto tra i quattro algoritmi di mining precedentemente citati.

In particolare vengono analizzate le principali differenze funzionali e prestazionali. Gli algoritmi presi in considerazione sono l'algoritmo che lavora con vincoli DCR, analizzato nel capitolo 3, l'algoritmo di Elena Palmieri [13], che lavora con vincoli Declare, e i due algoritmi di mining DCR che lavorano esclusivamente con tracce positive, Parnek e UlrikHovsgaard [12].

3.4.1 Differenze funzionali

Una principale differenza sta nei tipi di log che vengono utilizzati dall'algoritmo. Infatti, Parnek e UlrikHovsgaard usano dei log con solo tracce positive, di conseguenza, non discriminano i comportamenti desiderati da quelli indesiderati.
Questo può andar bene se un'azienda è brava ad avere processi completamente corretti, dove ogni azione è voluta. Nella realtà questo non accade; non è raro, infatti, imbattersi in situazioni impreviste che vengono catalogate come azioni non volute all'interno del processo, per esempio, l'arrivo in ritardo di un pacco Amazon.
L'algoritmo su DCR e quello su Declare sono molto simili tra di loro, la differenza principale sta nella semantica dei vincoli. In modo particolare, Declare utilizza dei vincoli che sono combinazione di vincoli più semplici e viceversa, cioè esiste un vincolo *v1* che è più specifico di un vincolo *v2* (al contrario esiste un vincolo *v3* che è più generale di *v4*). DCR non presenta questa caratteristica, infatti, utilizza cinque vincoli diversi tra loro. Questo è il motivo per cui viene utilizzato un diverso criterio di creazione del termine da aggiungere al modello. Si prenda in considerazione la versione DNF dell'algoritmo, in DCR la specializzazione avviene semplicemente aggiungendo vincoli, in AND, al termine, mentre in Declare avviene rimuovendo un vincolo e aggiungendone uno più specifico.
Un'altra differenza sta nella composizione del modello risultante, infatti, l'algoritmo UlrikHovsgaard parte da un modello vincolato e, analizzando il log, va a rimuovere

quei vincoli che escludono delle tracce. Avere un modello inizialmente vincolato significa avere un modello con tanti vincoli e di difficile lettura. Gli altri tre algoritmi, invece, partono da un modello vuoto, non vincolato, e vanno ad aggiungere vincoli al modello fin quando tutte le tracce positive vengono coperte e, nel caso dei due algoritmi con tracce negative, quelle negative vengono escluse.

3.4.2 Differenze prestazionali

Per confrontare le prestazioni di questi algoritmi sono stati utilizzati dei log di sole tracce positive, a causa dell'incompatibilità degli algoritmi Parnek e UlrikHovsgaard con log che contengono anche tracce negative. I test effettuati su un log composto da 64000 tracce positive hanno riscontrato che l'algoritmo Parnek è il più veloce, in quanto impiega 6,6802 secondi per trovare il modello. L'algoritmo DNF su DCR impiega 254,641 secondi, mentre l'algoritmo DNF su Declare ne impiega 1408,148 per ottenere il modello richiesto. La versione CNF dell'algoritmo su DCR impiega 259,023 secondi mentre quella su Declare ne impiega 4177,542.
L'algoritmo su Declare è sicuramente molto più lento dell'algoritmo su DCR. La causa principale è il numero di vincoli Declare che è maggiore dei vincoli di DCR, dunque, nel processo di scelta del vincolo da aggiungere al modello le 16 attività presenti nel log devono essere combinate con tutti i vincoli Declare e non con 5, con conseguente aumento del tempo di esecuzione. L'algoritmo UlrikHovsgaard non è stato testato con tale log a causa della sua incompatibilità; dunque, è stato effettuato un ulteriore test usando un log compatibile [15]. Tale log è costituito da 1153 tracce positive, ma ogni traccia contiene un numero eccessivamente elevato di eventi; pertanto, non è stato possibile effettuare un confronto con gli algoritmi che lavorano su DCR e su Declare in quanto, durante il test, è stato superato il limite massimo di memoria disponibile. L'algoritmo su DCR ha un tempo di esecuzione superiore rispetto a Parnek o Ul-

rikHovsgaard in quanto, in questo lavoro di tesi, non è stata effettuata nessuna ottimizzazione.

4 Risultati sperimentali

L'algoritmo è stato testato su diversi log di eventi per valutare sia la correttezza dei modelli scoperti che le performance. I parametri presi in considerazione nella valutazione delle prestazioni dell'algoritmo sono il tempo di esecuzione, ossia il tempo che impiega l'algoritmo per ottenere il modello adatto a quel dataset specifico, e l'occupazione dello stack locale e globale. Gli esperimenti sono stati effettuati su tre tipi di log:

- Log generati artificialmente da un modello di processo noto.
- Log su un caso di studio reale, costituito da tracce positive e negative.
- Log su un caso di studio reale, costituito solo da tracce positive.

4.1 Log di eventi controllato

Questo log contiene tracce che sono state generate artificialmente partendo da un modello di processo conosciuto, costituito da vincoli Declare. Tale modello è mostrato in Fig. 18. Ci sono 64000 tracce positive e tre differenti insiemi di tracce negative, ciascuno dei quali con un numero diverso di tracce.
Le attività presenti in questo log sono 16 e sono elencate di seguito:

- appraise_property
- approve_application
- ask_for_customer_feedback
- assess_eligibility
- assess_loan_risk

- cancel_application
- check_credit_history
- check_income_sources
- notify_approval
- notify_cancellation
- receive_loan_application
- receive_negative_feedback
- receive_positive_feedback
- reject_application
- send_acceptance_pack
- verify_receipt

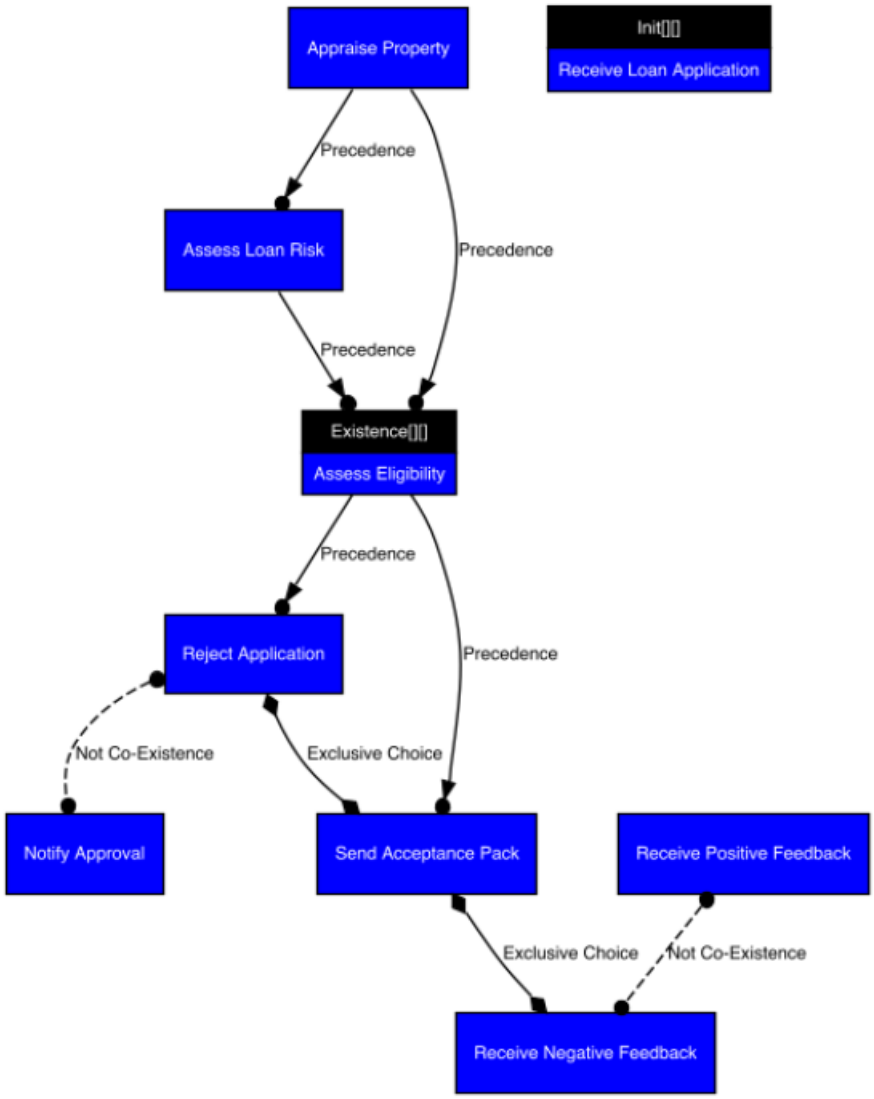


Figura 18: Modello di processo da cui sono state generate le tracce positive e negative [13]

Nei paragrafi successivi vengono mostrati i modelli ottenuti utilizzando i vari insiemi di tracce negative. Tutti i modelli sono stati verificati osservando le singole tracce che compongono quello specifico log.

4.1.1 Primo insieme di tracce negative

Questo insieme contiene 25600 tracce che vengono escluse dal vincolo *condition (assess_loan_risk, assess_eligibility)*. Testando il log su entrambe le versioni dell'algoritmo (DNF e CNF) è possibile notare che generano lo stesso modello:

```
condition(assess_loan_risk,assess_eligibility)
```

Questo significa che tale vincolo esclude tutte le tracce negative e copre tutte quelle positive. Escludere tutte le tracce negative significa, secondo la semantica del vincolo *condition*, che all'interno di ogni traccia è presente *assess_eligibility* ma in precedenza non era presente l'esecuzione di *assess_loan_risk*.
Coprire tutte le tracce positive significa, invece, per ogni traccia, o che l'evento *assess_eligibility* è preceduto dall'esecuzione dell'evento *assess_loan_risk* o che *assess_eligibility* non è presente nella traccia in questione. La versione DNF è leggermente più veloce della versione CNF, infatti, la prima impiega 378,540 secondi per ottenere il modello, mentre la seconda ne impiega 390,198.

4.1.2 Secondo insieme di tracce negative

Il secondo insieme contiene 10240 tracce negative. Eseguendo entrambi gli algoritmi con questo log di eventi si ottiene, per entrambi gli algoritmi, DNF e CNF, lo stesso modello:

```
inclusion(receive_negative_feedback,reject_application) OR
inclusion(reject_application,receive_negative_feedback)
```

Il vincolo *inclusion(reject_application,receive_negative_feedback)* esclude tutte le tracce negative ma copre solo 63968 tracce positive su 64000. Di conseguenza, l'algoritmo cerca un altro termine che escluda nuovamente tutte le tracce negative e

copra le restanti 32 tracce positive, cioè il vincolo *inclusion(receive_negative_feedback, reject_application)*. Per escludere una traccia negativa è necessario che entrambi i due *inclusion* siano violati, in quanto essi sono legati, in OR, nel modello. Un vincolo *inclusion* può essere violato in due modi:

1. È presente l'evento *reject_application* (*receive_negative_feedback*) e successivamente non è presente l'evento *receive_negative_feedback* (*reject_application*).
2. È presente l'evento *receive_negative_feedback* (*reject_application*) ma non è presente, in precedenza, l'evento *reject_application*(*receive_negative_feedback*).

Per coprire tutte le tracce positive è, invece, necessario che, per ogni traccia, almeno uno dei due vincoli che compongono il modello sia soddisfatto (in quanto il modello è composto dai due vincoli in OR); pertanto, o sono presenti entrambi gli eventi *reject_application* e *receive_negative_feedback* (l'ordine, in questo caso, è irrilevante in quanto le due inclusion contengono i due eventi scambiati) oppure nessuno dei due è presente.
Anche in questo caso, la versione DNF dell'algoritmo è più veloce della versione CNF, con un tempo di esecuzione pari a 352.992 secondi, per il primo, e 361.008 secondi, per il secondo.

4.1.3 Terzo insieme di tracce negative

Questo insieme contiene 12800 tracce che vengono escluse dal vincolo *response(appraise_property, assess_loan_risk)*. Testando il log su entrambi gli algoritmi è possibile notare, anche in questo caso, che generano lo stesso modello:

```
response(appraise_property,assess_loan_risk)
```

Questo significa che tale vincolo esclude tutte le tracce negative e copre tutte quelle positive. Escludere tutte le tracce negative significa, secondo la semantica del

vincolo *response*, che all'interno di ogni traccia è presente *appraise_property* ma negli istanti di tempo successivi non avviene l'esecuzione di *assess_loan_risk*.
Coprire tutte le tracce positive significa, invece, per ogni traccia, o che l'esecuzione di *appraise_property* è seguita dall'esecuzione dell'evento *assess_loan_risk* o che l'evento *appraise_property* non è presente all'interno della traccia. La versione DNF impiega 322,573 secondi per ottenere il modello mentre la versione CNF ne impiega 327,178.

4.1.4 Performance

Le prestazioni dell'algoritmo, nelle due versioni, sono riportate nella tabella sottostante per ogni log con cui è stato testato.

	Tempo di esecuzione	**Stack locale**	**Stack globale**
Primo test DNF	378.540	2084728	1073737712
Primo test CNF	390.198	1036152	134213616
Secondo test DNF	352.992	2084728	268431344
Secondo test CNF	361.008	1036152	134213616
Terzo test DNF	322.573	2084728	268431344
Terzo test CNF	327.178	1036152	134213616

Tabella 1: Tempi di esecuzione e occupazione dello stack locale e globale

Come si nota, l'algoritmo DNF è leggermente più veloce dell'algoritmo CNF. I tre test sono stati effettuati su log costituiti dallo stesso insieme di tracce positive, in cui sono presenti 64000 tracce e da un diverso insieme di tracce negative.
Il primo test ha un tempo di esecuzione maggiore rispetto agli altri, questo è dovuto al fatto che le tracce negative sono 25600, circa il doppio del numero di tracce negative presenti negli altri log.
Il secondo log contiene un insieme di 10240 tracce negative mentre l'ultimo log ne

contiene 12800. Seguendo il ragionamento precedente, il secondo test sarebbe dovuto essere più veloce del terzo, a causa del numero minore di tracce, ma quei pochi secondi in più che si ritrova sono dovuti al fatto che, durante il secondo test, l'algoritmo va alla ricerca di un secondo vincolo da aggiungere al modello per coprire alcune tracce positive che il primo vincolo non era riuscito a soddisfare.

Per misurare la memoria in uso nello stack locale e globale è stato utilizzato il predicato prolog *statistics(local, X1)* e *statistics(global, X2)* all'inizio e alla fine dell'algoritmo. In questo modo è stato possibile vedere la quantità di memoria utilizzata prima e dopo l'esecuzione dell'algoritmo. La misurazione iniziale ha restituito un valore di stack locale e globale uguale per tutti i test effettuati con un valore, rispettivamente, di 53112 e di 258032 bytes. I valori finali, invece, sono mostrati nella tabella 1.

4.2 Log di eventi relativo al Pap Test

Questo log contiene attività che consentono il rilevamento dell'infezione papilloma virus nelle donne con età superiore ai 25 anni. Le attività presenti sono 19 e sono elencate di seguito:

- execute_biopsy_exam
- execute_colposcopy_exam
- execute_papTest_exam
- send_biopsy_sample
- send_papTest_sample
- send_letter_negative_biopsy

- send_letter_negative_colposcopy
- send_letter_negative_papTest
- send_result_doubt_colposcopy
- send_result_inadequate_papTest
- send_result_negative_biopsy
- send_result_negative_colposcopy
- send_result_negative_papTest
- send_result_positive_biopsy
- send_result_positive_papTest
- invite
- refuse
- phone_call_positive_biopsy
- phone_call_positive_papTest

Come il precedente, anche questo log contiene sia tracce positive che negative ma in numero nettamente inferiore; infatti, il log è costituito da 55 tracce positive e da 102 tracce negative. Dato il numero molto basso di tracce, i tempi di esecuzione dell'algoritmo sono molto inferiori rispetto al precedente test sul log generato artificialmente. Nello specifico, la versione DNF dell'algoritmo ottiene il modello richiesto in 21,041 secondi mentre la versione CNF ne impiega 22,413.
La differenza in termini di velocità di esecuzione dell'algoritmo nelle due versioni,

meno evidente a causa del basso numero di tracce, è evidente anche in questo test; infatti, la versione DNF continua ad essere più veloce. Il modello generato nella versione DNF è:

```
response(invito,rifiuto) OR
(response(eseguiEsame_biopsia,invioRisultato_positivobiopsia) AND
response(invito,eseguiEsame_colposcopia)) OR
(exclusion(eseguiEsame_papTest,eseguiEsame_papTest) AND
inclusion(invioRisultato_positivopapTest,invioLetteraNegativaBiopsia)
AND response(invito,eseguiEsame_papTest))
```

mentre quello generato dalla versione CNF è:

```
response(invioCampione_biopsia,eseguiEsame_biopsia) AND
exclusion(eseguiEsame_papTest,eseguiEsame_papTest) AND
(response(eseguiEsame_papTest,eseguiEsame_colposcopia) OR
inclusion(invioRisultato_positivopapTest,invioLetteraNegativaBiopsia))
AND (response(invito,rifiuto) OR response(invito,eseguiEsame_papTest))
```

Per stabilire la correttezza dei modelli ottenuti è stato necessario osservare le tracce e verificare che tutte quelle positive siano state coperte e quelle negative siano state escluse dal modello. Anche in questo caso, la misurazione iniziale ha ottenuto un valore di stack locale e globale pari a 53112 e 258032 bytes, rispettivamente. Dopo l'esecuzione dell'algoritmo i valori aggiornati di stack locale e globale sono uguali, per entrambe le versioni, a 2084728 e 2093040, rispettivamente.

4.3 Log con solo tracce positive

Come abbiamo visto nel paragrafo 3.4.2 l'algoritmo può essere applicato anche su log con sole tracce positive. Il log utilizzato è *Hospital Billing* [16] che è stato ottenuto

dai moduli finanziari del sistema ERP di un ospedale regionale. Il log contiene 100000 tracce, con eventi correlati alla fatturazione dei servizi medici forniti dall'ospedale. Ogni traccia registra le attività eseguite per fatturare un pacchetto di servizi medici che sono stati raggruppati insieme, ma non contiene informazioni sui servizi medici effettivi forniti dall'ospedale. Il log contiene 18 attività, elencate di seguito:

- BILLED
- CHANGE DIAGN
- CHANGE END
- CODE ERROR
- CODE NOK
- CODE OK
- DELETE
- EMPTY
- FIN
- JOIN-PAT
- MANUAL
- NEW
- REJECT
- RELEASE
- REOPEN

- SET STATUS
- STORNO
- ZDBC_BEHAN

La versione DNF impiega 298,945 secondi restituendo il seguente modello:

```
response(CODE ERROR,CODE NOK)
```

Tale risultato significa che in ogni traccia o è presente *CODE ERROR* seguito da *CODE NOK*, o che *CODE ERROR* non è presente.
Questo è il modello restituito anche dalla versione CNF che, invece, impiega un tempo di esecuzione pari a 309.813 secondi. L'occupazione dello stack locale e globale, iniziale e finale, è uguale per entrambe le versioni. Lo stack locale assume un valore iniziale di 53112 e finale di 2084728, mentre quello globale assume un valore iniziale di 258032 e finale di 134213616. La scelta del vincolo viene fatta contando il numero di tracce che sono conformi al vincolo, che chiamiamo *guadagno*, e scegliendo quello che copre il maggior numero di tracce. Non avendo tracce negative, la condizione di terminazione dell'algoritmo diventa l'insieme dei positivi vuoto, infatti, l'algoritmo andrà alla ricerca di vincoli, aggiunti in OR al modello, fin quando tutte le tracce positive saranno coperte.

4.4 Considerazioni finali

In questo capitolo vengono mostrati i risultati ottenuti dall'esecuzione dell'algoritmo su dei log con un diverso numero di tracce. In modo particolare, è stato effettuato un test su un log controllato, generato artificialmente, costituito da un numero di tracce positive e negative elevato, un test su un caso di studio reale costituito da un numero di tracce positive e negative relativamente basso e un test su un log

composto esclusivamente da tracce positive. Un riassunto delle prestazioni dei vari test è mostrato in tabella 2:

Log	**Tempistiche DNF(s)**	**Tempistiche CNF(s)**	**N° di tracce**
1° test log controllato	378.540	390.198	64000 + 25600
2° test log controllato	352.992	361.008	64000 + 10240
3° test log controllato	322.573	327.178	64000 + 12800
Pap Test	21.041	22.413	55 + 102
Hospital Billing	298.945	309.813	100000

Tabella 2: Tempi di esecuzione dell'algoritmo su diversi log

La tabella è composta da quattro colonne, la prima mostra il log su cui sono stati effettuati i test, la seconda e la terza mostrano, rispettivamente, il tempo di esecuzione dell'algoritmo nella versione DNF e nella versione CNF; l'ultima colonna contiene, invece, il numero di tracce presenti nei log, nello specifico, il primo addendo è il numero di tracce positive e il secondo rappresenta il numero di tracce negative. Il numero di tracce del log *Hospital Billing* contiene solo un addendo in quanto possiede solo tracce positive.

Dalla tabella è possibile notare come l'algoritmo DNF sia più veloce della versione CNF; la differenza è, tuttavia, minima in quanto non esiste una gerarchia di vincoli come in Declare. Com'è possibile notare, il test sul log *Hospital Billing* ha un tempo di esecuzione inferiore rispetto al test sul log controllato, nonostante abbia un numero superiore di tracce. Il motivo è che l'algoritmo lavora esclusivamente su tracce positive, pertanto, non impiega tempo ad escludere le tracce negative, infatti, l'obiettivo finale è trovare un modello che copra tutte le tracce positive.

5 Estensione del modello

L'algoritmo implementato in questo lavoro di tesi è differente rispetto ai precedenti algoritmi di mining che lavorano con DCR. Infatti, per generare il modello, l'algoritmo va alla ricerca del vincolo migliore da aggiungere al termine o alla clausola e, non appena tutte le tracce positive sono coperte e quelle negative sono escluse, termina. Questo significa che il modello generato dall'algoritmo è un modello "minimo", cioè un modello costituito dal minor numero di vincoli. Se da un lato un modello con pochi vincoli è semplice da leggere e da gestire, dall'altro fornisce una limitata descrizione del processo. Di conseguenza, si potrebbe estendere tale modello in modo tale da aggiungere informazioni che descrivono meglio il processo.
L'estensione è stata applicata solo alla versione DNF. Supponiamo di avere un log costituito da P tracce positive e N tracce negative e che il modello restituito dall'algoritmo sia costituito da n termini in OR (t1 OR t2... OR tn). Questo significa che ogni *ti* copre alcune delle tracce positive ed esclude tutte quelle negative.
Per capire meglio il concetto, si supponga che il modello sia costituito da *isole* (insieme di vincoli in AND) legate in OR tra loro e che su ogni isola ci abitino solo quelle persone che rispettano le regole (vincoli) di quell'isola. Se un'isola contiene un numero molto elevato di abitanti deve essere descritta in modo più dettagliato rispetto ad una che ne contiene un numero basso; infatti, quell'isola potrebbe contenere persone, cani, gatti, piante ecc e avere, ad esempio, la sola regola "esseri viventi", il che potrebbe essere più limitante rispetto a un modello costituito da esseri viventi, persone, animali ecc, cioè che contiene più informazioni sugli abitanti di quell'isola.

Un'idea per estendere il modello potrebbe essere quella di suddividere l'insieme dei positivi, secondo un certo criterio, e trovare altri vincoli da aggiungere al modello per descrivere meglio i positivi. Un primo criterio di suddivisione è il calcolo della funzione di entropia degli alberi decisionali, adattata per il presente scopo.

L'ipotesi iniziale era quella di utilizzare tale funzione come criterio per suddividere, in modo bilanciato, le tracce positive (entropia minima).
Tuttavia, in un secondo momento, si è capito che avere degli insiemi sbilanciati fosse un'opzione migliore dal momento che sarebbe stato possibile descrivere quello contenente il maggior numero di tracce (entropia massima).
Un altro criterio consiste nel suddividere l'insieme delle tracce positive in base al numero di tracce che vengono coperte dalle isole del modello, cioè suddividere l'insieme dei positivi in un insieme *p1*, detto dei "finti positivi", contenente le tracce relative all'isola che copre il maggior numero di tracce e l'insieme *p2*, detto dei "finti negativi", contenente le restanti tracce.
Entrambi gli approcci necessitano di calcolare, per ogni vincolo, le tracce coperte da esso; pertanto, a parità di costo computazionale, si è scelto di utilizzare il secondo approccio dal momento che parte delle funzioni necessarie, essendo state utilizzate nella versione originale dell'algoritmo di Mooney, erano già a disposizione.

Una volta separati i due insiemi, l'algoritmo tiene conto degli elementi contenuti al loro interno e verifica se essi sono bilanciati o meno. Se sono sbilanciati, cioè il numero di tracce dell'insieme dei finti positivi è maggiore del 65% del numero totale di tracce positive, allora significa che l'insieme *p1* contiene un numero di tracce molto maggiore di *p2*, di conseguenza si andrà alla ricerca di un nuovo vincolo *v*, non presente nel modello, che descriva maggiormente l'insieme *p1*.
Se tale vincolo copre tutte le tracce presenti nell'insieme dei finti positivi, allora esso sarà aggiunto in AND all'isola dominante, se invece, anche una sola traccia non è coperta da tale vincolo, viene creata una nuova isola costituita dall'isola dominante e dal nuovo vincolo (i3 = i1 AND v), aggiunta in OR al modello precedente.
In quest'ultimo caso le tracce non coperte vengono rimosse dall'insieme dei finti positivi e inseriti nell'insieme dei finti negativi, cioè ad ogni iterazione, i due insiemi vengono aggiornati. Se sono bilanciati, cioè il numero di tracce dell'insieme *p1* è

compreso tra il 50% e il 65% del numero totale di tracce positive, significa che i due insiemi contengono più o meno lo stesso numero di tracce; conseguentemente, l'algoritmo andrà ad aggiungere due isole al modello, la prima costituita dai vincoli che coprono il maggior numero di tracce di *p1* ed escludono tutte le tracce di *p2* e la seconda costituita da tutti i vincoli che coprono il maggior numero di tracce di *p2* ed escludono tutte le tracce di *p1*.
L'algoritmo può terminare in 3 modi:

1. L'insieme dei finti positivi *p1* contiene un numero di tracce inferiore al 50% del numero totale di tracce positive, dunque, sarebbe inutile andare a descrivere maggiormente tale numero limitato di tracce.

2. Non esiste più un vincolo da aggiungere al modello, in quanto tutti i vincoli sono già presenti al suo interno. Un vincolo che non copre alcuna traccia non può essere scelto.

3. È stato raggiunto il numero massimo di iterazioni consentite.

5.1 Casi studio

In questa sezione verranno mostrati dei possibili casi studio in modo da comprendere il comportamento dell'algoritmo in base al modello iniziale ottenuto.
Il log è costituito da 100 tracce positive, il numero di tracce negative non è rilevante per l'estensione del modello.

1. **Primo caso**: il modello restituito è composto da una sola isola *i1*. In questo caso l'isola che copre il maggior numero di tracce positive è, ovviamente, *i1*, di conseguenza, l'insieme dei finti positivi *p1* conterrà le 100 tracce, mentre l'insieme dei finti negativi *p2* sarà vuoto.

L'algoritmo andrà alla ricerca di un vincolo *v1*, non presente nel modello, e lo aggiungerà all'isola *i1* se copre tutte le 100 tracce contenute in *p1*, mentre creerà la nuova isola *i2 = i1 AND v1*, aggiunta in OR al modello, se almeno una traccia viene esclusa da tale vincolo. In quest'ultimo caso il modello alla prima iterazione sarà *i1 OR i2* e i due insiemi saranno aggiornati.

Con l'aumentare delle iterazioni, l'insieme dei finti positivi conterrà un numero minore di tracce in quanto, con l'aggiunta di vincoli, alcune tracce verranno escluse e passeranno nell'insieme dei finti negativi.

L'algoritmo proseguirà fin quando o tutti i vincoli sono stati aggiunti al modello o l'insieme *p1* conterrà un numero di tracce inferiore al 50% delle tracce totali o se è stato raggiunto il numero massimo di iterazioni.

2. **Secondo caso**: il modello restituito è costituito da due isole (i1 OR i2). Si supponga che l'isola *i1* copra un numero di tracce positive maggiore rispetto all'isola *i2*. Pertanto, le tracce coperte da *i1* vengono inserite nell'insieme dei finti positivi, mentre le restanti tracce vengono inserite nell'insieme dei finti negativi.

   Se i due insiemi sono sbilanciati, ad esempio, l'insieme *p1* contiene 80 tracce e l'insieme *p2* ne contiene 20, l'algoritmo trova un nuovo vincolo *v*, non presente nel modello; se tutte le 80 tracce vengono coperte da tale vincolo, allora esso viene aggiunto in AND all'isola *i1*, altrimenti viene aggiunta al modello una nuova isola *i3* costituita dell'isola *i1* e dal vincolo *v* (i3 = i1 AND v). Ipotizzando che le tracce escluse da *v* siano 20, queste vengono rimosse da *p1* e inserite in *p2*. In quest'ultimo caso, l'insieme dei finti positivi è costituito da 60 tracce, mentre quello dei finti negativi ne contiene 40.

   Ora, i due insiemi sono bilanciati e l'algoritmo aggiungerà due nuove isole al modello: la prima *i4* costituita da un insieme di vincoli che escluda tutte tracce

contenute nell'insieme dei finti negativi e copra il numero maggiore di tracce nell'insieme dei finti positivi e la seconda *i5* costituita da un insieme di vincoli che escluda tutte le tracce contenute nell'insieme dei finti positivi e copra il numero maggiore di tracce nell'insieme dei finti negativi.
L'algoritmo procede fin quando si verifica una delle tre condizioni di terminazione.

3. **Terzo caso**: il modello restituito è costituito da n isole aventi circa lo stesso numero di tracce positive coperte. Si supponga di avere quattro isole e che ognuna copra 25 tracce positive su 100. L'algoritmo dividerà l'insieme dei positivi scegliendo una delle 4 isole e inserendo le 25 tracce nell'insieme dei finti positivi e le restanti 75 nell'insieme dei finti negativi.
In questo scenario, l'algoritmo termina alla prima iterazione in quanto l'insieme *p1* contiene un numero di elementi inferiore al 50% del numero totale di tracce positive, restituendo lo stesso modello iniziale. Il modello non viene esteso poiché ogni isola copre un numero basso di tracce, pertanto, le informazioni a disposizione sono sufficienti per la descrizione del processo.

5.2 Esempio

In questa sezione verrà presentato un esempio partendo da un modello con una sola isola. I parametri presi in considerazione sono:

- Numero massimo di iterazioni: 20.
- Soglia minima: 50%.
- Soglia massima: 65%.

Il log utilizzato è costituito da un numero molto basso di tracce, in modo da seguire, facilmente, tutti i passi intrapresi dall'algoritmo:

Tracce positive:

```
traccia(id1, [evento(a,1),evento(c,2),evento(b,3)])
traccia(id2, [evento(a,1),evento(b,3)])
traccia(id3, [evento(a,3),evento(b,4),evento(c,6)])
traccia(id4, [evento(a,1),evento(c,2),evento(b,4)])
traccia(id5, [evento(a,1),evento(b,3),evento(b,6)])
traccia(id6, [evento(a,3),evento(b,4),evento(c,6),evento(a,8)])
```

Tracce negative:

```
traccia(id7, [evento(a,1),evento(c,2)])
traccia(id8, [evento(c,3),evento(b,4),evento(a,5)])
```

Il modello iniziale restituito dall'algoritmo è *response(a,b)*. Tale vincolo copre tutte le tracce positive ed esclude tutte quelle negative.

A questo punto, inizia il procedimento di estensione del modello, aggiungendo vincoli in modo da descrivere maggiormente i positivi. Innanzitutto, l'algoritmo cerca l'isola del modello che copra il maggior numero di vincoli, in questo caso l'unica presente (*response(a,b)*), e inserisce nell'insieme dei finti positivi tutte le tracce contenute nell'insieme dei positivi; l'insieme dei finti negativi risulterà, quindi, vuoto.

Essendo i due insiemi sbilanciati, l'algoritmo andrà alla ricerca di un nuovo vincolo, non presente nel modello, analizzando solo l'insieme dei finti positivi. Il vincolo trovato è *condition(a,b)* che, come si può notare, copre tutte le 6 tracce contenute in *p1*; pertanto, esso viene aggiunto in AND all'isola precedente.

Al passo 1 otteniamo il seguente modello:

```
response(a,b) AND condition(a,b)
```

Fino al passo 6 l'algoritmo continuerà ad aggiungere vincoli in AND all'isola generata al passo precedente poiché tutti i vincoli trovati coprono tutte le 6 tracce presenti in *p1*.
Il modello ottenuto al passo 6 è:

```
response(a,b) AND condition(a,b) AND condition(a,c) AND
milestone(a,b) AND milestone(a,c) AND inclusion(a,b) AND
exclusion(c,c)
```

Al passo 7 il vincolo restituito dall'algoritmo è *exclusion(a,a)* che però copre 5 delle 6 tracce positive. Gli insiemi dei finti positivi e dei finti negativi diventano quindi:

```
p1 = {id1, id2, id3, id4, id5}
p2 = {id6}
```

Il numero di elementi di *p1* è ancora maggiore del 65% del numero totale di tracce positive, quindi l'algoritmo continuerà a descrivere le tracce contenute nell'insieme dei finti positivi. Tale vincolo non può essere aggiunto in AND all'isola precedente in quanto, se così fosse, la traccia *id6* non verrebbe più coperta dal modello. Dunque, si andrà a creare un'altra isola, costituita dall'isola generata al passo 6 e dal nuovo vincolo trovato, aggiunta in OR all'isola precedente. Il modello ottenuto al passo 7 sarà:

```
(response(a,b) AND condition(a,b) AND condition(a,c) AND
milestone(a,b) AND milestone(a,c) AND inclusion(a,b) AND
exclusion(c,c)) OR
(response(a,b) AND condition(a,b) AND condition(a,c) AND
milestone(a,b) AND milestone(a,c) AND inclusion(a,b) AND
exclusion(c,c) AND exclusion(a,a))
```

Ricapitolando, la prima isola copre tutte le 6 tracce positive mentre la seconda ne copre solo 5 in quanto *id6* non è coperta da *exclusion(a,a)*. Nell'iterazione successiva, l'algoritmo cercherà un vincolo che copra il maggior numero di tracce presenti nel nuovo insieme *p1* (che adesso contiene 5 tracce). Si prosegue in questo modo, aggiungendo vincoli in AND all'isola se tutte le tracce di *p1* sono coperte, oppure, in caso contrario, aggiungendo una nuova isola in OR alle precedenti; tutto ciò fino al passo 11:

```
(response(a,b) AND condition(a,b) AND condition(a,c) AND
milestone(a,b) AND milestone(a,c) AND inclusion(a,b) AND
exclusion(c,c)) OR
(response(a,b) AND condition(a,b) AND condition(a,c) AND
milestone(a,b) AND milestone(a,c) AND inclusion(a,b) AND
exclusion(c,c) AND exclusion(a,a)) OR
(response(a,b) AND condition(a,b) AND condition(a,c) AND
milestone(a,b) AND milestone(a,c) AND inclusion(a,b) AND
exclusion(c,c) AND exclusion(a,a) AND exclusion(b,a) AND
exclusion(c,a)) OR
(response(a,b) AND condition(a,b) AND condition(a,c) AND
milestone(a,b) AND milestone(a,c) AND inclusion(a,b) AND
exclusion(c,c) AND exclusion(a,a) AND exclusion(b,a) AND
exclusion(c,a) AND response(c,b) AND exclusion(b,c))
```

```
p1 = {id1, id2, id4, id5}
p2 = {id6, id3}
```

Al passo 12, il vincolo trovato è *exclusion(b,b)* che copre 3 delle 4 tracce contenute in *p1*; pertanto, una nuova isola è aggiunta al modello.

Al passo 13 l'insieme dei finti positivi contiene un numero di tracce pari al 50% (3) del numero totale di tracce. Essendo i due insiemi bilanciati, l'algoritmo procede aggiungendo due nuove isole al modello, una costituita dai vincoli che coprono il maggior numero di tracce di *p1* ed escludono tutte le tracce di *p2*, *condition(c,b)*, e l'altra che, viceversa, contiene i vincoli che coprono il maggior numero di tracce di *p2* ed escludono tutte le tracce di *p1*, *response(b,c)*.
Il modello al passo 13 diventa:

```
(response(a,b) AND condition(a,b) AND condition(a,c) AND
milestone(a,b) AND milestone(a,c) AND inclusion(a,b) AND
exclusion(c,c)) OR
(response(a,b) AND condition(a,b) AND condition(a,c) AND
milestone(a,b) AND milestone(a,c) AND inclusion(a,b) AND
exclusion(c,c) AND exclusion(a,a)) OR
(response(a,b) AND condition(a,b) AND condition(a,c) AND
milestone(a,b) AND milestone(a,c) AND inclusion(a,b) AND
exclusion(c,c) AND exclusion(a,a) AND exclusion(b,a) AND
exclusion(c,a)) OR
(response(a,b) AND condition(a,b) AND condition(a,c) AND
milestone(a,b) AND milestone(a,c) AND inclusion(a,b) AND
exclusion(c,c) AND exclusion(a,a) AND exclusion(b,a) AND
exclusion(c,a) AND response(c,b) AND exclusion(b,c)) OR
(response(a,b) AND condition(a,b) AND condition(a,c) AND
milestone(a,b) AND milestone(a,c) AND inclusion(a,b) AND
exclusion(c,c) AND exclusion(a,a) AND exclusion(b,a) AND
exclusion(c,a) AND response(c,b) AND exclusion(b,c)) OR
(response(a,b) AND condition(a,b) AND condition(a,c) AND
```

```
milestone(a,b) AND milestone(a,c) AND inclusion(a,b) AND
exclusion(c,c) AND exclusion(a,a) AND exclusion(b,a) AND
exclusion(c,a) AND response(c,b) AND exclusion(b,c) AND
exclusion(b,b)) OR
condition(c,b) OR
response(b,c)

p1 = {id1, id2, id4}
p2 = {id6, id3, id5}
```

Il vincolo *condition(c,b)* copre solo 2 delle 3 tracce presenti nell'insieme dei finti positivi, di conseguenza, l'insieme *p1* contiene un numero di elementi inferiore al 50% del numero totale di tracce (6).
L'algoritmo termina restituendo il modello al passo 13 e i seguenti insiemi:

```
p1 = {id1, id4}
p2 = {id6, id3, id5, id2}
```

5.3 Performance

L'estensione del modello permette l'aggiunta di informazioni per garantire una maggiore conoscenza del processo. Questo procedimento è molto costoso in quanto è necessario calcolare, ad ogni iterazione e per ogni vincolo, il numero di tracce soddisfatte. Il tempo di esecuzione dipende dal numero di attività presenti nel log, dal numero di tracce e dal numero di iterazioni che vengono eseguite.
Per verificare la correttezza dell'approccio e vedere le performance, sono stati effettuati dei test su 10 log utilizzati nel *Process Discovery Contest 2019* [17], ciascuno dei quali costituito da 700 tracce positive.
I parametri iniziali utilizzati per il test sono:

- Numero massimo di iterazioni: 30.
- Soglia minima: 50%.
- Soglia massima: 65%.

Per tutti i 10 test effettuati l'algoritmo termina quando vengono raggiunte le 30 iterazioni, in quanto, data la quantità di tracce e attività, esiste un gran numero di vincoli che possono essere aggiunti al modello.
In questo lavoro di tesi non è stata applicata nessuna ottimizzazione, di conseguenza, non è stato possibile testare l'algoritmo su un numero di iterazioni molto più elevato. Eventuali passi di ottimizzazione potranno essere applicati in futuro.
La tabella 3 mostra il modello iniziale e i tempi di esecuzione dell'algoritmo per ogni log. Il modello finale è mostrato di seguito.

Log	**Numero attività**	**Tempistiche DNF (s)**	**Modello iniziale**
1° test	45	1359.491	response(ab,ag)
2° test	46	1475.999	response(a,i)
3° test	48	1118.458	response(h,p)
4° test	34	534.759	response(ac,e)
5° test	44	622.439	response(a,al)
6° test	43	565.475	response(af,g)
7° test	35	525.561	response(al,ao)
8° test	44	871.504	response(ab,l)
9° test	29	1118.720	response(a,aa)
10° test	32	380.642	response(aa,i)

Tabella 3: Modello iniziale e tempo di esecuzione dell'algoritmo.

Modello finale 1° test:

```
response(ab,ag) AND response(al,t) AND response(i,t) AND
response(j,an) AND response(w,u) AND condition(a,aa) AND
condition(a,f) AND  condition(a,l) AND condition(a,w) AND
condition(ae,ah) AND condition(af,ad) AND condition(af,ao) AND
condition(af,v) AND condition(af,y) AND  condition(ag,ac) AND
condition(ag,ak) AND condition(ag,p) AND condition(ag,u) AND
condition(al,ad) AND condition(al,ap) AND condition(al,t) AND
condition(am,ai) AND condition(am,o) AND condition(am,x) AND
condition(ao,v) AND condition(ao,y) AND condition(b,v) AND
condition(c,ac) AND condition(c,ag) AND condition(c,aj) AND
condition(c,ak)
```

Modello finale 2° test:

```
response(a,i) AND response(ab,aq) AND response(ac,i) AND
response(ad,i) AND response(ae,i) AND response(ag,i) AND
response(aj,i) AND  response(ak,i) AND response(ak,q) AND
response(am,i) AND response(an,x) AND response(ap,i) AND
response(ar,i) AND response(as,ad) AND response(as,i) AND
response(at,i) AND response(at,q) AND response(b,ab) AND
response(b,aq) AND response(c,i) AND response(e,i) AND
response(f,ag) AND response(f,ap) AND response(f,i) AND
response(f,k) AND response(f,n) AND response(g,i) AND
response(g,q) AND  response(h,i) AND response(k,ap) AND
response(k,i)
```

Modello finale 3° test:

```
response(h,p) AND response(l,an) AND response(l,q) AND
```

```
response(l,x) AND response(q,an) AND response(q,x) AND
condition(an,al) AND condition(aq,ar) AND condition(b,u) AND
condition(d,ag) AND condition(d,ak) AND condition(d,au) AND
condition(d,m) AND condition(d,n) AND  condition(d,o) AND
condition(d,w) AND condition(e,ai) AND condition(e,at) AND
condition(e,s) AND condition(e,t) AND condition(h,p) AND
condition(i,ad) AND condition(j,af) AND condition(j,ap) AND
condition(j,v) AND condition(l,ab) AND condition(l,al) AND
condition(l,an) AND  condition(l,q) AND condition(l,x) AND
condition(l,z)
```

Modello finale 4° test:

```
response(ac,e) AND response(ac,l) AND response(ae,e) AND
response(ae,l) AND response(ag,e) AND response(ag,l) AND
response(ai,ak) AND  response(ai,v) AND response(aq,k) AND
response(l,e) AND response(n,al) AND response(y,aq) AND
response(y,k) AND condition(a,ab) AND  condition(a,ac) AND
condition(a,ad) AND condition(a,ae) AND condition(a,ag) AND
condition(a,ak) AND condition(a,al) AND condition(a,am) AND
condition(a,ao) AND condition(a,aq) AND condition(a,as) AND
condition(a,at) AND condition(a,au) AND condition(a,b) AND
condition(a,c) AND  condition(a,d) AND condition(a,e) AND
condition(a,f)
```

Modello finale 5° test:

```
response(a,al) AND response(ah,g) AND response(ao,ad) AND
response(ar,al) AND response(b,ai) AND response(b,f) AND
```

```
response(i,ah) AND  response(i,g) AND response(i,s) AND
response(m,ah) AND response(m,g) AND response(m,i) AND
response(m,s) AND response(s,ah) AND  response(s,g) AND
condition(aa,ab) AND condition(ac,ae) AND condition(ac,an) AND
condition(ah,x) AND condition(ai,ad) AND condition(ai,aj) AND
condition(ai,ao) AND condition(ai,aq) AND condition(ai,j) AND
condition(ai,l) AND condition(ai,r) AND condition(am,ar) AND
condition(ao,ad) AND  condition(b,ad) AND condition(b,ai) AND
condition(b,aj)
```

Modello finale 6° test:

```
response(af,g) AND response(af,s) AND response(ag,af) AND
response(ag,aq) AND response(ag,g) AND response(ag,s) AND
response(al,an) AND  response(al,t) AND response(am,t) AND
response(ap,ab) AND response(ap,af) AND response(ap,g) AND
response(ap,s) AND response(aq,af) AND  response(aq,g) AND
response(aq,s) AND response(f,ae) AND response(f,aj) AND
response(f,k) AND response(f,u) AND response(k,aj) AND
response(o,ae) AND response(o,aj) AND response(o,g) AND
response(o,k) AND response(o,u) AND response(o,z) AND
response(s,g) AND  response(x,an) AND response(x,t) AND
condition(a,aa)
```

Modello finale 7° test:

```
response(al,ao) AND response(al,r) AND response(c,o) AND
response(x,ad) AND response(x,b) AND response(x,i) AND
response(x,m) AND  response(y,o) AND response(z,ab) AND
```

```
condition(ac,ah) AND condition(aj,aa) AND condition(aj,ab) AND
condition(aj,aq) AND condition(aj,ar) AND  condition(aj,as) AND
condition(aj,i) AND condition(aj,l) AND condition(ao,aa) AND
condition(ao,as) AND condition(ao,i) AND condition(ar,aq) AND
condition(ar,l) AND condition(b,aa) AND condition(b,as) AND
condition(c,aq) AND condition(c,ar) AND condition(c,l) AND
condition(i,aa) AND condition(i,as) AND condition(l,aq) AND
condition(m,aa)
```

Modello finale 8° test:

```
response(ab,l) AND response(ac,ad) AND response(ac,l) AND
response(ad,l) AND response(aq,l) AND response(e,aq) AND
response(e,l) AND  response(h,ai) AND response(i,an) AND
condition(a,ae) AND condition(a,ai) AND condition(a,al) AND
condition(a,am) AND condition(a,an) AND  condition(a,ao) AND
condition(a,h) AND condition(a,i) AND condition(a,k) AND
condition(a,q) AND condition(a,y) AND condition(ac,ad) AND
condition(ak,t) AND condition(ap,ar) AND condition(b,af) AND
condition(e,aq) AND condition(h,ai) AND condition(i,an) AND
condition(l,aj) AND  condition(l,m) AND condition(o,v) AND
condition(o,x)
```

Modello finale 9° test:

```
response(a,aa) AND response(a,ab) AND response(a,ac) AND
response(a,ai) AND response(a,h) AND response(a,k) AND
response(a,s) AND response(a,z) AND response(aa,ab) AND
response(af,ai) AND response(af,k) AND response(af,z) AND
```

```
response(ah,ai) AND response(ah,aj) AND response(ah,k) AND
response(ah,z) AND response(ai,ac) AND response(ai,z) AND
response(aj,ai) AND response(aj,z) AND response(ak,ab) AND
response(ak,ac) AND response(ak,ai) AND response(ak,d) AND
response(ak,k) AND response(ak,w) AND response(ak,z) AND
response(al,o) AND response(al,w) AND response(al,z) AND
response(am,ai)
```

Modello finale 10° test:

```
response(aa,i) AND response(ab,ae) AND response(ab,ai) AND
response(ab,ao) AND response(ab,aq) AND response(ab,l) AND
response(ab,r) AND response(ai,ao) AND response(ai,l) AND
response(ai,r) AND response(ak,ae) AND response(ak,ai) AND
response(ak,ao) AND response(ak,l) AND response(ak,r) AND
response(aq,ae) AND response(aq,ai) AND response(aq,ao) AND
response(aq,l) AND response(aq,r) AND response(e,ae) AND
response(e,ai) AND response(e,ao) AND response(e,aq) AND
response(e,l) AND response(e,r) AND response(k,ar) AND
response(u,j) AND condition(ab,ai) AND condition(ab,aq) AND
condition(ad,aa)
```

Dato il numero basso di iterazioni non è stato possibile selezionare un vincolo che escluda almeno una delle 700 tracce positive (nessuna nuova isola è stata aggiunta al modello). Avere una sola isola è, ugualmente, un ottimo risultato, in quanto sono state aggiunte delle informazioni che mettono in risalto, in modo più dettagliato, quali abitanti sono presenti sull'isola.

6 Conclusione e lavori futuri

In questo elaborato è stato presentato un algoritmo di discovery che usa vincoli DCR al fine di ottenere un modello di processo. Tale risultato è ottenuto tramite estrazione di informazioni da un log di eventi costituito sia da tracce positive che da tracce negative. L'algoritmo in questione si ispira a quello di Mooney, nelle due versioni DNF e CNF, ed è un'estensione del lavoro svolto da Elena Palmieri [13] che ha implementato tale algoritmo usando vincoli Declare.

La versione DNF è una disgiunzione di congiunzioni, cioè i termini che compongono il modello sono legati in OR mentre i vincoli che compongono i termini sono legati in AND. Al contrario, la versione CNF è una congiunzione di disgiunzioni in quanto le clausole sono collegate in AND tra loro, mentre ogni clausola è costituita da vincoli connessi in OR.

L'obiettivo del presente lavoro è quello di mostrare come avviene la generazione di un modello di un processo, costituito da vincoli DCR, partendo esclusivamente da un log di eventi che contiene tracce positive e negative. L'uso di entrambi gli insiemi è fondamentale, infatti, la possibilità di possedere informazioni desiderate e non, è una caratteristica molto importante che permette di generare modelli che si avvicinino maggiormente alla realtà, poiché nessuna azienda è in grado di generare processi che presentino un comportamento corretto, sempre e ovunque.

Per verificare le prestazioni dell'algoritmo, al variare del numero di tracce contenute nei log, sono stati effettuati dei test.

Inoltre, al fine di fornire maggiori dettagli sul processo, si è deciso di estendere la versione DNF dell'algoritmo aggiungendo vincoli e/o termini al modello.

6.1 Lavori futuri

6.1.1 Ottimizzazione dell'algoritmo

La *choose_constraint*, osservata nel capitolo 3, restituisce il vincolo che copre il maggior numero di tracce positive, escludendo il maggior numero di tracce negative. Dunque, calcola, per ogni vincolo DCR, il guadagno, cioè un valore di bontà del vincolo, e sceglie quello con il valore più alto. Questa funzione continuerà a calcolare il guadagno dei vari vincoli, anche quando avrà trovato il valore di guadagno maggiore, con conseguente aumento del tempo di esecuzione dell'algoritmo.
Una possibile ottimizzazione potrebbe essere quella di impostare una soglia del valore di guadagno in modo tale che, non appena viene trovato un vincolo "buono", questo, semplicemente, venga aggiunto al termine, senza andare a calcolare il guadagno per i restanti vincoli.
Su un log con un numero di tracce molto basso, la differenza non sarebbe molto evidente, ma su un log con un numero elevato di tracce, sicuramente, si avrebbe un notevole miglioramento in termini di prestazioni.

6.1.2 Estensione delle modello nella versione CNF

Nei precedenti capitoli è stato illustrato un modo per estendere il modello ottenuto dalla versione originale dell'algoritmo di Mooney, caratterizzando maggiormente gli esempi positivi. Tale approccio è stato applicato esclusivamente alla versione DNF dell'algoritmo, di conseguenza, si potrebbe estendere anche il modello ottenuto nella versione CNF. Estendere il modello nella versione DNF significa aggiungere termini in OR al modello, mentre nella versione CNF non è possibile utilizzare lo stesso approccio in quanto i termini vengono aggiunti in AND e i vincoli in OR.

6.1.3 Estensione delle tracce

Tutti i test effettuati per validare l'algoritmo utilizzavano dei log costituiti da tracce nella seguente forma:

```
traccia(id_traccia, [evento(nome_attività1,timestamp),
evento(nome_attività2,timestamp),..., evento(nome_attivitàN,timestamp)])
```

Ogni traccia è, dunque, costituita, esclusivamente, dal nome dell'attività e dal timestamp che indica l'istante temporale di inizio attività. Si potrebbe pensare di aggiungere ulteriori attributi all'evento, come ad esempio, il nome dell'esecutore dell'attività o il luogo in cui si verifica. Il numero e il nome degli attributi dipendono dal tipo di attività che compongono le tracce, infatti, le tracce di un log di un'agenzia di trasporti avranno delle attività con un insieme di attributi che sarà diverso dagli attributi di un log di un sito web. Non tutti i log di eventi contengono queste informazioni; infatti, per trarre conclusioni corrette, i dati, per ogni attività, devono essere completi e corretti come descritto nella sezione 1.2.1.

6.1.4 Estensione con i dati

I processi reali fanno, spesso, uso di deadline temporali, cioè presentano attività che devono essere eseguite entro una determinata unità di tempo, altrimenti il vincolo è violato. L'algoritmo implementato in questo lavoro di tesi, invece, non considera alcuna deadline temporale, ma, semplicemente, verifica che un'attività B venga eseguita in un istante di tempo successivo all'esecuzione di A, senza specificare di quanto. Dunque, una possibile estensione dell'approccio potrebbe essere proprio quello del controllo dell'istante temporale di una attività.
Questo è fondamentale in processi in cui sono presenti delle scadenze, quali pagamento bollette, tasse e cosi via.

Ringraziamenti

Un grazie doveroso e sincero al mio Relatore, il Professore Federico Chesani, per avermi accompagnato durante questo lavoro di tesi come costante punto di riferimento in questi mesi di studio.

Un grazie enorme ai miei genitori e a mio fratello, per il sostegno che mi hanno dimostrato e per la grande pazienza che hanno avuto nell'attesa di questo momento.

Un grazie speciale per una persona speciale, Federica, per essere stata al mio fianco sia nei momenti belli ma, specialmente, in quelli bui. Grazie per avermi supportato e aiutato durante questo percorso, se sono arrivato fin qui è anche merito tuo.

Grazie ai miei amici per esserci stati, per avermi aiutato a separare lo studio dallo svago e divertimento.

Bibliografia

[1] Raymond J. Mooney. *Encouraging Experimental Results on Learning CNF*. 1995. URL: https://link.springer.com/content/pdf/10.1023/A:1022659107719.pdf.

[2] Lana Labs. *Process Model*. URL: https://lanalabs.com/en/glossary/process-model/.

[3] Wikipedia. *Log*. URL: https://it.wikipedia.org/wiki/Log.

[4] Wil Van der Aalst. *Process Mining: Discovery, Conformance and Enhancement of Business Processes*. 2011. URL: https://www.researchgate.net/publication/221585990_Process_Mining_Manifesto.

[5] Wil Van der Aalst. *Process Mining towards Semantics*. 1970. URL: https://www.researchgate.net/publication/225774556_Process_Mining_towards_Semantics.

[6] Søren Debois et al. *Declarative Process Mining for DCR Graphs*. URL: https://dl.acm.org/doi/pdf/10.1145/3019612.3019622.

[7] Raghava Rao Mukkamala. *A Formal Model For Declarative Workflows Dynamic Condition Response Graphs*. 2012. URL: https://www.researchgate.net/publication/262379110_A_Formal_Model_For_Declarative_Workflows_Dynamic_Condition_Response_Graphs.

[8] DCR Solutions. *Continuous Process Intelligence is a welcome change*. URL: https://dcrsolutions.net/.

[9] Thomas T. Hildebrandt e Raghava Rao Mukkamala. *Declarative Event-Based Workflow as Distributed Dynamic Condition Response Graphs*. URL: https://arxiv.org/pdf/1110.4161.pdf.

[10] Damián Pérez, Osiel Fundora-Ramírez e Manuel S. Lazo-Cortés. *Recommendation of Process Discovery Algorithms Through Event Log Classification*. 2015. URL: https://www.researchgate.net/publication/300797722_Recommendation_of_Process_Discovery_Algorithms_Through_Event_Log_Classification.

[11] Evelina Lamma et al. *Inducing Declarative Logic-Based Models from Labeled Traces*. URL: https://www.inf.unibz.it/~montali/papers/lamma-etal-BPM2007-mining.pdf.

[12] Tijs Slaats. *Declarative and Hybrid Process Discovery: Recent Advances and Open Challenges*. 2020. URL: https://www.researchgate.net/publication/340034142_Declarative_and_Hybrid_Process_Discovery_Recent_Advances_and_Open_Challenges.

[13] Elena Palmieri. *Learning declarative process models from positive and negative traces*. 2021. URL: https://amslaurea.unibo.it/22501/1/tesi.pdf.

[14] Hajo A. Reijers, Tijs Slaats e Christian Stahl. *Declarative Modeling-An Academic Dream or the Future for BPM?* URL: https://pure.itu.dk/ws/files/78918062/declare_bpm_review.pdf.

[15] P. H. Laursen e K. R. Ulrik. *DCR miner*. URL: https://github.com/Kirluu/UlrikHovsgaardAlgorithm/tree/master/UlrikHovsgaardAlgorithm/UlrikHovsgaardAlgorResources.

[16] Felix Mannhardt (Eindhoven University of Technology). *Hospital Billing - Event Log*. URL: https://doi.org/10.4121/uuid:76c46b83-c930-4798-a1c9-4be94dfeb741.

[17] Josep Carmona, Massimiliano de Leoni e Benoît Depaire. *Process Discovery Contest*. URL: https://data.4tu.nl/articles/dataset/Process_Discovery_Contest_2019/14625996.