**ALMA MATER STUDIORUM - UNIVERSITÀ DI BOLOGNA**

---

**SCUOLA DI INGEGNERIA E ARCHITETTURA**

*DIPARTIMENTO DI INFORMATICA*

*CORSO DI LAUREA MAGISTRALE IN INGEGNERIA INFORMATICA*

**TESI DI LAUREA**

in

Fondamenti di Intelligenza Artificiale M

# Process Mining e Inferenza Causale: Un Approccio per la Modellazione dei Processi Aziendali supportati da Logica Causativa

CANDIDATO:
Andrea Zecchini

RELATORE:
Prof. Ing. Federico Chesani

CORRELATORE:
Luca Giuliani

Anno accademico 2024/2025

Sessione 25/03/2025

# Contenuti

# Abstract

Comprendere e modellare i processi del mondo reale richiede la considerazione della loro complessità intrinseca e delle relazioni causali per migliorare l'analisi e l'ottimizzazione dei processi. In questo lavoro, proponiamo un nuovo approccio che integra il Process Mining con l'Inferenza Causale per scoprire le dipendenze sottostanti nei processi aziendali. La nostra metodologia estende la modellazione dichiarativa dei processi incorporando il ragionamento causale, consentendo la scoperta di vincoli causali e la loro integrazione nei modelli di processo dichiarativi.

In particolare, esploriamo come gli algoritmi di scoperta causale possano essere utilizzati per estrarre relazioni causali significative dai log degli eventi e come queste relazioni possano essere utilizzate per perfezionare i modelli DECLARE. Introduciamo un framework che apprende vincoli di processo basati su strutture causali, migliorando l'adattabilità e l'interpretabilità dei modelli dichiarativi. Il nostro approccio potrebbe consentire alle organizzazioni di ottenere una comprensione più approfondita dei propri processi.

Parole chiave: Declare, Inferenza Causale, Modelli di Processo Dichiarativi, Business Process Model, Log degli Eventi.

# Capitolo 1

# Introduzione

Il campo del Business Process Model (BPM) ha registrato significativi progressi negli ultimi anni, grazie alla crescente disponibilità di dati sugli eventi e alla necessità delle organizzazioni di ottimizzare i propri flussi di lavoro. Le tecniche tradizionali di process mining si concentrano principalmente sull'estrazione di modelli di processo dai log degli eventi utilizzando approcci procedurali o dichiarativi. Tuttavia, questi metodi si basano spesso su tecniche di correlazione, che non riescono a catturare le dipendenze causali che regolano i comportamenti dei processi. Comprendere perché un processo si comporta in un certo modo, piuttosto che limitarsi a descrivere come opera, è cruciale per migliorare il processo decisionale e l'ottimizzazione dei flussi di lavoro.

Il process mining dichiarativo, in particolare i framework come DECLARE , ha guadagnato popolarità grazie alla sua capacità di modellare processi flessibili e poco strutturati. A differenza dei modelli imperativi, che definiscono rigidamente le sequenze di esecuzione, i modelli dichiarativi specificano i vincoli che devono essere rispettati durante l'esecuzione del processo. Tuttavia, i modelli dichiarativi standard mancano di una base causale esplicita, limitando la loro interpretabilità e adattabilità in ambienti dinamici.

Per affrontare questa limitazione, si è deciso di esplorare l'integrazione dell'inferenza causale nel process mining. L'inferenza causale fornisce un approccio rigoroso per identificare le relazioni causa-effetto nei dati di processo, distinguendo tra semplici correlazioni e reali dipendenze. Combinando il process mining con le tecniche di scoperta causale, possiamo costruire modelli dichiarativi informati causalmente, offrendo una rappresentazione più ricca e approfondita dei processi aziendali.

Questo lavoro propone un nuovo framework che utilizza algoritmi di scoperta causale per migliorare il process mining dichiarativo. Il nostro approccio estrae vincoli causali dai log degli eventi e li incorpora nei modelli DECLARE , colmando il divario tra il process mining basato sulla correlazione e la modellazione informata causalmente. In particolare, noi:

- Introduciamo una metodologia per integrare la scoperta causale nella modellazione dichiarativa dei processi.
- Sviluppiamo tecniche per l'apprendimento di vincoli dai log degli eventi che supportino relazioni di causalità.

Per illustrare la rilevanza pratica del nostro approccio, forniamo un caso di studio sulla generazione di modelli DECLARE attraverso la scoperta di relazioni causali tra dati e attività. Il nostro framework si basa su un'analisi strutturata delle tracce di processo e sulla verifica delle relazioni causali tramite algoritmi come PC e FCI, migliorando l'affidabilità dei modelli estratti.

La restante parte della tesi è strutturata come segue: Sezione 2 fornisce una panoramica sul framework DECLARE , sugli algoritmi di learning presenti in letteratura e sull'inferenza causale. Sezione 3 descrive il metodo per la generazione di modelli DECLARE attraverso la scoperta di relazioni causali tra dati. Sezione 4 approfondisce l'estensione di tale approccio alla scoperta di relazioni causali tra dati e attività. Sezione 5 discute le conclusioni e le direzioni future della ricerca.

# Capitolo 2

# Background

## 2.1 I *Workflow Management Systems* e il *Business Process Model and Notation*

I Workflow Management Systems (WFMS) rappresentano una componente essenziale nella moderna gestione dei processi aziendali, un'evoluzione tecnologica che ha cambiato radicalmente il modo in cui le organizzazioni gestiscono le proprie attività interne. La loro introduzione, avvenuta tra gli anni '80 e '90, è stata catalizzata dalla necessità di automatizzare operazioni complesse e di coordinare attività all'interno di aziende sempre più globalizzate e strutturate. In un contesto di crescente pressione per migliorare efficienza e ridurre gli errori operativi, i WFMS hanno assunto un ruolo di primo piano grazie alla loro capacità di ottimizzare i flussi di lavoro, migliorare la comunicazione interna e assicurare la conformità alle normative. Questi sistemi software non si limitano alla gestione dei processi aziendali, ma forniscono strumenti per modellare, eseguire e monitorare ogni fase operativa, creando un ecosistema integrato in cui tecnologia e processi lavorano in sinergia.

Alla base del funzionamento di un WFMS vi è la modellazione dei processi aziendali. Attraverso linguaggi standardizzati come BPMN (Business Process Model and Notation), le aziende possono rappresentare graficamente le proprie attività, delineando la sequenza delle operazioni, i vincoli temporali e le interazioni tra le diverse unità organizzative. Questo approccio strutturato permette alle imprese di analizzare e ottimizzare i propri processi prima ancora di implementare il sistema. Una volta configurato, un WFMS si occupa di orchestrare le attività, garantendo che vengano svolte nella sequenza corretta e rispettando regole predefinite. Ciò non solo riduce significativamente la possibilità di errori umani, ma assicura anche una maggiore visibilità sull'avanzamento del lavoro, con reportistica e monitoraggio in tempo reale.

L'adozione di un WFMS porta con sé vantaggi significativi, che vanno oltre la semplice automazione. Uno dei benefici più evidenti è la standardizzazione dei processi ripetitivi, un elemento cruciale in settori regolamentati come quello sanitario, finanziario o farmaceutico, dove la conformità normativa è obbligatoria. Ad esempio, le banche e le assicurazioni possono utilizzare questi sistemi per garantire che ogni richiesta o operazione rispetti le normative internazionali, riducendo il rischio di sanzioni o controversie legali. Parallelamente, la traccia-

bilità fornita da un WFMS consente di documentare ogni fase operativa, un aspetto particolarmente utile in audit o verifiche esterne. Le aziende possono così dimostrare facilmente il rispetto delle procedure, rafforzando la fiducia dei clienti e degli stakeholder.

Un altro aspetto cruciale è rappresentato dall'efficienza operativa. Automatizzando i processi ripetitivi, i WFMS liberano risorse che possono essere dedicate a compiti più strategici o creativi, favorendo così l'innovazione all'interno dell'organizzazione. Inoltre, i sistemi di workflow migliorano la collaborazione tra i dipendenti, abbattendo i silos organizzativi[1] che spesso ostacolano la comunicazione tra reparti. che spesso ostacolano la comunicazione tra reparti. Attraverso piattaforme centralizzate, le informazioni diventano accessibili a tutti gli attori coinvolti, favorendo decisioni più rapide e coordinate. Questo è particolarmente utile in contesti complessi come la gestione della supply chain, dove una mancata comunicazione può causare ritardi significativi o problemi di qualità.

Nonostante i numerosi vantaggi, i WFMS tradizionali presentano alcune limitazioni. Essi sono spesso progettati seguendo un approccio "imperativo", che specifica dettagliatamente ogni passaggio del processo. Questo li rende estremamente efficaci per attività ben strutturate, ma meno adatti a contesti dinamici o imprevedibili. Ad esempio, in settori come quello tecnologico o creativo, dove le esigenze possono evolversi rapidamente, la rigidità di un WFMS può rappresentare un ostacolo piuttosto che un vantaggio. Inoltre, l'implementazione di questi sistemi richiede spesso un investimento significativo in termini di tempo, risorse economiche e competenze tecniche. Le aziende devono non solo adattare i propri processi alle specifiche del software, ma anche formare il personale affinché ne comprenda appieno il funzionamento e ne massimizzi i benefici. Si riporta in figura 2.1, a titolo di esempio, la modellazione di un processo di gestione di referti medici.

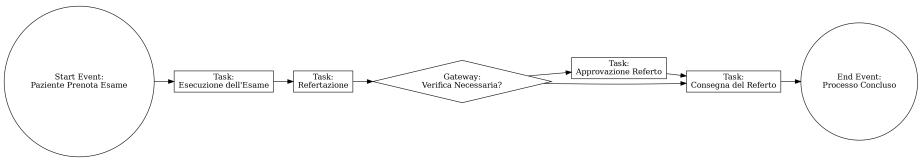



Figure 2.1: BPMN Model Example

## 2.2 DECLARE

In risposta alle limitazioni dei tradizionamli WFMS, è stato sviluppato DECLARE , un framework basato su un approccio dichiarativo per la modellazione e gestione dei processi aziendali. DECLARE rappresenta un'innovazione significativa nel panorama dei WFMS, in quanto consente di modellare processi attraverso l'uso di vincoli (constraints) che definiscono le regole da rispettare,

[1]Per silos organizzativi si intendono le divisioni o barriere tra i diversi reparti aziendali, che portano a un isolamento delle informazioni e a una scarsa cooperazione.

| | | |
|---|---|---|
| 1. | Start Event | **Paziente Prenota Esame**. Il paziente prenota un esame diagnostico online o tramite segreteria. Il sistema registra la prenotazione e invia una notifica al reparto diagnostico. |
| 2. | Task | **Esecuzione dell'Esame**. Il tecnico di laboratorio o radiologo esegue l'esame diagnostico. Output: immagini diagnostiche o dati associati. |
| 3. | Task | **Refertazione**. Il medico specialista accede al sistema e visualizza i dati diagnostici. Scrive il referto utilizzando strumenti digitali. |
| 4. | Gateway | **Verifica Necessaria?**. Sì: Il referto passa a un secondo medico per la verifica. No: Il referto è pronto per la consegna. |
| 5. | Task | **Approvazione Referto**. Il secondo medico approva o richiede modifiche. |
| 6. | Task | **Consegna del Referto**. Il referto approvato viene inviato al paziente tramite portale online o stampato per il ritiro. |
| 7. | End Event | **Processo Concluso** |

Table 2.1: Modellazione di un processo di gestione di referti medici

senza imporre una sequenza rigida di attività. Questa metodologia è particolarmente adatta per i loosely-structured processes (processi poco strutturati), caratterizzati da un'elevata imprevedibilità e dalla necessità di un adattamento continuo. [1]

Principi Fondamentali e Architettura di DECLARE sono stati sviluppati da M. Pesic e W. Van der Aalst e si fondano sull'utilizzo della logica temporale lineare (LTL) come formalismo per definire vincoli sui processi. La sua architettura è composta da tre componenti principali:
- Designer: un modulo per la creazione e modifica dei modelli di processo, che consente agli utenti di definire vincoli attraverso template visivi e intuitivi, evitando la necessità di una conoscenza approfondita della logica formale.
- Framework: il cuore del sistema, che gestisce l'esecuzione dei processi, monitorando il rispetto dei vincoli definiti e garantendo che le regole formali siano rispettate durante l'intera esecuzione.
- Worklist: un'interfaccia per gli utenti finali, che fornisce raccomandazioni operative in tempo reale e consente di monitorare lo stato dei vincoli durante l'esecuzione dei processi.
Questa struttura consente a DECLARE di coprire l'intero ciclo di vita di un processo: dalla modellazione iniziale, passando per l'esecuzione dinamica, fino all'analisi retrospettiva delle performance.

La distinzione tra approccio dichiarativo e imperativo è cruciale per comprendere l'innovazione introdotta da DECLARE . Nei modelli imperativi, l'esecuzione di un processo è rigidamente definita specificando una sequenza precisa di attività. In contesti dinamici, questa rigidità rappresenta un ostacolo, poiché non lascia spazio ad adattamenti o variazioni non previste.

L'approccio dichiarativo adottato da DECLARE , al contrario, si basa su un insieme di vincoli che specificano quali condizioni devono essere rispettate

durante l'esecuzione, lasciando agli utenti ampia libertà nel decidere come procedere. i vincoli a disposizione sono:

**Existence:** indica che l'attività A è eseguita almeno una volta in uno dei casi che compongono il log (vedi Figura 2.2a).

**Absence:** indica che l'attività A non viene mai eseguita all'interno del log di eventi (vedi Figura 2.2b).

**Responded Existence:** indica che se un'attività A è eseguita (almeno una volta) all'interno di una traccia, allora anche l'attività B deve essere eseguita all'interno della stessa traccia (vedi Figura 2.2c).

**Co-existence:** se un'attività A è eseguita almeno una volta in una determinata traccia, allora anche B deve essere eseguita prima o dopo A (vedi Figura 2.2d).

**Response:** se una attività A viene eseguita, B deve essere eseguita dopo di essa (vedi Figura 2.2e).

**Precedence:** L'attività B deve essere preceduta da A e non può verificarsi finché A non è stata eseguita (vedi Figura 2.2f).

**Succession:** Dopo A deve essere eseguita almeno una volta B. B deve inoltre essere preceduta da A e non può essere eseguita finché non è stata eseguita A (vedi Figura 2.2g).

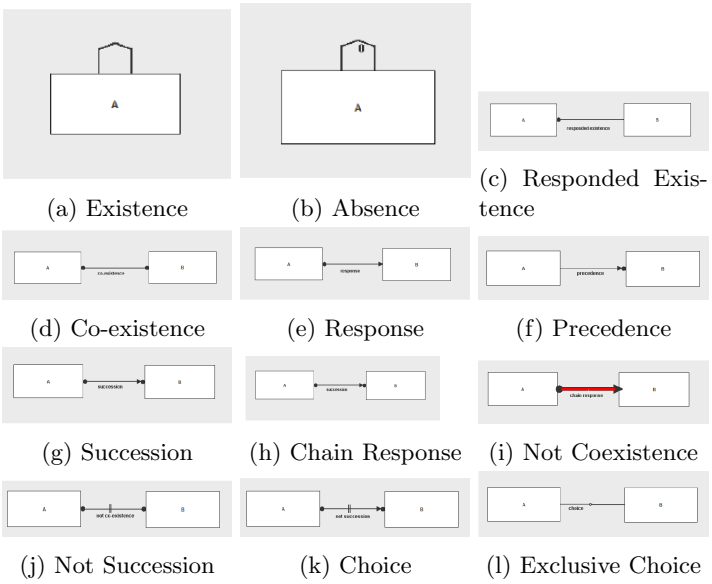
**Chain Response:** Tale template indica che dopo A viene immediatamente eseguita B (vedi Figura 2.2h).

**Not Coexistence:** Se A viene eseguita, B non può essere eseguita e viceversa (vedi Figura 2.2i).

**Not Succession:** Prima di B non può essere eseguita A e dopo A non può essere eseguita B (vedi Figura 2.2j).

**Choice:** Almeno una attività tra A e B deve essere eseguita (vedi Figura 2.2k).

**Exclusive Choice:** Una delle due attività a scelta tra A e B può essere eseguita, ma non entrambe (vedi Figura 2.2l).



(a) Existence (b) Absence (c) Responded Existence

(d) Co-existence (e) Response (f) Precedence

(g) Succession (h) Chain Response (i) Not Coexistence

(j) Not Succession (k) Choice (l) Exclusive Choice

Figure 2.2: Vincoli di esecuzione rappresentati graficamente

Questi vincoli non determinano una sequenza rigida, ma stabiliscono regole generali che il sistema monitora durante l'esecuzione, offrendo vantaggi come:

- **Flessibilità Operativa:** gli utenti possono adattare i processi in tempo reale, rispondendo prontamente a cambiamenti nelle esigenze o condizioni operative.
- **Riduzione della Complessità:** i modelli dichiarativi risultano meno complessi rispetto a quelli imperativi, poiché evitano di specificare dettagli inutili o ridondanti.
- **Analisi Avanzata:** grazie all'integrazione con strumenti come ProM, DECLARE consente di analizzare esecuzioni passate e di fornire raccomandazioni basate sui dati storici.
- **Gestione di Processi Misti:** DECLARE può essere integrato con altri framework, come YAWL, per gestire processi che combinano componenti strutturate e non strutturate.

### 2.2.1 DECLARE Limiti

I modelli dichiarativi basati su DECLARE sono strumenti potenti per descrivere processi attraverso vincoli logici che specificano regole di comportamento. Tuttavia, presentano alcune limitazioni che ne riducono l'efficacia in contesti reali. Uno dei principali limiti è la rigidità intrinseca dei vincoli, che devono essere soddisfatti pienamente in ogni tracciato. Questo approccio non considera le inevitabili eccezioni o deviazioni che caratterizzano molti processi aziendali, rendendo difficile modellare comportamenti occasionali o accettabili con basse frequenze. Inoltre, DECLARE ignora completamente l'incertezza e la variabilità

dei processi, che sono invece aspetti cruciali nei contesti pratici. In situazioni in cui i dati sono incompleti o rumorosi, o quando i comportamenti non seguono rigidi schemi predeterminati, questa mancanza di tolleranza alle incertezze porta a modelli poco rappresentativi e scarsamente adattabili.

Incorporare la valutazione delle incertezze e aggiungere una dimensione probabilistica ai modelli risponde a queste sfide. L'incertezza è una caratteristica fondamentale nei processi reali, dove il comportamento umano, le condizioni esterne e altre variabili introducono variazioni rispetto alle regole definite. Considerare l'incertezza consente di rappresentare in modo più realistico comportamenti rari ma accettabili, evitando che i modelli siano inutilmente restrittivi. Inoltre, l'aggiunta di probabilità ai vincoli permette di specificare non solo quali regole devono essere rispettate, ma anche con quale frequenza. Questo rende possibile descrivere comportamenti comuni senza imporli come obbligatori, catturando meglio la natura del processo osservato.

Il passaggio a un approccio probabilistico offre numerosi vantaggi. Innanzitutto, consente una modellazione più flessibile e ricca, andando oltre la tradizionale visione binaria (conforme/non conforme) per misurare invece il grado di conformità rispetto ai vincoli probabilistici. Inoltre, permette di assegnare pesi e priorità ai diversi vincoli, distinguendo tra regole più stringenti e quelle che possono essere violate più frequentemente. Questo approccio supporta decisioni operative più informate, poiché i modelli probabilistici diagnosticano non solo se un vincolo è stato violato, ma anche in che misura e con quale frequenza. Complessivamente, l'integrazione della probabilità nei modelli dichiarativi supera i limiti di DECLARE tradizionale, rendendo i modelli più realistici, adattabili e utili per analizzare e gestire processi complessi in scenari incerti.

### 2.2.2 Probabilità e DECLARE

In questa sezione, verranno analizzate alcune delle proposte presenti in letteratura relative all'introduzione della probabilità nei vincoli DECLARE . Tale analisi si concentrerà sull'esame delle metodologie, dei modelli e delle tecniche sviluppate per integrare una dimensione probabilistica nei vincoli dichiarativi, con l'obiettivo di aumentare la flessibilità e la capacità di rappresentare l'incertezza nei processi. In particolare, verranno approfonditi approcci basati su modelli probabilistici e strumenti di calcolo, evidenziando vantaggi, limiti e applicazioni specifiche.

#### ProbDECLARE

Nel lavoro seminale [2] gli autori propongono un'estensione probabilistica per i modelli dichiarativi, introducendo la nozione di vincoli probabilistici. Essi descrivono un modello chiamato ProbDECLARE , che integra:

- **Scoperta di vincoli probabilistici:** Utilizzando tecniche esistenti, adattate per gestire vincoli con una natura probabilistica.
- **Monitoraggio probabilistico:** La possibilità di monitorare tracciati parziali con stati multipli associati a diverse probabilità.

- **Verifica di conformità probabilistica:** Introducendo la nozione di *Earth Mover's Distance* per valutare la conformità di un registro rispetto a un modello probabilistico.

Accenniamo ad un minimo di nomenclatura sulle basi formali e le definizioni necessarie per comprendere il modello probabilistico proposto. Tra i concetti principali:

- **Multinsiemi:**
  - Un multinsieme $S$ su un insieme $A$ è una funzione $S : A \to \mathbb{N}$, dove $S(a)$ rappresenta il numero di volte in cui l'elemento $a$ appare in $S$.
  - **Notazione:** $a^n \in S$ se $S(a) = n$, e $|S|$ rappresenta la cardinalità totale del multinsieme:

$$|S| = \sum_{a \in S} S(a).$$

- **Alfabeto e Tracciati:**
  - $\Sigma$: alfabeto finito delle attività (azioni elementari di un processo).
  - Un tracciato $\tau$ è una sequenza finita di attività:

$$\tau = a_1, a_2, \ldots, a_n \quad \text{con } a_i \in \Sigma.$$

  - La lunghezza del tracciato è denotata da $\text{length}(\tau) = n$.
  - Il concetto di tracciato è fondamentale perché rappresenta un'esecuzione del processo, descrivendo l'ordine in cui si verificano le attività.
- **Registro degli Eventi:**
  - Un registro è un multinsieme di tracciati.
  - Formalmente, un registro $L$ su $\Sigma$ è un multinsieme finito su $\Sigma^*$, l'insieme di tutte le sequenze finite su $\Sigma$.
  - La cardinalità del registro è il numero totale di tracciati.
- **LTL su Tracciati Finiti (LTLf):**
  - La logica temporale lineare su tracciati finiti (**LTLf**) è utilizzata per modellare proprietà di sequenze finite.
  - Le formule LTLf sono costruite con la seguente grammatica:

$$\varphi ::= a \mid \neg\varphi \mid \varphi_1 \vee \varphi_2 \mid \bigcirc\varphi \mid \varphi_1 \mathcal{U} \varphi_2$$

  - **Elementi della sintassi:**
    - $a$: rappresenta un'attività appartenente all'insieme $\Sigma$ delle attività del processo. Ogni attività $a$ corrisponde a un evento registrato nel log.
    - $\neg$ e $\vee$: sono gli operatori booleani classici di negazione ($\neg$, "non") e disgiunzione ($\vee$, "oppure").

    - ∗ ◯ **(Next)**: operatore temporale che significa "nel prossimo istante". Indica che la formula successiva deve essere vera nel prossimo stato della traccia.
    - ∗ $U$ **(Until)**: operatore temporale che significa "fino a che". La formula $\varphi_1 \; U \; \varphi_2$ è vera se $\varphi_2$ diventa vera a un certo punto, e fino ad allora $\varphi_1$ deve essere vera.

- **Esempio:**
  - La formula:
    $$\Box(\text{close} \rightarrow \bigcirc\Diamond\text{acc})$$
    - ∗ □ **(Globally):** L'operatore □ indica che la formula contenuta deve essere vera in **tutti gli stati futuri**. Nel nostro caso, □(close → ◯◊acc) significa che la condizione close → ◯◊acc deve essere soddisfatta in ogni istante della traccia.
    - ∗ → **(Implica):** Il simbolo → rappresenta un'implicazione logica. La formula close → ◯◊acc significa che **se si verifica l'evento "close" in uno stato, allora la formula ◯◊acc deve essere vera nel prossimo stato**.
    - ∗ ◯ **(Next):** L'operatore ◯ significa **"nel prossimo stato"**. La formula ◯◊acc significa che **in uno degli stati successivi deve accadere acc**.
    - ∗ ◊ **(Eventually):** L'operatore ◊ significa **"eventualmente"** o **"prima o poi"**. La formula ◊acc indica che **l'attività acc deve verificarsi in uno degli stati futuri**.

    La formula puo' quindi essere interpretata come segue:

    > *"In ogni stato, se si verifica l'evento **close**, allora in uno degli stati successivi si deve verificare l'evento **acc** prima o poi."*

- **Automi:**
  - Ogni formula LTLf può essere tradotta in un automa a stati finiti che accetta esattamente i tracciati che soddisfano la formula.
  - Questo collegamento con gli automi è cruciale per l'implementazione degli algoritmi di verifica e monitoraggio.

- **Modelli Dichiarativi con DECLARE :**
  - DECLARE è un linguaggio di modellazione basato su vincoli temporali, utilizzando LTLf per specificare i vincoli.
  - Un modello DECLARE è una coppia $\langle \Sigma, C \rangle$, dove:
    - ∗ $\Sigma$: un insieme finito di attività.
    - ∗ $C$: un insieme di vincoli (formule LTLf).
  - Un tracciato soddisfa un modello DECLARE se soddisfa tutti i vincoli in $C$.

- **Vincolo probabilistico**:
  - Un triplo $\langle \varphi, \bowtie, p \rangle$, dove:

* $\varphi$: Una formula LTL che rappresenta il vincolo processuale.
* $\bowtie$: Un operatore probabilistico $(=, \neq, \leq, \geq, <, >)$.
* $p$: Una soglia probabilistica (es. $p = 0.8$).

- **Esempio**:
$$\langle \Box(\text{close} \rightarrow \bigcirc\Diamond\text{acc}), 0.8 \rangle$$
Significa che almeno l'80% dei tracciati deve soddisfare la condizione che ogni chiusura di ordine sia seguita da un'accettazione.

- **Scenari di vincoli**:
  - Gli scenari sono combinazioni di soddisfazione/violazione dei vincoli probabilistici.
  - Per un modello con $n$ vincoli, esistono $2^n$ scenari possibili.

- **Semantica probabilistica**:
  - Un vincolo probabilistico $\langle \varphi, \bowtie, p \rangle$ è soddisfatto dal linguaggio stocastico $\rho$ se la somma delle probabilità delle tracce $\tau \in \Sigma^*$ che soddisfano la proprietà $\varphi$, ovvero:
$$\sum_{\tau \in \Sigma^*,\, \tau \models \varphi} \rho(\tau),$$
rispetta la condizione $\bowtie\ p$, dove $\bowtie$ è un operatore relazionale (ad esempio $\geq, \leq, =$).
  - **Esempio:** Se $p = 0.8$ e $\bowtie = \geq$, allora almeno l'80% della probabilità totale del linguaggio deve essere associata a tracce che soddisfano $\varphi$.

Passiamo ora ad analizzare le tre fasi del metodo ProbDECLARE .

La scoperta di vincoli probabilistici rappresenta il primo passo e si articola in tre fasi principali. Nella prima fase, vengono generati vincoli candidati utilizzando tecniche consolidate per la scoperta di vincoli deterministici nei modelli DECLARE . Questi vincoli sono inizialmente privi di componenti probabilistiche.

Successivamente, si calcolano le probabilità empiriche associate a ciascun vincolo analizzando il registro di eventi. Per un vincolo $\phi$, la probabilità empirica è definita come il rapporto tra il numero di tracciati che soddisfano $\phi$ e il numero totale di tracciati nel registro:
$$p_\phi = \frac{|\{\tau \in L \mid \tau \models \phi\}|}{|L|}$$
dove $L$ è il registro degli eventi, $\tau$ rappresenta un tracciato e $\tau \models \phi$ indica che il tracciato $\tau$ soddisfa il vincolo $\phi$.

Infine, nella terza fase, si selezionano i vincoli rilevanti applicando filtri basati su soglie di probabilità. Si eliminano i vincoli ridondanti, inconsistenti o con probabilità troppo alte o basse per risultare significativi. Un ulteriore filtro verifica la consistenza logica dei vincoli, assicurandosi che non generino scenari con probabilità cumulative maggiori di 1.

L'algoritmo complessivo integra queste fasi per identificare un insieme di vincoli probabilistici significativi che descrivono accuratamente i comportamenti osservati nei dati.

Il monitoraggio probabilistico è il secondo passo del metodo ProbDECLARE e consente di verificare, durante l'esecuzione di un processo, il rispetto dei vincoli probabilistici definiti, anche in presenza di tracciati parziali. A differenza dell'analisi retrospettiva su tracciati completi, il monitoraggio runtime si basa su una sequenza incompleta di attività, detta tracciato parziale, e mira a prevedere la probabilità che il tracciato completo soddisfi un determinato vincolo probabilistico. Per realizzare questo obiettivo, si utilizzano automi probabilistici, una generalizzazione degli automi finiti che integra informazioni probabilistiche. Un automa probabilistico è definito come una struttura

$$A = (Q, \Sigma, \delta, q_0, F, P)$$

dove $Q$ è l'insieme degli stati, $\Sigma$ rappresenta l'alfabeto delle attività, $\delta : Q \times \Sigma \to Q$ è la funzione di transizione, $q_0$ è lo stato iniziale, $F \subseteq Q$ è l'insieme degli stati finali accettanti e $P$ rappresenta le probabilità associate alle transizioni tra stati. L'algoritmo di monitoraggio si articola in diverse fasi. In primo luogo, per ciascun vincolo probabilistico $\langle \phi, \bowtie, p \rangle$, si costruisce un automa probabilistico che accetta i tracciati che soddisfano la formula $\phi$. Durante l'esecuzione del processo, ogni nuovo evento osservato aggiorna lo stato corrente dell'automa e modifica le probabilità di appartenenza del tracciato parziale agli scenari definiti dal modello probabilistico.

Ad esempio, consideriamo un modello ProbDECLARE con tre scenari principali:

- **S011**: Include i vincoli existence(close) e la versione negata di response(close, acc). Questo implica che l'evento *acc* non deve seguire *close*.
- **S101**: Contiene il vincolo response(close, acc) e not coexistence(acc, ref) (cioè, *acc* e $ref$ non possono coesistere).
- **S110**: Richiede che i vincoli response(close, acc) e response(close, ref) siano rispettati, ma che not coexistence(acc, ref) sia violato.

Le probabilità associate agli scenari sono: $S011$, irrilevante; $S101$, lo scenario più probabile con 70%; e $S110$, lo scenario meno probabile con 10%.

All'inizio della traccia, non è stato osservato alcun evento, quindi tutti gli scenari sono potenzialmente validi. Dopo l'esecuzione di *close*, la situazione rimane invariata, poiché il prossimo evento può ancora soddisfare i vincoli di uno o più scenari. L'osservazione di *acc* modifica lo stato del monitoraggio: $S011$ viene permanentemente violato, poiché il vincolo not response(close, acc) richiede che *acc* non segua *close*. Tuttavia, $S101$ diventa potenzialmente valido, perché il vincolo response(close, acc) è rispettato, mentre not coexistence(acc, ref) rimane incerto. Se la traccia si fermasse qui, essa apparterrebbe a $S101$, il più probabile (70%).

Quando viene osservato l'evento $ref$, lo scenario cambia ancora. $S101$ viene permanentemente violato poiché il vincolo not coexistence(acc, ref) è violato (entrambi *acc* e $ref$ sono presenti). D'altro canto, $S110$ diventa potenzialmente soddisfatto, poiché i vincoli response(close, acc) e response(close, ref) sono rispettati e la violazione di not coexistence(acc, ref) è coerente con lo scenario. Alla fine, la traccia appartiene a $S110$, lo scenario meno probabile (10%), e non è non conforme, poiché soddisfa i vincoli di almeno uno scenario.

Questo esempio dimostra come il monitoraggio probabilistico consenta di adattarsi dinamicamente ai nuovi eventi, aggiornando lo stato e identificando lo scenario più coerente con l'evoluzione della traccia. Sebbene inizialmente sembrasse appartenere a $S101$, la traccia è stata infine classificata in $S110$, evidenziando la flessibilità del sistema nell'adattarsi a cambiamenti inattesi. Tale approccio è cruciale per garantire la coerenza tra i tracciati osservati e i modelli probabilistici definiti, anche in presenza di tracciati parziali o scenari meno probabili.

L'ultimo passo è quello della verifica di conformità probabilistica permette di valutare in che misura un registro di eventi aderisce a un modello probabilistico definito, estendendo i metodi tradizionali per includere una dimensione probabilistica. Questo approccio ha due obiettivi principali: misurare la conformità, determinando quanto il registro rispetti le probabilità richieste dai vincoli probabilistici, e quantificare le deviazioni, fornendo una misura della "distanza" tra il comportamento osservato e quello atteso. La verifica si basa sul calcolo delle probabilità osservate per ciascun vincolo $\langle \phi, \bowtie, p \rangle$, utilizzando la proporzione di tracciati che soddisfano il vincolo $\phi$ nel registro. Successivamente, queste probabilità osservate vengono confrontate con le soglie probabilistiche tramite una misura di distanza. Una delle metriche proposte è l'Earth Mover's Distance (EMD), che misura lo "sforzo" necessario per riallineare le distribuzioni probabilistiche osservate a quelle attese. La somma delle deviazioni permette di ottenere una misura aggregata della non conformità del registro al modello probabilistico. Questo approccio offre flessibilità e precisione, permettendo di individuare violazioni anche minime e di quantificare in modo continuo la distanza dal modello. Grazie alla robustezza dell'EMD e alla capacità di gestire scenari multipli, la verifica di conformità probabilistica si dimostra uno strumento essenziale per garantire la coerenza tra i processi osservati e i modelli probabilistici definiti, rendendola particolarmente utile in applicazioni aziendali e operative.

Il metodo ProbDECLARE rappresenta un significativo avanzamento rispetto ai modelli dichiarativi tradizionali, introducendo la capacità di gestire incertezze e variazioni nel comportamento dei processi. Tuttavia, presenta alcuni limiti che potrebbero ridurne l'applicabilità o la scalabilità in determinati contesti, il principale limite è quello della dipendenza dai dati del registro, il metodo ProbDECLARE si basa infatti sulla probabilità empirica calcolata a partire dai dati presenti nel registro degli eventi. Se il registro è incompleto, rumoroso, o non rappresentativo del processo reale, i risultati potrebbero essere inaccurati o fuorvianti. Nei prossimi modelli che andremo ad analizzare vedremo soluzioni che cercheranno di risolvere questa problematicità.

### Probabilistic Traces in Declarative Process Mining

Nelle prossime tre sezioni analizzeremo alcune proposte riguardanti l'integrazione della probabilità all'interno del framework Declare. Queste proposte mirano a superare il problema, discusso nella sezione precedente, legato alla dipendenza dai file di log. Come ricordato, il metodo analizzato in precedenza risultava inefficace qualora il log fosse incompleto o contenesse rumore o errori, producendo risultati potenzialmente fuorvianti.

In [3], gli autori propongono una semantica innovativa per gestire l'incertezza a livello degli eventi in una traccia, nell'ambito del process mining dichiarativo.

L'idea chiave è che alcune informazioni in una traccia di processo possano essere inaffidabili, e che un esperto di dominio possa indicare un grado di credibilità associato a tali informazioni. Questo grado di credibilità è espresso attraverso un valore di probabilità epistemica, che può essere attribuito all'esecuzione di un evento.

Per chiarire meglio il concetto, gli autori forniscono un esempio basato su un protocollo medico. Supponiamo che, secondo le linee guida, la somministrazione di antibiotici pre-operatori sia cruciale per ridurre il rischio di infezioni. Tuttavia, potrebbero verificarsi casi in cui questa somministrazione non venga correttamente documentata a causa di errori umani o problemi tecnici. Ad esempio, un chirurgo potrebbe ricordare di aver ordinato antibiotici per un paziente prima di un intervento chirurgico, ma non trovare traccia di ciò nei registri medici. Basandosi sulla pratica abituale, sulle interazioni con il personale infermieristico e sulle raccomandazioni del protocollo, il chirurgo potrebbe stimare con una probabilità del 95% che gli antibiotici siano stati effettivamente somministrati. Questa stima non si basa su dati quantitativi, ma sulla fiducia del chirurgo nei propri protocolli standard.

Dal punto di vista formale, gli autori introducono i concetti di *Evento Probabilistico* e *Traccia Probabilistica* partendo dalla *Probabilistic Logic Programming* (PLP) e dalla semantica di distribuzione. La probabilità di un evento viene trattata come la probabilità che l'evento si manifesti (o meno) in un mondo. Secondo la semantica proposta, tutti i mondi che includono un determinato evento "ereditano" la sua probabilità, mentre i mondi che non lo includono considerano il complemento a 1 della sua probabilità. Gli autori affrontano anche il problema del *probabilistic conformance checking*, definendo il punteggio di conformità come la somma delle probabilità dei mondi in cui la traccia risulta conforme.

**Definizioni**

Per comprendere meglio la struttura della proposta, riportiamo le definizioni fornite dagli autori:

1. **Evento Probabilistico** Un *Evento Probabilistico* è una coppia:

   $$Prob : EventDescription$$

   dove $EventDescription$ è un simbolo che descrive un evento ($EventDescription \in \Sigma$), mentre $Prob \in [0, 1]$ rappresenta la probabilità epistemica che l'evento sia avvenuto. Un valore di probabilità pari a 1 indica che l'evento è certo e, per semplicità, tale probabilità viene omessa.

2. **Scelta Atomica** Una *Scelta Atomica* è una coppia $(E_i, k)$, dove $E_i$ è un evento probabilistico $E$ che appare nella $i$-esima posizione in una traccia probabilistica e $k \in \{0, 1\}$. Il valore $k$ indica se $E_i$ è incluso in un mondo con probabilità $p_i$ ($k = 1$) oppure no, con probabilità $1 - p_i$ ($k = 0$). Si assume l'indipendenza tra le diverse scelte atomiche.

3. **Scelta Composita** Una *Scelta Composita* $\kappa(t)$ è un insieme consistente di scelte atomiche relative agli eventi probabilistici in $t$, tale che $(E_i, k) \in \kappa(t)$ e $(E_i, m) \in \kappa(t)$ implicano $k = m$ (una sola decisione per ciascun evento probabilistico). La probabilità $P(\kappa)$ di una scelta composita $\kappa$ è:

   $$P(\kappa) = \prod_{(E_i,1)\in\kappa} p_i \prod_{(E_i,0)\in\kappa} (1 - p_i),$$

dove $p_i$ è la probabilità associata a $E_i$.

4. **Selezione** Una *Selezione* $\sigma(t)$ su una traccia probabilistica $t$ è una scelta composita contenente una scelta atomica $(E_i, k)$ per ciascun evento probabilistico in $t$.

5. **Probabilità di una Selezione** La probabilità di una selezione $\sigma(t)$ è definita come:

$$P(\sigma(t)) = \prod_{(E_i,1)\in\sigma(t)} p_i \prod_{(E_i,0)\in\sigma(t)} (1 - p_i).$$

   La probabilità di una selezione corrisponde alla probabilità di un mondo $w_\sigma$, ovvero $P(w_\sigma(t)) = P(\sigma(t))$. La probabilità di un mondo si ottiene moltiplicando le probabilità associate a ciascuna alternativa (presenza o assenza di un evento), poiché queste sono considerate indipendenti. Questo genera una distribuzione di probabilità $P(w(t))$ sui mondi, tale che:

$$\sum_{w(t)\in W(t)} P(w(t)) = 1.$$

6. **Conformità di una Traccia Probabilistica** Data un modello $M$, una traccia probabilistica $t$, e un insieme $S(t)$ di tutte le selezioni $\sigma_i(t)$, definiamo la conformità $Comp(M, t)$ di una traccia probabilistica $t$ rispetto al modello $M$ come la somma delle probabilità dei mondi $w_{\sigma_i(t)}$ per cui la funzione di conformità restituisce valore vero:

$$Comp(M, t) = \sum_{w(t)\in W(t)} \begin{cases} P(w(t)) & \text{se } w(t) \text{ è conforme a } M, \\ 0 & \text{altrimenti.} \end{cases}$$

**Probabilistic Compliance in Declarative Process Mining**

In [4] gli autori,studiano un diverso approccio, sempre con l'idea di risolvere il problema della dipendenza dal log vista in [1]. Introducono il concetto di **Probabilistic Declarative Process Specification (PDS)**, un'estensione delle Declarative Process Specifications basata sulla semantica di distribuzione derivata dalla *Probabilistic Logic Programming*. Questo approccio permette di modellare processi dichiarativi che includono sia vincoli certi (*crisp constraints*) sia vincoli probabilistici. La probabilità associata a ciascun vincolo riflette la sua rilevanza o forza all'interno del dominio di processo considerato. Un vincolo con probabilità pari a 1 è considerato "certo" e corrisponde ai vincoli definiti nei modelli dichiarativi tradizionali.

Un insieme di vincoli probabilistici e certi forma una **Probabilistic Declarative Process Specification (PDS)**, che si distingue da una tradizionale Declarative Specification proprio per l'integrazione della probabilità. Gli autori evidenziano che, nel caso in cui tutti i vincoli abbiano probabilità pari a 1, il PDS coincide con una Declarative Specification standard.

Per interpretare il significato probabilistico dei vincoli, viene definito il concetto di **selezione**, che consiste in un insieme di scelte atomiche. Ogni scelta atomica specifica se un determinato vincolo probabilistico è presente o meno in una configurazione specifica, chiamata "mondo". La probabilità di una selezione

è determinata assumendo l'indipendenza tra i vincoli, ed è calcolata moltiplicando le probabilità associate ai vincoli inclusi e ai complementi delle probabilità per i vincoli esclusi. Un mondo rappresenta, quindi, una configurazione specifica di vincoli selezionati e può essere considerato un modello dichiarativo classico.

Gli autori definiscono la probabilità di un mondo $w_\sigma$ come segue:

$$P(w_\sigma) = \prod_{(c_i,1)\in\sigma} p_i \prod_{(c_i,0)\in\sigma} (1 - p_i),$$

dove $p_i$ rappresenta la probabilità associata al vincolo $c_i$. Questo approccio genera una distribuzione di probabilità sui diversi modelli dichiarativi derivati dal PDS.

Un contributo significativo del lavoro è la definizione della **conformità probabilistica** di una traccia rispetto a un PDS. Questa misura rappresenta la probabilità che una traccia $t$ sia conforme ai vincoli di almeno uno dei modelli dichiarativi derivati dal PDS. La conformità probabilistica è formalizzata come segue:

$$\text{comp}(t, PDS) = \sum_{\sigma_i : t \text{ conforme a } DS_i} P(DS_i),$$

dove $DS_i$ è un modello dichiarativo associato a una selezione $\sigma_i$.

### Efficient compliance computation in Probabilistic Declarative Specifications

In [5] gli autori propongono un'estensione delle specifiche di processo dichiarative introducendo elementi probabilistici per gestire l'incertezza nel process mining. Questo approccio si basa sul concetto di Specifiche Dichiarative Probabilistiche (PDS), che combinano vincoli certi e probabilistici. I vincoli probabilistici sono associati a una probabilità che rappresenta la loro probabilità di inclusione nel modello. La presenza o l'assenza di ciascun vincolo è considerata indipendente, il che porta alla creazione di molteplici mondi possibili, ciascuno rappresentante un sottoinsieme unico di vincoli. La probabilità di un mondo è calcolata come il prodotto delle probabilità dei vincoli inclusi e dei complementi delle probabilità dei vincoli esclusi:

$$P(w) = \prod_{c\in w} p(c) \cdot \prod_{c\notin w} (1 - p(c)),$$

dove $p(c)$ è la probabilità associata al vincolo $c$. Per una determinata traccia $t$, la probabilità che essa sia conforme al modello probabilistico Declare è la somma delle probabilità di tutti i mondi in cui i vincoli soddisfano la traccia:

$$\text{comp}(t, PDS) = \sum_{w\in W : t \models w} P(w).$$

Tuttavia, a causa dell'esplosione combinatoria dei mondi possibili, l'enumerazione esplicita non è praticabile. Gli autori superano questa sfida applicando il principio di inclusione-esclusione per calcolare direttamente la probabilità di conformità senza dover considerare esplicitamente ogni mondo.

Il calcolo della conformità avviene in due fasi. Nella prima fase, i vincoli violati ($VIO$) sono identificati tramite uno strumento esterno. Questi rappresentano i vincoli non soddisfatti dalla traccia $t$. Successivamente, nella seconda

fase, viene applicato il principio di inclusione-esclusione per determinare la probabilità di conformità. Il metodo considera il minor numero tra i vincoli violati ($VIO$) e quelli soddisfatti ($SAT$), definiti come:

$$SAT = \{c \in \mathcal{M} \setminus VIO : p(c) < 1\},$$

dove $\mathcal{M}$ è l'insieme dei vincoli del PDS. Se $VIO$ contiene vincoli certi ($p(c) = 1$), la conformità è immediatamente zero, poiché la traccia non può essere conforme a un modello che include vincoli certi e violati. Altrimenti, la probabilità di conformità viene calcolata come:

$$\text{comp}(t, PDS) = 1 - \text{InclusionExclusion}(VIO, p),$$

quando $|VIO| \leq |SAT|$, oppure:

$$\text{comp}(t, PDS) = P_{no\_vio} \cdot \big(\text{InclusionExclusion}(SAT, p) + P_{no\_sat}\big),$$

quando $|VIO| > |SAT|$, dove:

$$P_{no\_vio} = \prod_{c \in VIO} (1 - p(c)), \quad P_{no\_sat} = \prod_{c \in SAT} (1 - p(c)).$$

Il principio di inclusione-esclusione, applicato a un insieme di vincoli $S$, calcola la probabilità dell'unione dei vincoli iterando su tutti i sottoinsiemi $T \subseteq S$. La probabilità è espressa come:

$$P(S) = \sum_{T \subseteq S} (-1)^{|T|+1} \prod_{c \in T} p(c).$$

Questo metodo consente di considerare le sovrapposizioni tra vincoli senza enumerare esplicitamente tutti i mondi.

**Conclusioni**

SCRIVERE IL CONFRONTO TRA I TRE APPROCCI

## 2.3 Algoritmi di Learning Declare

In questa sezione ci proponiamo di analizzare lo stato dell'arte relativo agli algoritmi di apprendimento per la generazione automatica di modelli *Declare*, un approccio sempre più rilevante nell'ambito del *Process Mining*. Partiremo esaminando uno dei tool più consolidati e ampiamente citati in letteratura, ovvero *decMiner*, soffermandoci sulle sue principali caratteristiche e limiti. Successivamente, prenderemo in considerazione una serie di algoritmi alternativi che sono stati proposti negli ultimi anni, analizzandone le prestazioni in termini di accuratezza, capacità di generalizzazione e applicabilità a diversi contesti reali.

L'obiettivo di questa rassegna è quello di fornire una panoramica critica dei principali strumenti disponibili, evidenziando i progressi compiuti fino ad oggi.

### 2.3.1 Declare Miner

Il **Declare Miner** [6] è uno strumento avanzato per il process mining dichiarativo, progettato per scoprire modelli di processo flessibili a partire dai log di eventi. Integrato come plug-in nel toolkit **ProM**, il Declare Miner consente di rappresentare processi aziendali attraverso vincoli logici, piuttosto che sequenze rigide di attività, utilizzando il linguaggio **Declare** basato sulla *Linear Temporal Logic* (LTL) per tracce finite.

Questo strumento è nato per rispondere alla crescente esigenza di analizzare processi aziendali caratterizzati da elevata variabilità e complessità. In questi contesti, i modelli procedurali tradizionali risultano spesso inadatti, poiché definiscono in modo rigido tutte le sequenze possibili di attività. Al contrario, i modelli dichiarativi permettono di rappresentare i processi in modo più flessibile, utilizzando regole che specificano vincoli sul comportamento consentito. Questo approccio garantisce una maggiore adattabilità in ambienti dinamici, come il settore sanitario o i processi di gestione delle emergenze.

Il metodo implementato nel Declare Miner si articola in due fasi principali: la fase Apriori e la fase di Analisi delle Sequenze. Queste fasi lavorano in sinergia per identificare vincoli dichiarativi basati sui template del linguaggio Declare, garantendo un processo di scoperta efficiente anche su log di grandi dimensioni.

#### Fase 1: Algoritmo Apriori

La prima fase mira a individuare insiemi frequenti di attività all'interno del log, utilizzando l'algoritmo Apriori. Questo è fondamentale per ridurre lo spazio di ricerca dei vincoli candidati. L'algoritmo si basa sul concetto di supporto, definito come la proporzione di tracce in cui un determinato insieme di attività appare. La formula del supporto è:

$$supp(A) = \frac{|L_A|}{|L|} \tag{2.1}$$

dove $L_A$ rappresenta il sottoinsieme di tracce in cui tutte le attività di $A$ sono presenti, e $L$ è l'insieme totale delle tracce nel log. Un insieme di attività è considerato frequente se il suo supporto supera una soglia minima $supp_{min}$. L'algoritmo opera iterativamente:

- **Passo iniziale**: Identifica gli insiemi frequenti di singole attività.
- **Passo iterativo**: Genera insiemi di attività di dimensioni maggiori combinando insiemi già trovati frequenti.

Ad esempio, se un log contiene 100 tracce e un insieme $\{A, B\}$ compare in 80 di esse, il supporto di questo insieme è calcolato come:

$$supp(\{A, B\}) = \frac{80}{100} = 0.8 \tag{2.2}$$

L'algoritmo Apriori sfrutta la proprietà di **chiusura discendente**, secondo cui un insieme è frequente solo se tutti i suoi sottoinsiemi sono frequenti.

### Fase 2: Analisi delle Sequenze

Nella seconda fase, chiamata Analisi delle Sequenze, vengono verificati i vincoli candidati generati dalla fase Apriori, utilizzando algoritmi specifici per ciascun template di Declare. La verifica avviene confrontando le tracce del log con automi associati ai vincoli candidati.

I principali template di vincoli analizzati sono:

- **Response (A, B)**: Se $A$ si verifica, allora $B$ deve verificarsi successivamente.
- **ChainResponse (A, B)**: Se $A$ si verifica, $B$ deve verificarsi immediatamente dopo.
- **Existence (n, A)**: L'attività $A$ deve verificarsi almeno $n$ volte.

Per migliorare l'efficienza del Declare Miner, vengono introdotte due strategie di parallelizzazione: **Search Space Partitioning** e **Database Partitioning**.

**Search Space Partitioning** suddivide l'analisi in base ai *template di vincoli*. In questo approccio, l'insieme dei vincoli candidati viene suddiviso in gruppi distinti, ognuno dei quali corrisponde a un template specifico (ad esempio, Response, Precedence o ChainResponse). Ogni gruppo viene elaborato da un thread separato, consentendo di parallelizzare l'elaborazione dei vincoli e ridurre significativamente il tempo necessario per analizzare grandi log di eventi.

Questo approccio si rivela particolarmente utile quando il numero di vincoli candidati è elevato, poiché ogni thread lavora in modo indipendente senza necessità di sincronizzazione. Inoltre, viene implementata una **gerarchia dei vincoli** per ottimizzare ulteriormente il processo: se un vincolo più forte (ad esempio, *ChainResponse(A, B)*) è soddisfatto, i vincoli più deboli derivati dallo stesso insieme (come *Response(A, B)*) possono essere automaticamente considerati soddisfatti, evitando ulteriori verifiche.

Al contrario, **Database Partitioning** suddivide l'analisi in base alle *tracce del log*. In questo approccio, il log di eventi viene suddiviso in blocchi di tracce, ciascuno dei quali viene analizzato indipendentemente da un thread separato. Ogni thread verifica i vincoli candidati sulle tracce assegnate, garantendo un'elaborazione parallela. Durante l'analisi, i thread filtrano i vincoli vacui, ovvero quelli che non vengono mai attivati in nessuna traccia del blocco, e i vincoli soddisfatti da vincoli più forti.

Queste strategie di parallelizzazione permettono di migliorare significativamente le prestazioni del Declare Miner, consentendo di analizzare log complessi in tempi ragionevoli. La combinazione delle due tecniche può essere utilizzata in alcuni scenari, dove il log viene prima suddiviso in blocchi (Database Partitioning) e ogni blocco viene successivamente analizzato per template (Search Space Partitioning). L'approccio proposto si dimostra efficace nel produrre modelli dichiarativi scalabili e interpretabili anche in contesti industriali.

## 2.3.2 Algoritmi di Model Learning per la Scoperta di DFAs

La scoperta automatica di *Automata a Stati Finiti Deterministici* (Deterministic Finite State Automata, DFAs) a partire dai registri di eventi è un elemento

chiave nel campo della *process mining*, poiché consente di modellare i comportamenti osservati nei processi aziendali, verificare la conformità dei processi stessi e monitorare l'esecuzione delle attività per individuare eventuali anomalie. Tuttavia, il processo di scoperta automatica di questi modelli richiede l'impiego di algoritmi avanzati di *Model Learning* (ML), che si distinguono principalmente in due categorie: algoritmi attivi e algoritmi passivi. Entrambi gli approcci hanno lo scopo di costruire un DFA che rappresenti fedelmente i comportamenti osservati nei registri di eventi, ma differiscono per metodologia e applicabilità pratica.

**RPNI (Regular Positive and Negative Inference)**
L'algoritmo RPNI [7] è uno dei metodi più utilizzati per la scoperta di DFAs. Parte da un *prefix tree acceptor* (PTA), una struttura ad albero che rappresenta tutte le tracce positive presenti nel registro di eventi. RPNI procede poi unendo gli stati equivalenti del DFA, generando un modello che accetta tutte le tracce positive e rifiuta quelle negative.

L'obiettivo di RPNI è trovare un DFA minimo che sia consistente con le tracce positive e compatibile con le tracce negative. Il processo di unione degli stati avviene seguendo un ordine di visita sugli stati del PTA, evitando di accettare tracce indesiderate.

- Costruzione iniziale del *prefix tree acceptor* (PTA).
- Per ogni coppia di stati $q_i$ e $q_j$, verifica se unire gli stati porta a un DFA consistente.
- Se la fusione è consentita, unisce gli stati; altrimenti, li mantiene separati.

Il principale limite di RPNI è la sua tendenza a sovradimensionare i modelli in presenza di rumore nei dati.

**EDSM (Evidence Driven State Merging)**
L'algoritmo EDSM [7] rappresenta un miglioramento di RPNI, introducendo una funzione di valutazione basata sulle evidenze per guidare la fusione degli stati. Questa funzione calcola la probabilità che due stati siano equivalenti in base alle transizioni osservate. La funzione di evidenza è definita come:

$$E(q_i, q_j) = \sum_{t \in T} P(t \mid q_i) \cdot P(t \mid q_j)$$

dove $P(t \mid q)$ rappresenta la probabilità che una transizione $t$ sia osservata partendo dallo stato $q$. L'obiettivo di EDSM è ridurre la complessità computazionale del processo di scoperta, producendo modelli più semplici e accurati rispetto a RPNI.

**MDL (Minimum Description Length)**
L'algoritmo MDL [7] applica un criterio basato sulla teoria della compressione per minimizzare la lunghezza complessiva del modello e dei dati codificati tramite il modello stesso. L'idea principale è che un DFA più compatto, che rappresenta efficacemente i dati, sia preferibile.

La funzione obiettivo di MDL è:

$$L(M) + L(D \mid M)$$

dove:

- $L(M)$ è la lunghezza del modello in termini di numero di stati e transizioni.
- $L(D \mid M)$ è la lunghezza dei dati compressi tramite il modello.

MDL opera selezionando le fusioni di stati che riducono maggiormente la lunghezza complessiva della descrizione del sistema. Questo approccio garantisce che il DFA risultante sia compatto e interpretabile. L'algoritmo è particolarmente utile in contesti dove sono disponibili solo tracce positive e dove è necessario generare modelli che rappresentino una generalizzazione ragionevole dei dati.

**$L^*$ e Variante TTT**

L'algoritmo $L^*$, proposto da Dana Angluin nel 1987, è un metodo di apprendimento attivo basato su query. Il learner interagisce con un insegnante (*teacher*), ponendo due tipi principali di query:

- *Membership Query (MQ)*: verifica se una sequenza appartiene al linguaggio del processo.
- *Equivalence Query (EQ)*: verifica se il DFA generato è equivalente al comportamento del sistema. Se il modello è incompleto, il teacher fornisce un controesempio.

La variante TTT (*Two-Threshold Tree*) migliora l'efficienza dell'algoritmo $L^*$ utilizzando alberi di discriminazione per ridurre il numero di query necessarie.

**Metriche di Valutazione**

Per valutare le performance dei modelli generati dagli algoritmi di Model Learning, gli autori del paper hanno utilizzato quattro metriche principali:

- **Fitness**: misura quanto il modello generato accetta le tracce presenti nel registro di eventi. La fitness perfetta (1.0) si ottiene quando tutte le tracce del log sono accettate dal DFA. La formula per calcolare la fitness è:

$$Fitness(M, L) = \frac{|Tracce\_accettate(M, L)|}{|Tracce\_totali(L)|}$$

  dove $M$ è il modello generato, $L$ è il registro di eventi e $Tracce\_accettate(M, L)$ rappresenta il numero di tracce accettate dal modello.

- **Precisione**: valuta quanto il DFA accetta solo comportamenti osservati nei dati di addestramento, evitando di accettare tracce non presenti nel registro. La precisione viene calcolata come:

$$Precision(M, L) = \frac{|Subtracce(M) \cap Subtracce(L)|}{|Subtracce(M)|}$$

  dove $Subtracce(M)$ rappresenta l'insieme delle sottosequenze generate dal modello e $Subtracce(L)$ rappresenta quelle osservate nel registro.

- **Generalizzazione**: verifica la capacità del modello di accettare nuove tracce valide non presenti nei dati di addestramento, attraverso una validazione incrociata. La formula utilizzata è:

$$Generalizzazione = 1 - \frac{|Errori\_sul\_test|}{|Tracce\_di\_test|}$$

dove $Errori\_sul\_test$ è il numero di tracce valide che il modello non riesce ad accettare.

- **Semplicità**: misura la complessità del DFA in termini di numero di stati e transizioni, secondo la formula:

$$Semplicità(M) = \frac{1}{|Stati(M)| + |Transizioni(M)|}$$

dove $Stati(M)$ rappresenta il numero di stati e $Transizioni(M)$ il numero di transizioni nel DFA.

Gli autori hanno condotto esperimenti su dataset reali e sintetici per valutare gli algoritmi di Model Learning.

I risultati mostrano che gli **algoritmi passivi** hanno ottenuto fitness perfetta e generato modelli più compatti e interpretabili rispetto a Declare Miner. Inoltre, hanno dimostrato maggiore precisione e generalizzazione, accettando solo sequenze valide senza introdurre comportamenti indesiderati. **MDL**, **RPNI** ed **EDSM** si sono rivelati i più efficaci, mentre **L*** ha mostrato difficoltà con dataset complessi.

Declare Miner ha prodotto modelli più complessi e meno interpretabili, con precisione inferiore rispetto agli algoritmi passivi. Inoltre, ha evidenziato problemi di scalabilità, superando in alcuni casi i limiti di tempo previsti.

In conclusione, gli algoritmi passivi risultano più adatti alla scoperta di modelli DFA precisi e compatti nei registri di eventi complessi, mentre Declare Miner offre una rappresentazione più flessibile ma con costi computazionali maggiori.

### 2.3.3 Process Discovery on Deviant Traces and Other Stranger Things

#### Formalizzazione del problema

L'obiettivo di NegDis è costruire un modello dichiarativo ottimale in grado di:

- Accettare tutte le tracce positive, ossia quelle che rappresentano esecuzioni corrette del processo.
- Rigettare tutte le tracce negative, che corrispondono a esecuzioni deviate o indesiderate.

Il modello è espresso mediante vincoli dichiarativi nel linguaggio Declare, un formalismo basato su logica temporale lineare su tracce finite (LTLf). Il problema può essere formalizzato come la ricerca di un insieme di vincoli $M$ tale che:

$$\forall t \in L^+, \quad t \models M$$

$$\forall t \in L^-, \quad t \not\models M$$

Dove $M$ è un insieme di vincoli $c$ appartenenti all'insieme dei vincoli dichiarativi possibili, definiti come:

$$D[A] = \{c \mid c \text{ è un'istanza di un template in } D \text{ su attività in } A\}$$

Un modello $M$ è quindi una congiunzione di vincoli che devono essere soddisfatti da tutte le tracce accettate:

$$M = \bigwedge_{c \in D[A]} c$$

La qualità del modello viene valutata secondo i principi di:

- **Generalità**: la capacità di astrarre comportamenti non osservati nel log.
- **Precisione**: il grado di restrizione del modello, evitando sia overfitting che underfitting.
- **Semplicità**: la riduzione del numero di vincoli per migliorare la leggibilità e interpretabilità del modello.

Per costruire il modello ottimale, l'algoritmo segue due fasi principali: identificazione dei vincoli candidati e ottimizzazione.

**Fase 1: Identificazione dei vincoli candidati**

Nella prima fase, vengono individuati i vincoli che accettano tutte le tracce positive e al contempo escludono almeno una traccia negativa. Si parte calcolando l'insieme di vincoli compatibili:

$$\text{compatibles}(D[A], L^+) = \{c \in D[A] \mid \forall t \in L^+, t \models c\}$$

Successivamente, si determina la funzione *sheriffs*, che associa a ciascuna traccia negativa i vincoli che la escludono:

$$\text{sheriffs}(t) = \{c \in \text{compatibles} \mid t \not\models c\}, \quad \forall t \in L^-$$

Un modello candidato $S$ è un insieme di vincoli che soddisfa:

- Accetta tutte le tracce positive.
- Esclude tutte le tracce negative attraverso almeno un vincolo.

L'insieme di tutte le soluzioni possibili è quindi:

$$Z = \{S \subseteq D[A] \mid \forall t \in L^-, \exists c \in S \text{ tale che } t \not\models c\}$$

**Fase 2: Ottimizzazione del modello**

Una volta individuati i modelli candidati, viene selezionato quello ottimale secondo i seguenti criteri:

- **Ottimizzazione per generalità**: Si cerca il modello che ammette il maggior numero di tracce non osservate, evitando eccessive restrizioni:

$$\forall S' \in Z, \quad \text{se } cl(S' \cup P) \supset cl(S \cup P) \text{ allora } S \text{ non è ottimale} \tag{2.3}$$

  dove $cl(S)$ è l'operatore di chiusura deduttiva, che considera la gerarchia tra vincoli Declare.

- **Ottimizzazione per semplicità**: Si sceglie il modello con il minor numero di vincoli, mantenendo la capacità di distinguere tra tracce positive e negative:

$$\forall S' \in Z, \quad |cl(S' \cup P)| < |cl(S \cup P)| \Rightarrow S \text{ non è ottimale} \tag{2.4}$$

L'ottimizzazione è implementata tramite *Answer Set Programming* (ASP), sfruttando *CLINGO* per selezionare il modello migliore con vincoli deboli.

**Efficienza computazionale e implementazione**

Un aspetto chiave di *NegDis* è la sua efficienza computazionale, ottenuta attraverso:

- Verifica lineare delle constraints grazie all'uso di *regexp* in Go.
- Selezione ottimale dei vincoli con ASP e CLINGO.
- Scalabilità superiore rispetto ai metodi basati su *pattern mining* o *sequence learning*.

L'algoritmo esegue il controllo di soddisfacibilità utilizzando espressioni regolari (*regexp*), garantendo una verifica in tempo lineare rispetto alla dimensione del log.

La fase di ottimizzazione ASP è $DP_2$-completa, ma grazie alla riduzione del numero di vincoli e all'uso di tecniche di *pruning*, il metodo risulta scalabile anche per dataset di grandi dimensioni.

## 2.4 Inferenza Causale

### 2.4.1 Fondamenti di Causalità

#### Introduzione alla Causalità

La distinzione tra correlazione e causalità è fondamentale nella statistica, nelle scienze sociali e nell'intelligenza artificiale (AI). La correlazione tra due variabili indica che esiste una relazione statistica tra di esse: quando una variabile cambia, anche l'altra tende a cambiare. Tuttavia, questa relazione non implica necessariamente un legame causale.

Per esempio, supponiamo di osservare che il consumo di gelato e il numero di annegamenti aumentano entrambi durante l'estate. Sebbene esista una correlazione tra questi eventi, non possiamo concludere che mangiare gelato causi annegamenti. In realtà, entrambe le variabili sono influenzate da una terza variabile: la temperatura. Questo fenomeno è noto come *confondimento*.

Formalmente, possiamo definire la correlazione tra due variabili $X$ e $Y$ utilizzando il coefficiente di correlazione di Pearson:

$$\rho_{X,Y} = \frac{\text{Cov}(X, Y)}{\sigma_X \sigma_Y}$$

dove:

- $\text{Cov}(X, Y)$ è la covarianza tra $X$ e $Y$,
- $\sigma_X$ e $\sigma_Y$ sono le deviazioni standard di $X$ e $Y$.

Un coefficiente di correlazione vicino a 1 o -1 indica una forte relazione lineare tra le variabili, ma **non ci dice nulla sul fatto che una variabile causi l'altra**.

Al contrario, la causalità implica che un cambiamento nella variabile $X$ provoca direttamente un cambiamento nella variabile $Y$. Per distinguere tra correlazione e causalità, dobbiamo considerare esperimenti controllati o utilizzare modelli causali formali, come quelli basati su **grafi aciclici diretti (DAGs)**.

### 2.4.2 Modelli Statistici vs. Modelli Causali

#### Modelli Meccanistici, Strutturali e Statistici

Esistono tre principali tipi di modelli utilizzati per descrivere i fenomeni naturali:

- **Modelli Meccanistici**: Descrivono i processi fisici che governano un sistema. Ad esempio, le leggi di Newton possono essere utilizzate per prevedere il movimento di un oggetto.
- **Modelli Strutturali (SCMs)**: Definiscono le relazioni causali tra variabili utilizzando funzioni e variabili di rumore indipendenti. Ogni variabile $X_i$ è espressa come una funzione dei suoi genitori causali:

  $$X_i = f_i(\text{PA}_i, U_i)$$

  dove:

  - $\text{PA}_i$ rappresenta l'insieme dei genitori di $X_i$ nel grafo causale,

– $U_i$ è un termine di rumore indipendente.

- **Modelli Statistici**: Si concentrano sull'identificazione di associazioni nei dati, senza considerare le relazioni causali sottostanti. Ad esempio, un modello di regressione può prevedere il prezzo di una casa in base alla sua dimensione, ma non può determinare se la dimensione della casa causa un cambiamento nel prezzo.

### Il Principio della Causa Comune di Reichenbach

Un concetto fondamentale nella causalità è il *Principio della Causa Comune* di Hans Reichenbach. Questo principio afferma che ogni correlazione tra due variabili $X$ e $Y$ può essere spiegata da un set di variabili $Z$, detta *causa comune*, che spiega tale correlazione.

Formalmente, il principio può essere espresso come:

$$P(X, Y \mid Z) = P(X \mid Z) \cdot P(Y \mid Z)$$

Ad esempio, consideriamo le variabili:

- $X$: numero di gelati venduti,
- $Y$: numero di annegamenti,
- $Z$: temperatura.

La correlazione tra $X$ e $Y$ può essere spiegata dal fatto che entrambi sono influenzati dalla temperatura $Z$.

### Limitazioni dei Modelli Statistici nell'Inferenza Causale

I modelli statistici tradizionali soffrono di diverse limitazioni quando si tratta di inferire relazioni causali:

- **Confondimento**: I modelli statistici non possono distinguere tra correlazioni dirette e correlate attraverso una variabile confondente.
- **Bias di Selezione**: La correlazione osservata nei dati potrebbe essere distorta da un bias nella selezione del campione.
- **Assenza di Interventi**: I modelli statistici si basano solo su dati osservazionali e non possono prevedere l'effetto degli interventi.

Qeusti modelli tradizionali apprendono correlazioni tra variabili nei dati osservazionali, ma queste correlazioni potrebbero non essere valide in scenari nuovi, dove le distribuzioni dei dati cambiano. Al contrario, i modelli causali sono piu' robusti a questi cambiamenti, perchè catturano le relazioni causali sottostanti, che rimangono stabili anche quando la dis- tribuzione dei dati varia.

**Esempio:** Consideriamo un modello che predice il prezzo di una casa in base alla di- mensione del giardino e alla vicinanza a una scuola. Se la relazione tra queste variabili cambia (ad esempio, a causa di nuove normative urbanistiche), un mod- ello statistico tradizionale potrebbe fallire. Tuttavia, un modello causale che cattura le relazioni tra le variabili (ad esempio, che la vicinanza a una scuola influisce sulla domanda di abitazioni) sarebbe in grado

di adattarsi meglio a questi cambiamenti. Per superare queste limitazioni, è necessario adottare un approccio basato sui modelli causali che includa la rappresentazione esplicita delle relazioni causali e delle assunzioni necessarie per inferire la causalità.

### 2.4.3 Rappresentazione della Causalità

#### Grafi Causali (DAGs)

Un *grafo aciclico diretto (DAG)* è una rappresentazione grafica delle relazioni causali tra variabili in un sistema. È costituito da:

- **Nodi**: che rappresentano le variabili del sistema,
- **Archi diretti**: che indicano relazioni causali dirette tra le variabili.

Un DAG è definito "aciclico" perché non contiene cicli, ovvero sequenze di archi che riconducono a un nodo già visitato. Questa proprietà garantisce che le relazioni causali siano **unidirezionali**.

Esempio di DAG:

$$X_1 \to X_2 \to X_3$$

In questo grafo:

- $X_1$ è la causa di $X_2$,
- $X_2$ è la causa di $X_3$,
- Non esiste un percorso che riporti a $X_1$, garantendo l'assenza di cicli.

#### d-Separation e Indipendenza Condizionale

La *d-separation* (directional separation) è una proprietà dei DAG che permette di determinare se due variabili sono **indipendenti** dato un insieme di altre variabili.

Due variabili $X$ e $Y$ sono *d-separate* in un DAG rispetto a un insieme di variabili $Z$ se tutti i percorsi tra $X$ e $Y$ sono bloccati da $Z$. Un percorso può essere bloccato in tre modi:

1. Se il percorso contiene una catena $X \to Z \to Y$ e $Z \in \mathcal{Z}$,
2. Se il percorso contiene un biforcuto $X \leftarrow Z \to Y$ e $Z \in \mathcal{Z}$,
3. Se il percorso contiene un collisore $X \to Z \leftarrow Y$ e $Z \notin \mathcal{Z}$ e nessuno dei discendenti di $Z$ è in $\mathcal{Z}$.

Questa proprietà è fondamentale per determinare le **relazioni di indipendenza condizionale** che possono essere derivate da un DAG.

Esempio:

Consideriamo il seguente DAG:

$$X_1 \to X_2 \to X_3$$

Le seguenti relazioni di indipendenza possono essere derivate utilizzando la d-separation:

$$X_1 \perp X_3 \mid X_2$$

dove $\perp$ indica l'indipendenza condizionale.

#### Differenza tra Modelli Grafici Probabilistici e Causali

I modelli grafici probabilistici, come le reti bayesiane, rappresentano le relazioni di dipendenza tra variabili, ma non distinguono tra correlazione e causalità. Al contrario, i **modelli grafici causali** utilizzano i DAG per rappresentare le **relazioni causali** tra variabili.

Questa differenza è cruciale quando si tratta di prevedere l'effetto di interventi. Nei modelli grafici probabilistici, un arco tra due variabili indica una dipendenza statistica, mentre nei modelli causali indica che una variabile è **causa diretta** dell'altra.

### 2.4.4 Modelli Causali Strutturali (SCMs)

#### Definizione e Formalizzazione degli SCM

I *modelli causali strutturali (SCMs)* sono una rappresentazione formale delle relazioni causali tra variabili. Ogni variabile in un SCM è determinata da:

- Una **funzione deterministica** che dipende dalle sue cause dirette (genitori),
- Un **termine di rumore indipendente** che cattura l'incertezza e le influenze non misurate.

Formalmente, un SCM è definito come un insieme di equazioni strutturali:

$$X_i = f_i(\mathrm{PA}_i, U_i)$$

dove:

- $X_i$ è la variabile dipendente,
- $\mathrm{PA}_i$ rappresenta l'insieme dei genitori di $X_i$ nel DAG,
- $U_i$ è un termine di rumore indipendente,
- $f_i$ è una funzione deterministica.

Esempio:
Consideriamo un semplice sistema causale:

$$X_1 \to X_2 \to X_3$$

Possiamo rappresentare questo sistema utilizzando un SCM:

$$\begin{cases} X_1 = U_1 \\ X_2 = f_2(X_1, U_2) \\ X_3 = f_3(X_2, U_3) \end{cases}$$

dove $U_1, U_2, U_3$ sono termini di rumore indipendenti.

### 2.4.5 Inferenza Causale: Assunzioni Fondamentali

Per inferire correttamente le relazioni causali dai dati osservazionali, è necessario fare alcune assunzioni fondamentali che permettono di distinguere la mera correlazione dalla causalità. In particolare, ci si basa su tre assunzioni principali: la **Causal Markov Condition**, la **Faithfulness** e la **Causal Sufficiency**. Queste assunzioni forniscono il fondamento teorico per l'analisi causale basata su grafi diretti aciclici (DAG, Directed Acyclic Graphs) e altri strumenti statistici.

#### Causal Markov Condition

La **Causal Markov Condition** (Condizione di Markov Causale) afferma che ogni variabile in un grafo causale è indipendente da tutte le altre variabili non discendenti, condizionatamente ai suoi genitori diretti.

#### Definizione formale

Sia dato un modello causale rappresentato da un grafo aciclico diretto (DAG), in cui le variabili casuali $X_1, X_2, ..., X_n$ sono i nodi e le relazioni causali sono rappresentate dagli archi diretti. La Causal Markov Condition può essere espressa formalmente come:

$$X_i \perp \text{Desc}(X_i) \mid \text{PA}(X_i)$$

dove:

$X_i$ è una variabile nel modello,
$\text{PA}(X_i)$ indica l'insieme dei genitori diretti di $X_i$,
$\text{Desc}(X_i)$ indica l'insieme dei discendenti di $X_i$,
$\perp$ indica l'indipendenza condizionale.

Questa condizione implica che una variabile $X_i$ dipende solo dai suoi genitori diretti e non dalle altre variabili, a meno che non siano discendenti di $X_i$.

#### Esempio pratico

Consideriamo un sistema causale in cui **Fumo** ($X_1$) causa **Malattie Cardiache** ($X_2$) e **Cancro ai Polmoni** ($X_3$):

Secondo la Causal Markov Condition, condizionatamente al Fumo ($X_1$), la Malattia Cardiaca ($X_2$) e il Cancro ai Polmoni ($X_3$) sono indipendenti tra loro. Questo significa che, se conosciamo l'abitudine al fumo di una persona, conoscere se ha il cancro ai polmoni non ci dà informazioni aggiuntive sulla probabilità di avere una malattia cardiaca.

#### Assunzione di Faithfulness

L'assunzione di **Faithfulness** (Fedeltà) implica che tutte le dipendenze condizionali presenti nel grafo causale siano anche presenti nella distribuzione dei dati. Questo elimina la possibilità di coincidenze statistiche che potrebbero nascondere reali relazioni causali.

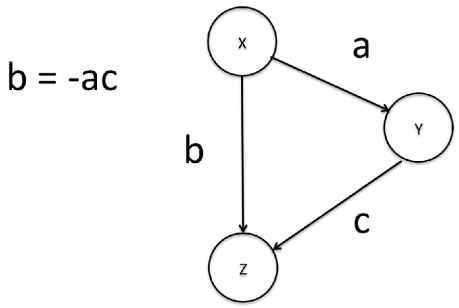



Figure 2.3: Esempio violazione Faithfulness tratto da Philosophy of Science.

### Definizione formale

Se una distribuzione di probabilità segue un DAG causale, allora ogni indipendenza statistica nella distribuzione dei dati deve riflettere un'assenza di connessione causale nel grafo. In altre parole, non esistono cancellazioni accidentali di correlazioni dovute a combinazioni particolari di parametri.

### Esempio pratico

Se nel nostro modello il Fumo ($X_1$) causa sia il Cancro ai Polmoni ($X_3$) che le Malattie Cardiache ($X_2$), non può accadere che la correlazione statistica tra $X_1$ e $X_3$ scompaia casualmente a causa di combinazioni particolari dei parametri del modello.

Tuttavia, una violazione dell'assunzione di *Faithfulness* si verifica quando effetti causali opposti si annullano esattamente.

Osservando l'immagine Fig. 2.3 supponiamo che i coefficienti $a$, $b$ e $c$ rappresentino le forze delle relazioni causali tra le variabili. Se la relazione tra $X$ e $Z$ ($b$) viene "cancellata" da una correlazione di segno opposto tra $X$ e $Y$ ($a$), che a sua volta influenza $Z$ ($c$), potremmo osservare un'apparente indipendenza tra $X$ e $Z$. Questo accade perché l'effetto diretto $b$ e il percorso indiretto $a \cdot c$ si bilanciano esattamente, risultando in:

$$b = -a \cdot c$$

In questa situazione, se osserviamo i dati senza conoscere la struttura causale, potremmo concludere erroneamente che $X$ non ha alcun effetto su $Z$, anche se esiste un percorso causale che le collega. Questo fenomeno è un esempio di come la cancellazione esatta degli effetti possa nascondere relazioni causali reali nel modello.

### Assunzione di Causal Sufficiency

L'assunzione di **Causal Sufficiency** richiede che tutte le cause comuni delle variabili osservate siano incluse nel modello. In pratica, questo implica che non ci siano **confondenti non osservati**, cioè variabili che influenzano simultaneamente più variabili osservate ma che non sono state incluse nel modello.

#### Definizione formale

Sia $X_1, ..., X_n$ un insieme di variabili osservate in un modello causale. L'assunzione di Causal Sufficiency afferma che non esiste una variabile non osservata $U$ che causi contemporaneamente due o più variabili in $X_1, ..., X_n$.

#### Esempio pratico

Consideriamo un caso in cui stiamo studiando la relazione tra **Consumo di Gelato** ($X_1$) e **Incidenti in Mare** ($X_2$). Se non consideriamo la **Temperatura** ($U$) come variabile nel nostro modello, potremmo erroneamente concludere che mangiare più gelato causi più incidenti in mare. In realtà, la temperatura più alta in estate aumenta sia il consumo di gelato che il numero di persone che nuotano, portando a più incidenti.

Se la Temperatura non è inclusa nel modello, violiamo l'assunzione di Causal Sufficiency, il che può portare a conclusioni errate.

### 2.4.6 Algoritmi di Causal Discoveri

L'analisi causale rappresenta un obiettivo fondamentale in molte discipline scientifiche e applicazioni pratiche, poiché consente di inferire le relazioni di causa-effetto tra variabili a partire da dati osservazionali. Diversi algoritmi sono stati sviluppati per affrontare questo problema, ciascuno con ipotesi specifiche e metodologie differenti. Tra i più noti si annoverano l'algoritmo PC, il Fast Causal Inference (FCI), il Greedy Equivalence Search (GES) e modelli basati sulla non-gaussianità e sulle relazioni non lineari. Ciascuno di essi presenta vantaggi e limitazioni, a seconda della struttura dei dati e della presenza di variabili latenti. Nei paragrafi seguenti, verranno analizzati in dettaglio questi algoritmi, descrivendone il funzionamento, le ipotesi di base e le applicazioni.

#### L'Algoritmo PC

L'algoritmo PC (Peter-Clark), introdotto da Spirtes et al. (2001), è uno dei più antichi metodi di scoperta causale. Esso garantisce coerenza statistica sotto il campionamento i.i.d., assumendo che non vi siano confondenti latenti. Il suo framework consente di integrare diverse procedure statistiche per determinare le relazioni di indipendenza condizionale tra variabili, come test d'ipotesi o criteri basati sulla differenza di fitting score, ad esempio il Bayesian Information Criterion (BIC) tra modelli con e senza un determinato arco diretto.

L'algoritmo si basa su due ipotesi fondamentali:

1. **Ipotesi di Markov causale**: ogni variabile è indipendente da tutte le altre non discendenti dato il proprio insieme di genitori.

2. **Principio di Faithfulness**: due variabili sono direttamente collegate (con un arco tra di esse) se e solo se non esiste un sottoinsieme delle restanti variabili che le renda indipendenti condizionatamente.

L'algoritmo PC segue una sequenza di passaggi iterativi per identificare la struttura causale sottostante:

1. **Costruzione del grafo iniziale**: Si parte da un grafo completamente connesso e non orientato tra tutte le variabili (vedi Fig. 2.4, parte B).

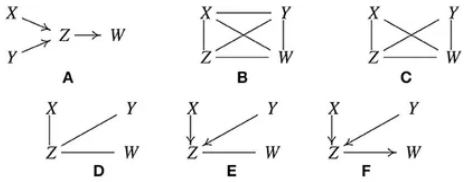



Figure 2.4: PC Algorithm Step tratto da Frontiers in Genetics.

2. **Rimozione degli archi per indipendenze marginali**: Gli archi tra variabili statisticamente indipendenti vengono eliminati (vedi Fig. 2.4, parte C).

3. **Test di indipendenza condizionata**: Per ogni coppia di variabili collegate da un arco $(A, B)$, si valuta l'indipendenza condizionata su ciascuna delle altre variabili connesse. Se si verifica $A \perp B|C$, l'arco tra $A$ e $B$ viene rimosso (vedi Fig. 2.4, parte D).

4. **Estensione del test ad insiemi più grandi**: Si ripete il processo aumentando la dimensione dell'insieme condizionante $\{C, D\}$, eliminando ulteriori archi se si verifica $A \perp B|\{C, D\}$.

5. **Individuazione delle strutture a "v" (v-structures)**: Se tre variabili $A, B, C$ formano una catena $A - B - C$, e $A$ e $C$ non sono collegate, allora si orienta $B$ come nodo centrale $A \rightarrow B \leftarrow C$ (vedi Fig. 2.4, parte E).

6. **Propagazione dell'orientamento**: Se in un triplo $A \rightarrow B - C$ l'arco tra $B$ e $C$ è ancora non orientato, e ci sono vincoli strutturali che ne impongono la direzione, l'orientamento viene propagato (vedi Fig. 2.4, parte F).

L'algoritmo continua fino a quando non è più possibile determinare nuove indipendenze condizionali. Se i test di indipendenza sono corretti e il campione è sufficientemente grande, il PC convergerà alla Classe di Equivalenza Markoviana (MEC) della struttura causale vera.

L'output dell'algoritmo PC è un CPDAG (Completed Partially Directed Acyclic Graph), un grafo che rappresenta un insieme di DAG equivalenti rispetto alle relazioni di indipendenza condizionale osservate. In alcuni casi, alcuni archi possono rimanere non orientati se il modello non fornisce evidenze sufficienti per determinare la direzione causale.

Un aspetto chiave è la differenza tra l'output dell'algoritmo PC e un semplice grafo di indipendenza condizionale (Lauritzen, 1996). Mentre il grafo di indipendenza condizionale rappresenta relazioni di indipendenza tra variabili considerando tutte le altre variabili nel sistema, l'output dell'algoritmo PC fornisce una struttura causale con archi direzionati, distinguendo tra correlazione e causalità.

Ad esempio, in un caso specifico, $X$ e $Y$ possono risultare indipendenti marginalmente, ma non condizionatamente su $Z$ e $W$, il che porta a una struttura differente rispetto a un grafo di sola indipendenza condizionale. Grazie alla sua efficienza computazionale, l'algoritmo PC è applicabile a reti causali sparse con decine di migliaia di variabili, specialmente nei contesti lineari o multinomiali, dove i test di indipendenza sono computazionalmente efficienti.

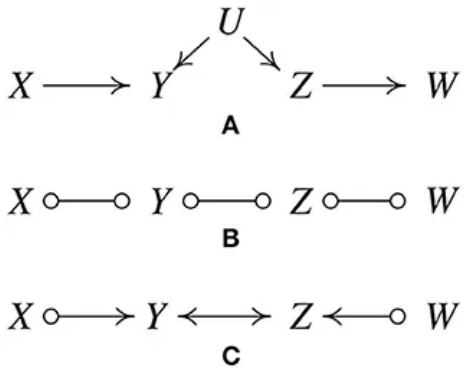



Figure 2.5: FCI Algorithm Step tratto da Frontiers in Genetics.

**L'Algoritmo FCI**

L'algoritmo Fast Causal Inference (FCI) è una generalizzazione dell'algoritmo PC progettata per gestire scenari in cui possono essere presenti variabili latenti. Introdotto da Spirtes et al. (2001), FCI consente di inferire la struttura causale anche in presenza di confondenti non misurati e può fornire informazioni sulla presenza di tali variabili nascoste.

Il primo passo dell'algoritmo FCI è simile a quello dell'algoritmo PC: vengono eliminati gli archi che non sono supportati dai test di indipendenza condizionale, ottenendo così un grafo parzialmente connesso. Tuttavia, a differenza dell'algoritmo PC, che presuppone che ogni connessione sia direzionata in base alle informazioni di indipendenza condizionale, l'algoritmo FCI introduce una maggiore flessibilità nell'orientamento degli archi.

**Passaggi dell'algoritmo FCI**

1. **Costruzione del grafo iniziale**: Si parte da un grafo completamente connesso, in cui ogni variabile è collegata a tutte le altre.

2. **Rimozione degli archi basata su indipendenze marginali**: Gli archi tra variabili che risultano indipendenti vengono eliminati.

3. **Test di indipendenza condizionata**: Come nel PC, si verifica se due variabili sono indipendenti condizionatamente su altre variabili e si eliminano gli archi non supportati.

4. **Propagazione delle informazioni causali**: A differenza dell'algoritmo PC, FCI non assume che tutte le relazioni siano direttamente osservabili. Per questo motivo, alcuni archi nel grafo risultante possono contenere simboli speciali, come "ooo", che indicano che la direzione causale non può essere determinata con certezza.

Un esempio pratico è illustrato nella Figura 2.5A, in cui è presente una variabile latente $U$, che funge da confondente tra due variabili osservabili $Y$ e $Z$. Il grafo iniziale mostra un collegamento tra $Y$ e $Z$, ma dopo l'applicazione dell'algoritmo FCI, il collegamento diventa bidirezionale (indicando la possibile presenza di una variabile latente).

Nel grafo risultante (Figura 2.5C), alcuni archi tra variabili contengono marcatori "ooo", che indicano incertezza nella direzione della causalità. Questo avviene perché l'algoritmo, basandosi esclusivamente sulle relazioni di indipendenza condizionale, non è in grado di distinguere tra un effetto diretto e un effetto mediato da un confondente non osservato.

Come nel caso dell'algoritmo PC, esistono diverse varianti di FCI sviluppate per migliorare l'efficienza computazionale. Tra queste, una delle più note è l'algoritmo RFCI (Colombo et al., 2012), che riduce il numero di test di indipendenza effettuati, accelerando così il processo di inferenza senza perdere informazioni chiave sulle relazioni causali.

### L'Algoritmo GES (Greedy Equivalence Search)

A differenza degli algoritmi PC e FCI, che partono da un grafo completamente connesso e successivamente eliminano le connessioni non supportate dai test di indipendenza, l'algoritmo Greedy Equivalence Search (GES) adotta un approccio opposto:

1. Parte da un grafo vuoto e aggiunge progressivamente gli archi necessari.
2. Rimuove eventuali connessioni superflue in una fase successiva.

Introdotto da Chickering (2003), GES è basato su una strategia greedy che utilizza criteri bayesiani per valutare iterativamente se l'aggiunta di un arco migliora l'aderenza del modello ai dati.

L'algoritmo si divide in:

1. **Fase di Forward Search**: Si parte da un grafo privo di connessioni e vengono aggiunti progressivamente gli archi tra le variabili, selezionando quelli che massimizzano un criterio di ottimizzazione basato su punteggi quasi-bayesiani, come il Bayesian Information Criterion (BIC).
2. **Fase di Backward Search**: Dopo aver costruito una struttura iniziale, vengono rimossi alcuni archi, verificando se la loro eliminazione migliora ulteriormente il punteggio del modello.

GES assegna ogni modello alla Markov Equivalence Class (MEC) corrispondente, garantendo che la struttura risultante sia coerente con le ipotesi di indipendenza condizionale.

Una caratteristica distintiva di GES rispetto a PC è che non si basa su test di indipendenza, ma sulla ricerca di un modello che massimizzi la probabilità dei dati osservati. Questo lo rende più robusto in presenza di errori di misura o campioni di dimensione ridotta, ma meno interpretabile rispetto agli algoritmi basati sull'indipendenza condizionale.

Per affrontare la limitazione di GES nell'identificare variabili latenti, è stato sviluppato l'algoritmo GFCI (Greedy Fast Causal Inference) (Ogarrio et al., 2016). Questo metodo combina GES con FCI, permettendo di:

- Utilizzare GES per identificare una superstruttura del grafo scheletrico.
- Applicare FCI per affinare il grafo, eliminando connessioni spurie e determinando le direzioni delle relazioni causali.

Studi sperimentali hanno dimostrato che GFCI è più accurato dell'algoritmo FCI originale in numerose simulazioni, rendendolo una scelta preferibile in contesti pratici.

### Modelli Basati sulla Non-Gaussianità

Un altro approccio alla scoperta causale si basa sulla **non-gaussianità** degli errori nei modelli di regressione lineare. Questi metodi sfruttano proprietà statistiche che emergono quando le variabili non seguono una distribuzione gaussiana, permettendo di determinare la direzione della causalità tra due variabili.

### Il Modello LiNGAM (Linear Non-Gaussian Acyclic Model)

Il modello **LiNGAM (Linear Non-Gaussian Acyclic Model)** è stato sviluppato per affrontare il problema dell'identificazione causale in situazioni in cui la relazione tra due variabili è **lineare ma non gaussiana**. L'idea alla base di LiNGAM è che, se il modello generativo dei dati segue una relazione lineare e gli errori non sono gaussiani, allora è possibile determinare in modo univoco la direzione causale.

Un modello causale lineare per due variabili può essere espresso come:

$$Y = bX + \varepsilon$$

dove $\varepsilon$ è un termine di errore **indipendente da** $X$.

Se i residui della regressione rispettano questa indipendenza, allora possiamo affermare che $X$ è la causa di $Y$. Tuttavia, se proviamo a invertire la regressione, ossia modelliamo $X$ come funzione di $Y$, è probabile che i residui della regressione inversa **non** siano indipendenti da $Y$, indicando che la direzione scelta è errata.

Questa proprietà consente di identificare la direzione causale in maniera rigorosa quando almeno una delle variabili o degli errori non segue una distribuzione gaussiana.

Il modello LiNGAM può essere espresso in forma matriciale come:

$$X = BX + E$$

dove $B$ è una matrice triangolare inferiore stretta e $E$ rappresenta un vettore di errori indipendenti.

### Estensioni del Modello LiNGAM

Una limitazione di LiNGAM è che assume **assenza di cicli** nella struttura causale e non considera la presenza di variabili latenti. Per superare questi problemi, sono state proposte diverse estensioni:

- **DirectLiNGAM**: utilizza una strategia iterativa di regressione e test di indipendenza per determinare l'ordine causale tra le variabili.
- **ICA-LiNGAM**: combina l'analisi delle componenti indipendenti (ICA) con LiNGAM per migliorare la stima della struttura causale.

- **LiNGAM con confondenti latenti**: utilizza ICA sovracompleto per stimare effetti latenti, consentendo di rilevare variabili non misurate che influenzano il modello.

Un aspetto cruciale nell'analisi causale basata su LiNGAM è la **non-gaussianità** degli errori. Secondo il **teorema di Darmois-Skitovich**, se almeno una delle due variabili $X$ o $\varepsilon$ non è gaussiana, allora la direzione causale può essere identificata.

Inoltre, il **teorema di decomposizione di Cramér** dimostra che la somma di due variabili indipendenti è gaussiana **solo se entrambe le variabili sono gaussiane**. Questo implica che la gaussianità è una condizione speciale e che, nella maggior parte dei casi reali, la **non-gaussianità** è una proprietà utile per la scoperta causale.

### Modelli Non Lineari

In molti contesti reali, le relazioni tra variabili non sono puramente lineari. Per affrontare questo problema, sono stati sviluppati modelli causali che incorporano **trasformazioni non lineari** nei dati.

### Modello di Rumore Additivo Non Lineare

Una generalizzazione del modello LiNGAM è il **modello di rumore additivo non lineare**, in cui la relazione tra causa ed effetto è rappresentata come:

$$Y = f(X) + \varepsilon$$

dove $f(X)$ è una funzione **non lineare** della causa e $\varepsilon$ è un errore indipendente da $X$.

Questo modello permette di catturare relazioni più complesse rispetto al modello lineare, ma introduce anche sfide computazionali maggiori. Se il modello funzionale è troppo restrittivo rispetto al vero processo generativo dei dati, l'inferenza causale può essere fuorviante.

### Modello Post-NonLinear (PNL)

Per superare le limitazioni dei modelli additivi, è stato proposto il **modello PNL (Post-NonLinear)**, che tiene conto non solo dell'influenza non lineare della causa, ma anche di possibili **distorsioni di misura o effetti sensoriali** nelle variabili osservate.

Questo modello ha la forma:

$$Y = g(f(X)) + \varepsilon$$

dove $g$ è una trasformazione ulteriore applicata alla variabile dipendente.

Il modello PNL è il più generale tra i modelli di **Funzioni Causali Modellabili (FCM)**, ed è stato dimostrato che **la direzione causale può essere identificata nella maggior parte dei casi**, a meno che le distribuzioni e le funzioni coinvolte rispettino particolari condizioni di simmetria.

**Problemi Pratici nella Scoperta Causale**

L'applicazione degli algoritmi di scoperta causale a dati reali è complicata da numerosi problemi pratici, tra cui:

1. **Causalità nelle serie temporali**: le serie temporali rappresentano una sfida particolare, poiché le relazioni causali possono cambiare nel tempo. Metodi come la **Granger Causality** sono ampiamente utilizzati, ma possono essere sensibili alla frequenza di campionamento.

2. **Dati non stazionari ed eterogenei**: la struttura causale può variare nel tempo o tra diversi insiemi di dati, rendendo difficile l'identificazione di leggi causali stabili.

3. **Errori di misura**: possono alterare le relazioni di indipendenza condizionale osservate, portando a conclusioni errate. Sono stati sviluppati metodi specifici per correggere questi errori.

4. **Bias di selezione**: se la probabilità di includere un campione dipende da alcune sue caratteristiche, il risultato dell'analisi può essere distorto.

5. **Dati mancanti**: molte applicazioni pratiche presentano dati mancanti, che possono influenzare l'inferenza causale. Algoritmi modificati dell'algoritmo PC sono stati sviluppati per affrontare questa problematica.

L'integrazione di metodi avanzati, combinando test di indipendenza condizionale con modelli non lineari, sta migliorando progressivamente la scoperta causale. Tuttavia, rimangono ancora molte sfide aperte, specialmente in contesti con **campioni di piccole dimensioni, forti confondenti latenti e dati rumorosi**.

# Capitolo 3

# Generazioni di Modelli Declare attraverso la Scoperta di Relazioni Causali tra Dati

Rispetto ai metodi tradizionalmente proposti in letteratura, l'approccio adottato in questa tesi mira a migliorare l'accuratezza nel processo di apprendimento dei vincoli DECLARE . Questo obiettivo viene raggiunto non solo attraverso la generazione di modelli che incorporano la causalità, ma anche superando una limitazione comune alla maggior parte degli approcci esistenti, ovvero la mancata considerazione dei dati relativi alle attività del Log. L'obiettivo principale è integrare le relazioni di causalità all'interno degli algoritmi di apprendimento, superando la mera selezione di vincoli basata su un ordinamento frequentista, dove i vincoli selezionati sono quelli che soddisfano il Log con una frequenza maggiore. In altre parole, anziché limitarsi a un'analisi che privilegia esclusivamente la frequenza con cui determinati schemi si manifestano nei dati, il nostro metodo mira a valorizzare, ove presenti, le attività che risultano legate da una relazione causale. Questa prospettiva introduce un cambiamento sostanziale nell'approccio all'apprendimento dei vincoli, poiché consente di identificare non solo correlazioni statistiche ma anche legami di dipendenza più profondi tra le attività analizzate. In tal modo, il modello proposto risulta maggiormente in grado di cogliere la struttura sottostante dei processi esaminati, migliorando la capacità predittiva e interpretativa degli algoritmi utilizzati.

Per raggiungere questo obiettivo, è stato sviluppato un algoritmo strutturato in più fasi, che segue un approccio sistematico per identificare relazioni causali e vincoli *Declare*. Il processo può essere formalizzato come segue:

1. **Estrazione dei dati dal log:** Il log è strutturato in tracce, ciascuna composta da una sequenza di attività, ognuna delle quali è associata a determinati dati (es. timestamp, prezzo, ID acquirente, ecc.). Formalmente, possiamo rappresentare il log come:

$$L = \{\tau_1, \tau_2, \ldots, \tau_T\}$$

dove ogni traccia $\tau_i$ è una sequenza ordinata di attività:

$$\tau_i = \langle a_1^{(i)}, a_2^{(i)}, \ldots, a_{n_i}^{(i)} \rangle$$

con:

- $a_j^{(i)}$ che rappresenta un'istanza dell'attività $a_j$ nella traccia $\tau_i$,
- $n_i$ il numero di attività nella traccia $\tau_i$,
- $T$ il numero totale di tracce nel log.

A partire dal log, costruiamo un dataframe $D$ in cui ogni riga rappresenta una singola istanza di un'attività, estratta da una delle tracce. Quindi, ogni attività $a_j$ eseguita nella traccia $\tau_i$ genera una riga nel dataframe con i relativi dati osservabili $x_k$:

$$D = \left\{ (i, j, a_j^{(i)}, x_1^{(i,j)}, x_2^{(i,j)}, \ldots, x_m^{(i,j)}) \mid a_j^{(i)} \in \tau_i, 1 \leq i \leq T, 1 \leq j \leq n_i \right\}$$

Dove:

- $i$ è l'indice della traccia da cui proviene l'attività,
- $j$ è la posizione dell'attività nella traccia,
- $a_j^{(i)}$ è l'attività eseguita nella traccia $\tau_i$,
- $x_k^{(i,j)}$ è il valore del $k$-esimo attributo per quell'istanza dell'attività,
- $m$ è il numero di attributi disponibili per ciascuna attività.

2. **Apprendimento delle relazioni causali:** Il dataset $D$ viene dato in ingresso a un algoritmo di *causal learning* $\mathcal{L}$, che apprende le relazioni tra le variabili identificando la struttura del grafo causale $G$.

3. **Individuazione delle relazioni causali tra attività:** Se due variabili $x_i$ e $x_j$, corrispondenti ad attività distinte $a_i, a_j$, sono collegate da un arco causale nel grafo $G$, allora si assume una relazione causale tra le attività corrispondenti:
$$(x_i \to x_j) \Rightarrow (a_i \to a_j)$$

4. **Estrazione dei vincoli Declare:** Tramite algoritmi di learning, si estraggono tutti i vincoli *Declare* $C$ che soddisfano le tracce del log. Questi vincoli definiscono proprietà temporali e logiche che devono essere rispettate nel processo:
$$C = \{c_1, c_2, \ldots, c_k\}$$
dove ogni vincolo può essere espresso in forma logica, ad esempio:
$$\forall t_1, t_2, \quad (a_i(t_1) \Rightarrow a_j(t_2)) \text{ con } t_1 < t_2$$

5. **Generazione del modello finale:** Dopo aver estratto l'insieme di vincoli *Declare* $C$, viene generato il modello finale, dando priorità ai vincoli relativi ad attività che presentano una relazione causale. Tuttavia, il modello non contiene solo questi vincoli, ma include anche gli altri, strutturati in modo che quelli con una base causale vengano enfatizzati.

   Formalmente, possiamo definire due sottoinsiemi:

- $C_{\text{caus}} \subseteq C$ che contiene i vincoli tra attività che hanno una relazione causale,
- $C_{\text{rest}} = C \setminus C_{\text{caus}}$ che contiene tutti gli altri vincoli.

Il modello iniziale viene costruito come:

$$M = C_{\text{caus}} \cup C_{\text{rest}}$$

dove gli elementi di $C_{\text{caus}}$ hanno una priorità più alta rispetto a quelli di $C_{\text{rest}}$, garantendo che i vincoli basati su relazioni causali abbiano maggiore rilevanza.

Dopo questa fase, il modello viene ottimizzato attraverso un processo di eliminazione dei vincoli ridondanti, sussunti o derivati.

- Un vincolo **ridondante** è un vincolo che può essere inferito da altri già presenti nel modello.
- Un vincolo **sussunto** è un vincolo che è implicitamente soddisfatto da un altro più generale (es. se $A$ deve precedere $B$ e $B$ deve precedere $C$, allora $A$ precede $C$ è ridondante).
- Un vincolo **derivato** è un vincolo che può essere ottenuto logicamente dalla combinazione di più vincoli esistenti.

Questo processo può essere formalizzato come:

$$M' = M \setminus \{c \in M \mid c \text{ è ridondante, sussunto o derivato}\}$$

ottenendo un modello finale ottimizzato:

$$M_{\text{final}} = M'$$

che mantiene la priorità dei vincoli causali e riduce la complessità eliminando quelli superflui.

Nei prossimi paragrafi verranno illustrati in dettaglio i diversi passaggi seguiti durante lo sviluppo di questo approccio, i ragionamenti teorici che ne hanno guidato la formulazione e i risultati ottenuti attraverso la sperimentazione.

## 3.1 Generazione di Log con Relazioni Causali tra Dati

Il primo passo della nostra pipeline consiste nella generazione di un dataset contenente le relazioni causali necessarie. La scelta della struttura del grafo mostrato in Figura 3.2 non è casuale: come primo approccio, era essenziale verificare che gli algoritmi di causal learning impiegati fossero in grado non solo di identificare correttamente le relazioni causali inserite nel dataset, ma anche di evitare il rilevamento di relazioni inesistenti, garantendo così la coerenza con la struttura del Log sintetico creato.

Definita la struttura del grafo causale, il passo successivo è stata la generazione dei dati. In particolare, abbiamo simulato il Log di un'azienda che offre spedizione gratuita in base al valore dell'acquisto. Pertanto, i dati sono stati generati modellando le attività di acquisto e spedizione in modo da riflettere esattamente la relazione causale che ci eravamo prefissati.

Un elemento chiave è la definizione della variabile `num_items` (numero di articoli acquistati), che viene generata casualmente entro un intervallo predefinito. Da questa informazione si calcola il prezzo totale dell'acquisto `total_price`, il quale non è semplicemente il prodotto tra il numero di articoli e un prezzo fisso, ma viene determinato assegnando un valore casuale a ciascun articolo per simulare una variabilità realistica nei prezzi.

Listing 3.1: Calcolo del prezzo totale dell'acquisto

```
ALGORITMO CalcolaPrezzoTotale(num_items)
    INIZIALIZZA prezzi con num_items numeri casuali tra 20 e 70
    prezzo_totale     SOMMA(prezzi)
    RESTITUISCI prezzo_totale
FINE ALGORITMO
```

Dopo aver calcolato il totale dell'acquisto, viene determinato il costo della spedizione `shipping_price`, che dipende sia dal totale speso sia dal numero di articoli acquistati. La logica implementata segue le seguenti regole condizionali:

- Se il prezzo totale supera 40€, la spedizione è gratuita.
- Se il prezzo totale è inferiore ma il numero di articoli è maggiore di 8, viene applicato un costo di spedizione di 20€.
- In tutti gli altri casi, il costo di spedizione è di 10€.

Listing 3.2: Calcolo del costo di spedizione

```
ALGORITMO CalcolaCostoSpedizione(total_price, num_items)
    SE total_price > 40 ALLORA
        RESTITUISCI 0
    ALTRIMENTI SE num_items > 8 ALLORA
        RESTITUISCI 20
    ALTRIMENTI
        RESTITUISCI 10
    FINE SE
FINE ALGORITMO
```

Questa modellazione causale assicura che il costo della spedizione non sia determinato casualmente, ma segua una logica coerente con il comportamento reale di una piattaforma di e-commerce come si evince dalla distribuzione dei dati in Figura 3.1. Una volta definite queste relazioni, il dataset viene popolato generando eventi con timestamp ordinati, garantendo una progressione temporale tra le attività. Per ogni transazione:

- Si assegna un timestamp di acquisto.
- Dopo un intervallo casuale, si registra un timestamp di spedizione coerente con l'acquisto.
- Con una probabilità del 30%, si genera un evento di registrazione contenente nome, cognome, età e timestamp.

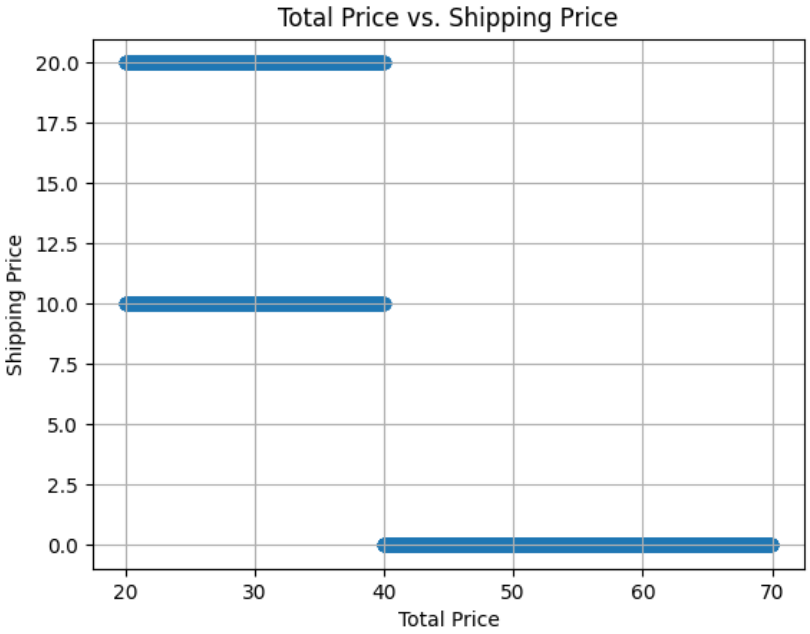



Figure 3.1: Distribuzione Dati

- Con una probabilità del 40%, si genera un evento di feedback con rating casuale tra 1 e 5.

Listing 3.3: Generazione dei dati sintetici

```
ALGORITMO GeneraDati(n)
    INIZIALIZZA lista vuota data
    IMPOSTA timestamp attuale

    PER i da 1 a n FAI
        GENERA dati di acquisto: timestamp, num_items, total_price,
             shipping_price, timestamp_spedizione
        AGGIUNGI dati all'elenco data

        SE condizione casuale soddisfatta ALLORA
            AGGIUNGI registrazione utente
        FINE SE

        SE condizione casuale soddisfatta ALLORA
            AGGIUNGI feedback cliente
        FINE SE
    FINE PER

    RESTITUISCI tabella data
FINE ALGORITMO
```

Dopo aver generato i dati, questi vengono esportati nei formati `XES` e `CSV` che serviranno per l'analisi causale tramite gli algoritmi di learning.

Infine, il grafo causale generato, come mostrato in Figura 3.2, viene visualizzato per confermare la corretta modellazione delle relazioni tra le variabili e

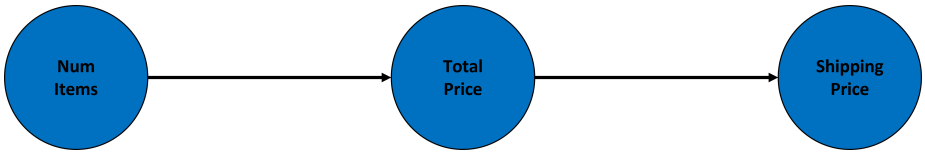



Figure 3.2: Grafico CausalGen

la coerenza dell'intero processo di generazione dati.

Listing 3.4: Visualizzazione del grafo causale

```
causal_graph.visualize()
```

Grazie a questa pipeline, è stato possibile ottenere un dataset strutturato e realistico, con una logica causale ben definita tra acquisti e spedizioni, garantendo una rappresentazione fedele dei fenomeni simulati.

## 3.2 Verifica delle Relazioni Causali tramite Algoritmi di Learning

Dopo aver generato un dataset strutturato con relazioni causali ben definite, il passo successivo consiste nel verificare se i modelli di *causal discovery* siano in grado di recuperarle correttamente. Questo è fondamentale per validare l'affidabilità degli algoritmi di apprendimento causale, che potrebbero poi essere utilizzati per la modellazione di vincoli basati sulla causalità invece che su semplici correlazioni.

A tale scopo, utilizziamo gli algoritmi **PC (Peter-Clark)** e **FCI (Fast Causal Inference)**, implementati nella libreria `Causal-learn` [8]. L'algoritmo PC esegue un'analisi basata su test di indipendenza condizionale per costruire un grafo causale, mentre FCI è un'estensione che permette di gestire variabili latenti e confondenti [9].

### 3.2.1 Preparazione dei dati per l'analisi causale

Per applicare gli algoritmi di causal discovery, è necessario selezionare esclusivamente le colonne numeriche del dataset, in modo da escludere variabili categoriche o identificatori che non sono utili per l'analisi causale.

Listing 3.5: Selezione delle variabili numeriche per l'analisi causale

```
ALGORITMO SelezionaVariabiliNumeriche(df)
    IDENTIFICA colonne numeriche in df
    ESTRARRE valori delle colonne numeriche
    VISUALIZZA colonne selezionate
FINE ALGORITMO
```

Ci aspettiamo che in questa fase vengano mantenute solo le variabili effettivamente utili all'inferenza causale, come il prezzo totale dell'acquisto, il numero di articoli acquistati, il costo della spedizione e i timestamp normalizzati.

**Processo di Suddivisione**

L'analisi dei log degli eventi rappresenta una sfida complessa a causa della naturale eterogeneità delle tracce presenti nei dati. Ogni traccia è costituita da una sequenza di eventi, ma non tutte le tracce seguono lo stesso schema. Questa variabilità introduce difficoltà significative nell'analisi causale, poiché un'aggregazione indiscriminata delle tracce con strutture differenti potrebbe generare problemi dovuti alla presenza di valori nulli (NaN), con conseguenze negative sulla qualità dell'inferenza causale.

Per ovviare a questo problema, è necessario adottare una strategia che consenta di suddividere il dataset in gruppi omogenei, in modo che ciascun sottoinsieme contenga solo tracce con la stessa struttura di eventi. Questa suddivisione permette:

- **Migliore qualità dei dati**: eliminando i valori nulli, i test di indipendenza condizionale utilizzati dagli algoritmi PC e FCI risultano più affidabili.
- **Grafo causale più interpretabile**: ogni rete causale rappresenta in modo chiaro il comportamento specifico di un determinato tipo di traccia.
- **Migliore generalizzabilità**: se un tipo di traccia ha relazioni causali diverse da un altro, il modello causale risultante riflette tali differenze invece di forzare un'unica struttura per tutti i dati.

Il processo seguito per ottenere questa suddivisione può essere articolato in tre fasi principali:

1. Identificazione della sequenza di eventi presente in ciascuna traccia e creazione di una mappatura che assegni un identificativo numerico (`trace_type`) a ogni combinazione unica di eventi.
2. Suddivisione del dataset principale (`df`) in sotto-dataset distinti, così che ogni gruppo contenga solo tracce con la stessa struttura di eventi.
3. Applicazione degli algoritmi PC e FCI per inferire le relazioni causali, con salvataggio dei risultati sotto forma di grafi aciclici diretti (DAG) e grafi più complessi in presenza di variabili latenti.

Questa metodologia consente di ottenere una rappresentazione più chiara delle dipendenze causali nei dati e permette di verificare se gli algoritmi di causal learning siano in grado di individuare le relazioni effettivamente presenti nelle tracce.

### 3.2.2 Suddivisione delle Tracce per Tipologia

La prima operazione necessaria per rendere il dataset analizzabile consiste nella classificazione delle tracce in base alla sequenza di eventi che le compongono. Poiché non tutte le tracce includono gli stessi eventi, è opportuno assegnare un identificativo numerico (`trace_type`) a ogni combinazione unica di eventi, in modo da garantire che ogni tipologia di traccia possa essere analizzata separatamente.

Questa classificazione viene effettuata costruendo inizialmente un insieme (`unique_event_sequences`) contenente tutte le sequenze di eventi presenti nel dataset. Successivamente, viene generato un dizionario (`type_mapping`) che associa a ogni sequenza unica un numero identificativo.

Il codice che implementa questa operazione è il seguente:

Listing 3.6: Mappatura delle sequenze di eventi in tipi numerici

```
ALGORITMO CreaMappaturaSequenze(unique_event_sequences)
    INIZIALIZZA type_mapping come dizionario vuoto
    ASSEGNA a ogni sequenza un valore numerico univoco
    INIZIALIZZA lista vuota trace_to_type
FINE ALGORITMO
```

Questa mappatura consente di trasformare ogni combinazione di eventi in un valore numerico, semplificando così la gestione e la classificazione delle tracce. Ad esempio, se nel dataset sono presenti quattro tipologie di tracce con strutture differenti, il dizionario risultante sarà strutturato nel seguente modo:

Listing 3.7: Esempio Mappatura

```
type_mapping = {
    ("Acquisto", "Spedizione"): 1,
    ("Registrazione", "Acquisto", "Spedizione"): 2,
    ("Acquisto", "Spedizione", "Feedback"): 3,
    ("Registrazione", "Acquisto", "Spedizione", "Feedback"): 4
}
```

### 3.2.3 Assegnazione delle Tracce ai rispettivi `trace_type`

A questo punto, il codice assegna a ogni traccia il valore `trace_type` corrispondente alla sua sequenza di eventi. Questa operazione avviene iterando su tutte le righe del DataFrame `df` e identificando gli eventi effettivamente presenti per ciascuna traccia.

Listing 3.8: Assegnazione del tipo di traccia in base alla sequenza di eventi

```
ALGORITMO AssegnaTipoTraccia(df, type_mapping)
    INIZIALIZZA lista vuota trace_to_type

    PER ogni riga in df FAI
        IDENTIFICA eventi presenti nella traccia
        ORDINA e rimuovi duplicati per ottenere event_sequence

        SE event_sequence non    in type_mapping ALLORA
            MOSTRA avviso e salta la traccia
        ALTRIMENTI
            ASSEGNA trace_type corrispondente
            AGGIUNGI trace_type a trace_to_type
        FINE SE
    FINE PER

    AGGIUNGI trace_to_type come nuova colonna "trace_type" in df
FINE ALGORITMO
```

Una volta completata l'iterazione su tutte le tracce, la lista `trace_to_type` contenente i valori `trace_type` viene aggiunta al DataFrame `df`, permettendo di suddividere successivamente il dataset in gruppi omogenei. Il risultato sarà una tabella in cui ogni traccia è associata a una specifica tipologia.

### 3.2.4 Preparazione dei Dati per l'Analisi Causale

Dopo aver suddiviso il dataset principale in base alla tipologia di traccia, è necessario trasformare i dati in un formato compatibile con gli algoritmi PC e FCI. Questi modelli operano eseguendo test di indipendenza condizionale su dati numerici, per cui è fondamentale eliminare tutte le colonne contenenti informazioni testuali o categoriche e convertire i dati in un array NumPy.

Questa operazione è implementata nel codice attraverso un ciclo che scorre ogni tipologia di traccia (`trace_type`), estrae le colonne numeriche e le trasforma in una matrice utilizzabile dagli algoritmi di discovery causale.

Listing 3.9: Preparazione dei dati numerici per l'analisi causale

```
ALGORITMO PreparaDatiCausali(df)
    PER ogni trace_type univoco in df["trace_type"] FAI
        FILTRA df per ottenere df_trace con il tipo di traccia
            corrente

        SELEZIONA colonne numeriche in df_trace
        CONVERTI i dati numerici in matrice

        MOSTRA informazioni sulla traccia elaborata (numero di
            campioni e variabili)
    FINE PER
FINE ALGORITMO
```

Questa fase garantisce che ogni sotto-dataset (`df_trace`) contenga esclusivamente variabili numeriche, evitando interferenze da dati testuali che potrebbero compromettere l'efficacia dei test di indipendenza condizionale. Dopo la selezione, i dati vengono convertiti in array NumPy bidimensionali, nel formato richiesto dagli algoritmi di discovery causale.

### 3.2.5 Inferenza delle Relazioni Causali tramite PC e FCI

Dopo aver suddiviso le tracce in gruppi omogenei e convertito i dati in un formato compatibile con gli algoritmi di causal discovery, il passo finale consiste nell'applicazione degli algoritmi PC (Peter-Clark) e FCI (Fast Causal Inference). L'obiettivo è verificare se gli algoritmi sono in grado di individuare le relazioni di dipendenza causale presenti nel dataset e confrontarle con la struttura causale effettiva, nota a priori grazie al processo di generazione dei dati.

L'algoritmo PC si basa su test di indipendenza condizionale per costruire un grafo aciclico diretto (DAG), assumendo che non esistano variabili latenti. Questo modello parte da un grafo completamente connesso e procede eliminando progressivamente gli archi tra i nodi sulla base dei risultati dei test di indipendenza. L'algoritmo FCI, invece, rappresenta un'estensione del PC che permette di gestire scenari con variabili latenti e confondenti, generando un Partial Ancestral Graph (PAG) che tiene conto dell'incertezza nelle relazioni causali.

L'esecuzione di questi algoritmi su ciascuna tipologia di traccia consente di ottenere una rappresentazione chiara delle relazioni causali identificate, fornendo così un'importante base per la validazione del modello e per l'interpretazione delle relazioni tra gli eventi nei log analizzati.

### Applicazione degli Algoritmi di Discovery Causale

L'implementazione segue un approccio iterativo: per ogni tipologia di traccia (`trace_type`), viene estratto il sotto-dataset numerico e vengono applicati gli algoritmi PC e FCI. I risultati vengono poi salvati sotto forma di grafi causali, che possono essere analizzati e visualizzati successivamente.

Listing 3.10: Generazione di grafi causali con PC e FCI

```
ALGORITMO GeneraGrafiCausali(df)
    PER ogni trace_type univoco in df["trace_type"] FAI
        SELEZIONA dati numerici per il trace_type corrente

        # Algoritmo PC
        ESEGUE PC Algorithm sui dati numerici
        CONVERTE il risultato in grafo
        SALVA il grafo come immagine PNG

        # Algoritmo FCI
        ESEGUE FCI Algorithm sui dati numerici
        CONVERTE il risultato in grafo
        SALVA il grafo come immagine PNG
    FINE PER
FINE ALGORITMO
```

Questo codice esegue i seguenti passaggi:

1. Itera su tutte le tipologie di tracce presenti nel dataset, garantendo che ogni tipologia venga analizzata separatamente.
2. Estrae solo le colonne numeriche del sotto-dataset corrispondente, convertendole in un array NumPy (`numeric_data`).
3. Esegue l'algoritmo PC, generando un grafo causale in formato DAG e salvandolo in un file `.png` per la successiva visualizzazione.
4. Esegue l'algoritmo FCI, che produce una rappresentazione causale più complessa, in grado di gestire variabili latenti e confondenti, anch'essa salvata in formato `.png`.

### Esecuzione dell'Algoritmo PC

L'algoritmo PC è uno dei metodi più noti per la discovery causale basata su vincoli. Questo modello assume che non vi siano variabili latenti, cioè che tutte le dipendenze osservate nei dati siano effettivamente rappresentative della struttura causale reale. L'algoritmo opera in tre fasi:

- **Fase 1 (Skeleton Identification)**: Si parte da un grafo completamente connesso tra tutte le variabili. Successivamente, vengono eseguiti test di indipendenza condizionale per rimuovere progressivamente gli archi non supportati dai dati.
- **Fase 2 (V-structure Identification)**: Una volta determinata la struttura del grafo, vengono identificati i nodi intermedi nelle relazioni causali (le cosiddette V-structures, ovvero pattern del tipo $A \rightarrow C \leftarrow B$).

- **Fase 3 (Edge Orientation)**: Infine, vengono applicate delle regole di orientamento per determinare la direzione degli archi, completando la costruzione del DAG.

Nel codice, l'implementazione dell'algoritmo PC avviene con la funzione:

```
cg_pc = pc(numeric_data, alpha=0.01, stable=True)
```

Il parametro `alpha=0.01` indica il livello di significatività dei test di indipendenza: valori più piccoli rendono il modello più conservativo nel rimuovere connessioni. L'opzione `stable=True` assicura un'esecuzione deterministica dell'algoritmo, garantendo che i risultati siano riproducibili.

L'output prodotto è un grafo diretto aciclico (DAG), che fornisce una rappresentazione strutturata delle dipendenze causali individuate nei dati.

### Esecuzione dell'Algoritmo FCI

Mentre PC assume che tutti i nodi osservati nel dataset siano completamente rappresentativi della realtà, l'algoritmo FCI rilassa questa ipotesi e permette di modellare scenari con variabili latenti e confondenti. Questo lo rende particolarmente utile per analisi su dati reali, dove alcune informazioni potrebbero non essere direttamente osservabili.

L'algoritmo FCI segue una logica simile a PC, ma introduce ulteriori passaggi per tenere conto dell'incertezza nelle direzioni causali:

1. **Skeleton Identification**: Come in PC, si parte da un grafo completamente connesso e si eliminano progressivamente gli archi in base ai test di indipendenza condizionale.
2. **Orientation Rules**: Gli archi vengono orientati applicando regole che consentono di distinguere tra relazioni dirette e relazioni spurie dovute a variabili latenti.
3. **Partial Ancestral Graph (PAG) Construction**: Poiché non è possibile determinare con certezza tutte le relazioni causali a causa della presenza di variabili latenti, il risultato finale è un PAG, che rappresenta le possibili dipendenze causali compatibili con i dati osservati.

Nel codice, l'algoritmo viene eseguito tramite:

```
g_fci, edges_fci = fci(numeric_data, alpha=0.01)
```

in questo caso l'output prodotto è un Partial Ancestral Graph (PAG), che può contenere non solo archi diretti ($\rightarrow$), ma anche archi con incertezze direzionali ($\circ\!\rightarrow$), che indicano relazioni per le quali non è stato possibile determinare una direzione causale univoca.

### Conclusioni e Visualizzazione

Infine, i grafi causali generati (Figure 3.3 3.4 3.5 3.6) vengono visualizzati per confrontare i risultati ottenuti con i due metodi. L'analisi dei grafi permette di valutare se le relazioni causali predefinite nel dataset vengano correttamente identificate. Se il grafo generato corrisponde alla struttura causale di partenza, possiamo concludere che il metodo utilizzato è efficace per identificare dipendenze reali tra le variabili.

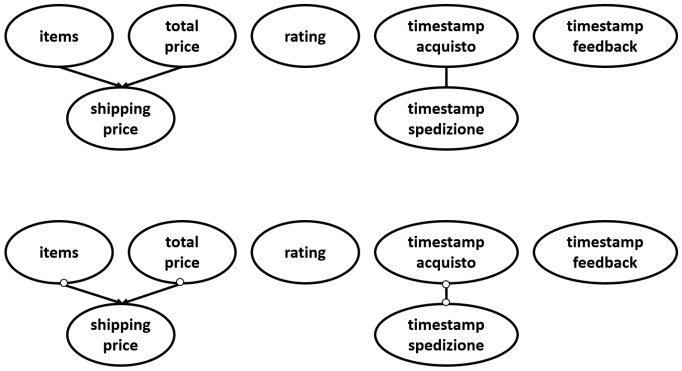



Figure 3.3: Grafici Causali FCI e PC della traccia 2

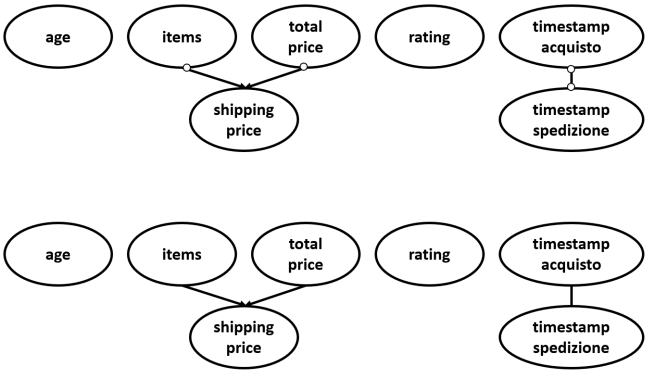



Figure 3.4: Grafici Causali FCI e PC della traccia 3

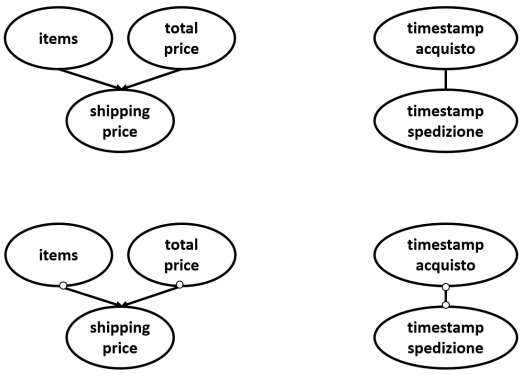



Figure 3.5: Grafici Causali FCI e PC della traccia 4


Figure 3.6: Grafici Causali FCI e PC della traccia 1

## 3.3 Learning di Vincoli Declare dal file di Log

### 3.3.1 Analisi della suddivisione del log in sotto-log omogenei

L'obiettivo principale di questa fase è superare la limitazione di NegDis, che restituisce esclusivamente i vincoli soddisfatti nel 100% delle tracce del log. Tuttavia, per un'analisi più dettagliata e realistica, è necessario identificare anche quei vincoli che, pur non essendo sempre rispettati, possono offrire informazioni rilevanti sulla struttura del processo.

Per risolvere questo problema, abbiamo deciso di adottare una strategia basata sulla suddivisione del log in sotto-log omogenei, in cui ogni sotto-log contiene solo tracce della stessa tipologia.

#### Suddivisione del Log

Questa suddivisione è motivata dal fatto che, se un sotto-log contiene solo un tipo di traccia, tutti i vincoli identificati per esso saranno necessariamente soddisfatti nel 100% delle tracce appartenenti a quel sotto-log. Ciò consente di ottenere un insieme più ampio di vincoli, che non sarebbe stato possibile estrarre analizzando il log originale nel suo complesso.

La prima operazione necessaria per realizzare questa suddivisione consiste nell'identificare tutte le tipologie di tracce presenti nel log. Questo viene fatto analizzando il file XES e memorizzando tutte le combinazioni uniche di eventi nelle tracce. Il seguente codice implementa questa operazione:

Listing 3.11: Estrazione dei tipi di tracce da un file XES

```
ALGORITMO EstraiTipiTracce(xes_file)
    LOG "Analisi del file XES per estrarre i tipi di tracce..."
    CARICA e ANALIZZA il file XES
    INIZIALIZZA insieme vuoto unique_event_sequences

    PER ogni traccia nel file XES FAI
        INIZIALIZZA event_sequence come lista vuota
        PER ogni elemento della traccia FAI
            SE    un evento e contiene "concept:name" ALLORA
                AGGIUNGI il valore dell'evento a event_sequence
            FINE SE
        FINE PER

        SE event_sequence non    vuota ALLORA
            AGGIUNGI la versione ordinata e senza duplicati di
                event_sequence a unique_event_sequences
        FINE SE
    FINE PER

    LOG numero di tipi di tracce trovati
    RESTITUISCI unique_event_sequences
FINE ALGORITMO
```

Il codice effettua il parsing del file XES e raccoglie tutte le sequenze di eventi uniche presenti nel dataset. Utilizziamo un `set` per eliminare automaticamente eventuali duplicati e garantire che ogni sequenza venga considerata una sola volta.

Una volta completata questa fase, disponiamo di un elenco di tipologie di tracce che ci consentirà di suddividere il log in sotto-log distinti. L'uso di un `set` è particolarmente vantaggioso perché consente di individuare rapidamente tutte le strutture di eventi uniche, riducendo il tempo di elaborazione.

Questa strategia offre diversi vantaggi:

- Evita la perdita di vincoli potenzialmente utili che potrebbero essere validi solo per determinate tipologie di tracce.
- Permette di ottenere risultati più dettagliati e di ottimizzare l'esecuzione di NegDis, che lavora meglio su log più omogenei.
- Consente una rappresentazione più chiara delle dipendenze tra gli eventi e una maggiore precisione nell'analisi causale.

La suddivisione del log, quindi, non solo migliora l'efficacia del discovery dei vincoli, ma garantisce anche una migliore interpretazione dei risultati ottenuti.

### 3.3.2 Creazione dei Sotto-Log

Dopo aver identificato le diverse tipologie di tracce presenti nel log, il passo successivo consiste nella creazione di sotto-log distinti, ognuno contenente solo tracce di una determinata tipologia. Questo passaggio è fondamentale perché permette di applicare NegDis separatamente su ciascun sotto-log, garantendo che i vincoli estratti siano validi al 100% delle tracce presenti in quel sotto-log.

La logica alla base di questa operazione è che, se un vincolo viene scoperto in un sotto-log contenente esclusivamente tracce della stessa tipologia, allora i vincoli trovati da NegDis per quel sotto-log saranno validi nel 100% dei casi.

Listing 3.12: Generazione dei file XES per ogni tipologia di traccia

```
ALGORITMO GeneraFileXES(xes_file, unique_event_sequences)
    CARICA il file XES
    CREA directory per i file delle tracce, se non esiste
    INIZIALIZZA lista xes_files

    PER ogni event_sequence univoca FAI
        CREA nuovo file XES vuoto
        PER ogni traccia nel file originale FAI
            ESTRAI eventi della traccia
            SE gli eventi corrispondono alla event_sequence ATTUALE
                 ALLORA
                AGGIUNGI traccia al nuovo file XES
            FINE SE
        FINE PER

        SALVA il nuovo file XES
        REGISTRA il file creato nella lista xes_files
        LOG nome del file creato
    FINE PER

    RESTITUISCI xes_files
FINE ALGORITMO
```

Dopo aver caricato il file XES, viene creato un nuovo sotto-log per ciascuna delle tipologie di tracce identificate nella fase precedente. Per ogni traccia nel

log originale, il codice controlla la sequenza di eventi che contiene e, se corrisponde a una delle sequenze uniche individuate, la copia in un nuovo file XES corrispondente.

L'output finale è una directory contenente $N$ sotto-log, ognuno corrispondente a un tipo specifico di traccia. Questo approccio presenta diversi vantaggi:

- Aumenta la precisione dell'analisi, garantendo che i vincoli estratti siano specifici per ogni tipologia di traccia.
- Evita la perdita di informazioni, consentendo di identificare vincoli che potrebbero essere validi solo per determinate categorie di tracce.

Questa suddivisione prepara il terreno per l'applicazione di NegDis, permettendoci di ottenere vincoli più dettagliati e significativi rispetto a un'analisi globale del log originale.

### 3.3.3 Applicazione di NegDis su Ogni Sotto-Log

Dopo aver suddiviso il log originale in sotto-log omogenei, possiamo procedere con l'applicazione di NegDis su ciascun sotto-log generato. Questa operazione ci permette di eseguire il discovery dei vincoli in maniera mirata, evitando il problema iniziale che limitava l'analisi ai soli vincoli soddisfatti nel 100% delle tracce del log completo.

L'idea alla base di questa fase è che, analizzando ogni sotto-log separatamente, i vincoli individuati saranno necessariamente validi per tutte le tracce contenute al suo interno. In altre parole, dato che ciascun sotto-log contiene solo un tipo di traccia, i vincoli scoperti non sono influenzati dalla presenza di altre strutture di eventi nel dataset. Questo approccio migliora la precisione dell'analisi, permettendo di identificare vincoli specifici per ogni classe di tracce.

L'esecuzione di NegDis su ciascun sotto-log viene gestita dal seguente codice:

Listing 3.13: Esecuzione di NegDis su ogni sotto-log

```
ALGORITMO EseguiNegDis(xes_files)
    INIZIALIZZA lista compatible_files

    PER ogni xes_file in xes_files FAI
        OTTIENI trace_type dal nome del file
        CREA il nome del file di output per i risultati compatibili

        LOG "Esecuzione NegDis su xes_file"
        COSTRUISCI il comando per NegDis

        PROVA a eseguire il comando
            SE eseguito con successo ALLORA
                LOG output del comando
                AGGIUNGI il file compatibile alla lista
            ALTRIMENTI
                LOG errore nell'esecuzione
        FINE PROVA
    FINE PER

    RESTITUISCI compatible_files
FINE ALGORITMO
```

Per ogni sotto-log generato viene eseguito un comando NegDis in modalità `discover compatible`, che ha il compito di individuare i vincoli compatibili con il sotto-log in esame. Il tool prende in input il file XES della traccia, il file contenente i template dei vincoli (`DECL_TEMPLATES`) e genera un file di output (`compatible_file`) contenente tutti i vincoli scoperti.

L'uso di `subprocess.run` permette di eseguire il comando NegDis direttamente dal codice Python, catturando sia l'output che eventuali errori. Se l'esecuzione è completata con successo, il file di vincoli generato viene salvato e registrato nella lista `compatible_files`, che conterrà tutti i risultati ottenuti per i vari sotto-log.

L'approccio adottato presenta diversi vantaggi:

- **Precisione**: ogni sotto-log viene analizzato indipendentemente, consentendo di identificare vincoli specifici per ciascun tipo di traccia.
- **Scalabilità**: l'esecuzione di NegDis su log più piccoli riduce il carico computazionale rispetto all'analisi dell'intero dataset.
- **Maggiore copertura**: l'uso di sotto-log permette di individuare vincoli che potrebbero non essere soddisfatti nel 100% delle tracce globali, ma che sono comunque validi per sottogruppi specifici.

Grazie a questa fase, otteniamo un set più ampio e dettagliato di vincoli, che verranno poi unificati nella fase successiva per fornire una visione globale della struttura causale del processo.

### 3.3.4 Unione dei Vincoli Estratti

Dopo aver eseguito NegDis su ciascun sotto-log, disponiamo di un insieme di file contenenti i vincoli scoperti per ogni tipologia di traccia. Tuttavia, per ottenere un modello globale del processo, è necessario unificare tutti i vincoli in un unico file, eliminando eventuali duplicati. Questa fase di merge è fondamentale per consolidare i risultati e creare una rappresentazione coerente delle regole di vincolo presenti nel log.

L'operazione di unione avviene in più passaggi:

- Lettura di tutti i file di vincoli generati da NegDis.
- Filtraggio dei vincoli validi, assicurandoci che siano conformi a specifici pattern di vincolo.
- Eliminazione dei duplicati, in modo da conservare un'unica versione di ciascun vincolo.
- Salvataggio del file unificato, che servirà come base per la successiva fase di verifica nel log originale.

Il codice che implementa questa operazione è il seguente:

Listing 3.14: Unione dei file di vincoli generati da NegDis

```
1 ALGORITMO UnisciFileVincoli(compatible_files, output_file)
2     INIZIALIZZA insieme valid_constraints
3     DEFINISCI lista valid_patterns con i vincoli riconosciuti
4
```

```
    PER ogni file in compatible_files FAI
        APRI il file e LEGGI riga per riga
        PER ogni linea nel file FAI
            RIMUOVI spazi vuoti
            SE la linea inizia con un pattern valido ALLORA
                AGGIUNGI il vincolo a valid_constraints
            ALTRIMENTI
                LOG "Vincolo ignorato o errato"
            FINE SE
        FINE PER
    FINE PER

    SCRIVI i vincoli validi nel file di output
    LOG "File merged creato con successo"
FINE ALGORITMO
```

Il codice esegue un'iterazione su tutti i file di vincoli generati da NegDis, leggendo riga per riga ogni vincolo scoperto. Per garantire che solo vincoli validi e interpretabili vengano mantenuti, utilizziamo una lista di pattern predefiniti, che rappresentano le principali tipologie di vincoli supportati dallo strumento. Se un vincolo non appartiene a questa lista, viene scartato, evitando errori o risultati incoerenti.

Un aspetto chiave di questa implementazione è l'uso di un `set` per memorizzare i vincoli, il che garantisce l'eliminazione automatica dei duplicati. Infatti, poiché un `set` non ammette elementi ripetuti, ogni vincolo viene inserito una sola volta, evitando la ridondanza che potrebbe verificarsi nel caso in cui un vincolo venga scoperto in più sotto-log distinti.

Quest'operazione di unione permette quindi di:

- **Evita la duplicazione** dei vincoli, riducendo il rischio di risultati ridondanti.
- **Assicura che solo vincoli validi vengano considerati**, migliorando l'affidabilità del modello finale.
- **Crea una base consolidata per l'analisi successiva**, permettendoci di valutare la frequenza con cui ciascun vincolo è rispettato nel log originale.

Al termine di questa fase, disponiamo di un file unificato contenente tutti i vincoli scoperti, che rappresenta la struttura generale delle regole di comportamento nel log analizzato. Questo file verrà poi utilizzato nella prossima fase per verificare la percentuale di soddisfacimento di ogni vincolo all'interno del log originale.

### 3.3.5 Valutazione della Frequenza dei Vincoli nel Log Originale

Dopo aver ottenuto un file unificato contenente tutti i vincoli scoperti tramite NegDis, il passo finale consiste nel verificare la loro effettiva frequenza nel log originale. Questa operazione è essenziale perché ci permette di comprendere quali vincoli sono effettivamente rispettati all'interno del log e con quale frequenza.

Infatti, mentre nella fase precedente ci siamo assicurati di identificare il maggior numero possibile di vincoli, ora dobbiamo quantificare quanto siano real-

mente rappresentativi del comportamento del sistema. In particolare, vogliamo rispondere a domande come:

- Quali vincoli sono rispettati in tutte le tracce del log originale?
- Quali vincoli sono soddisfatti solo da una percentuale del log?
- Esistono vincoli che vengono violati frequentemente?

Per ottenere queste informazioni, sfruttiamo un'ulteriore funzionalità di NegDis, ovvero il comando `check`, che ci permette di esaminare un vincolo alla volta e determinare in quante tracce esso è soddisfatto. Il codice che implementa questa operazione è il seguente:

Listing 3.15: Verifica della frequenza dei vincoli nel log originale

```
ALGORITMO VerificaVincoliLog(merged_constraints_file, log_file,
    output_file)
    LOG "Verifica dei vincoli nel log originale..."

    COSTRUISCI comando per eseguire NegDis

    PROVA a eseguire il comando
        SE eseguito con successo ALLORA
            LOG "Verifica completata con successo"

            PARSE dell'output JSON
            INIZIALIZZA satisfaction_counts come dizionario vuoto

            PER ogni traccia nell'output JSON FAI
                PER ogni vincolo soddisfatto FAI
                    AGGIORNA conteggio nel dizionario
                FINE PER
            FINE PER

            SCRIVI i risultati su output_file
            LOG "Risultati salvati in output_file"
        ALTRIMENTI
            LOG "Errore durante la verifica"
        FINE SE
    FINE PROVA
FINE ALGORITMO
```

Questo codice esegue un'iterazione su tutti i vincoli unificati e utilizza NegDis per verificare in quante tracce ogni vincolo è rispettato. L'output del comando `check` viene restituito in formato JSON, che viene successivamente elaborato per calcolare il numero di tracce che rispettano ciascun vincolo.

Ogni vincolo viene quindi registrato in un file di output, insieme alla frequenza con cui esso è soddisfatto nel log originale. Questo file è particolarmente utile per interpretare l'importanza di ciascun vincolo, in quanto ci permette di distinguere tra:

- **Vincoli forti**, che sono rispettati nella quasi totalità delle tracce.
- **Vincoli deboli**, che sono soddisfatti solo in una parte del log.
- **Vincoli violati**, che raramente si verificano o che potrebbero essere il risultato di rumore nei dati.

Qeusto permette di:

- **Quantificare la rilevanza** dei vincoli, identificando quelli più significativi.
- **Individuare eventuali anomalie**, segnalando vincoli che vengono frequentemente violati.
- **Supportare la modellazione dei processi**, fornendo dati utili per costruire modelli predittivi basati su regole causali.

Questa fase rappresenta quindi un passaggio chiave per la validazione dell'intero approccio, permettendoci di confrontare le relazioni identificate con il comportamento reale del sistema e migliorare la nostra comprensione del processo analizzato.

## 3.4 Generazione del Modello Declare

Nel processo di analisi dei log di eventi, la scoperta di vincoli tra le attività è un aspetto fondamentale per la generazione di modelli descrittivi e predittivi dei processi. Tradizionalmente, l'approccio frequenzista è il più utilizzato per classificare e selezionare i vincoli, ordinandoli semplicemente in base alla loro frequenza di soddisfacimento nel log. Tuttavia, questo metodo presenta delle limitazioni in termini di robustezza logica e capacità esplicativa, in quanto non considera la relazione causale tra le attività né la struttura logica dei vincoli stessi.

Il nostro approccio mira a superare queste limitazioni introducendo una strategia di selezione dei vincoli più avanzata, basata su tre criteri principali, ordinati in termini di priorità:

- **Vincoli tra attività con relazioni causali**: i vincoli che coinvolgono attività connesse da una relazione causale vengono privilegiati rispetto agli altri.
- **Vincoli unari rispetto a quelli binari**: le regole che riguardano una singola attività (vincoli unari) vengono preferite rispetto a quelle che coinvolgono coppie di attività (vincoli binari).
- **Frequenza di soddisfacimento del vincolo**: a parità dei criteri precedenti, la selezione è guidata dalla frequenza con cui il vincolo è rispettato nel log.

Questa metodologia consente di enfatizzare i vincoli che rappresentano le relazioni causali tra le attività, eliminando vincoli ridondanti o sussunti. Ad esempio, se si considerano due modelli equivalenti:

- **Modello 1**: Existence(A), Response(A,B)
- **Modello 2**: Existence(A), Existence(B)

Se tra A e B esiste una relazione causale, il primo modello è preferito in quanto cattura meglio la dipendenza tra le attività. In assenza di una relazione causale, i due modelli rimangono equivalenti.

### 3.4.1 Estrazione della Mappatura Attività-Dati dal Log XES

Per poter identificare le relazioni causali tra le attività, è necessario estrarre la mappatura tra le attività e i dati associati presenti nel log XES. Questo viene realizzato attraverso il seguente algoritmo:

Listing 3.16: Estrazione della mappatura attività-dati dal log XES

```
ALGORITMO estrai_mappatura_xes(xes_file)
    INIZIALIZZA mappatura_dati_attivita come dizionario vuoto
    INIZIALIZZA mappatura_inversa come dizionario vuoto

    PROVA ad aprire e analizzare il file XES
        PER ogni traccia nel log FAI
            PER ogni evento nella traccia FAI
                INIZIALIZZA attivita come nulla
                INIZIALIZZA dati_associati come lista vuota

                PER ogni attributo dell'evento FAI
                    SE l'attributo    "concept:name" ALLORA
                        Salva il nome dell'attivit  in maiuscolo
                    ALTRIMENTI
                        Aggiungi la chiave alla lista
                            dati_associati
                FINE PER

                SE attivita non    nulla ALLORA
                    Aggiorna mappatura_dati_attivita con l'
                        attivit  e i suoi dati
                    Aggiorna mappatura_inversa associando ogni dato
                         all'attivit  corrispondente
                FINE SE
            FINE PER
        FINE PER

    RESTITUISCI mappatura_dati_attivita e mappatura_inversa
FINE ALGORITMO
```

Questa funzione analizza il file XES per mappare le attività presenti nel log con i dati associati a ciascuna di esse. La mappatura risultante permette di stabilire quali dati sono utilizzati da quali attività, informazione essenziale per determinare relazioni di causalità.

### 3.4.2 Identificazione delle Relazioni Causali dai Grafi DOT

Dopo aver ottenuto la mappatura tra dati e attività, il passo successivo è identificare le relazioni causali tra le attività stesse. Questa informazione viene estratta analizzando i file DOT, i quali rappresentano i grafi causali scoperti da algoritmi di causalità, come il PC Algorithm o metodi simili di causal discovery.

L'obiettivo di questa fase è trasformare il grafo scoperto, che può contenere informazioni a livello di attributi e dati, in un grafo che rappresenta le relazioni dirette tra le attività. Questo è essenziale perché molti vincoli comportamentali si basano su interazioni causali tra eventi nei processi.

Listing 3.17: Estrazione delle relazioni causali dai file DOT

```
ALGORITMO estrai_mappatura_nodi_dot(mappatura_inversa)
    CERCA tutti i file DOT nella cartella di output
```

```
    SE non ci sono file DOT ALLORA
        RESTITUISCI un set vuoto
    FINE SE

    INIZIALIZZA mappatura_nodi come dizionario vuoto
    INIZIALIZZA relazioni_causali come set vuoto

    PER ogni file DOT trovato FAI
        PROVA a leggere e analizzare il grafo DOT
            PER ogni nodo nel grafo FAI
                ESTRAI l'ID del nodo e la sua etichetta
                SALVA l'associazione ID-etichetta in mappatura_nodi
            FINE PER

            PER ogni arco nel grafo FAI
                ESTRAI il nodo sorgente e il nodo destinazione
                CONVERTI le etichette in nomi attivit  usando
                    mappatura_inversa
                AGGIUNGI la coppia (attivit _causa ,
                    attivit _effetto) a relazioni_causali
            FINE PER
    FINE PER

    RESTITUISCI relazioni_causali
FINE ALGORITMO
```

L'analisi di questi file ci permette di trasformare il grafo scoperto, che può includere relazioni tra i dati, in una struttura più interpretabile che evidenzia le relazioni dirette tra le attività. Questo passaggio è essenziale per distinguere le semplici correlazioni da reali dipendenze causali.

Per identificare queste relazioni, il primo passo è cercare tutti i file DOT disponibili nella cartella di output. Se non ne troviamo nessuno, significa che non sono state rilevate relazioni causali e la funzione restituisce semplicemente un set vuoto. Se invece i file sono presenti, procediamo alla loro analisi. Ogni file DOT rappresenta un grafo, quindi dobbiamo esaminare tutti i nodi e i collegamenti tra di essi.

I nodi contengono un identificativo relativo al dato di un attività, ricordando che se due variabili $x_i$ e $x_j$, corrispondenti ad attività distinte $a_i, a_j$, sono collegate da un arco causale nel grafo causale, allora si assume una relazione causale tra le attività corrispondenti:

$$(x_i \to x_j) \Rightarrow (a_i \to a_j)$$

possiamo ricondurre i nodi alle attività effettive, sfruttiamo la mappatura inversa tra dati e attività che abbiamo ottenuto in precedenza. Dopo aver costruito questa corrispondenza, analizziamo gli archi del grafo, che rappresentano le relazioni causa-effetto tra i nodi.

L'insieme finale delle relazioni causali contiene quindi coppie di attività, ciascuna delle quali rappresenta un collegamento causa-effetto. Questa informazione sarà utilizzata in seguito per stabilire quali vincoli abbiano maggiore rilevanza logica e quindi meritino di essere conservati rispetto ad altri meno significativi.

### 3.4.3 Lettura e Ordinamento dei Vincoli

Dopo aver estratto i vincoli dal file di input, questi vengono ordinati in base ai tre criteri sopra descritti:

1. Presenza di una relazione causale tra le attività coinvolte.
2. Tipo del vincolo (unario o binario).
3. Frequenza di soddisfacimento nel log.

Listing 3.18: Ordinamento dei vincoli in base alla priorità

```
ALGORITMO priorita_vincolo(vincolo, score, relazioni_causali)
    ESTRAI tipo e attivit  coinvolte dal vincolo

    SE il vincolo    binario E le attivit  hanno relazione causale
         ALLORA
        ASSEGNA alta priorit
    FINE SE

    SE il vincolo    unario ALLORA
        ASSEGNA priorit  media
    FINE SE

    DETERMINA la priorit  finale basandoti sulla frequenza di
        soddisfacimento

    RESTITUISCI la priorit  come tupla ordinabile
FINE ALGORITMO
```

Questa funzione assegna un punteggio di priorità ai vincoli in base alla loro importanza logica e alla loro frequenza di soddisfacimento. Il criterio principale riguarda la presenza di una relazione causale tra le attività coinvolte: se due attività sono direttamente collegate da una relazione causale, i vincoli che le coinvolgono vengono privilegiati. Successivamente, vengono preferiti i vincoli unari rispetto a quelli binari, poiché forniscono regole più generali sul comportamento delle singole attività. Infine, in caso di parità nei criteri precedenti, la frequenza con cui un vincolo è soddisfatto nel log viene utilizzata come criterio di ordinamento.

### 3.4.4 Eliminazione di Vincoli Ridondanti e Sussunti

Dopo aver estratto e ordinato i vincoli in base alla priorità, è necessario rimuovere quelli ridondanti o sussunti. Questo processo è essenziale per garantire un modello conciso ed efficace, eliminando informazioni duplicate o meno significative.

I vincoli ridondanti possono manifestarsi in diversi modi:

- **Vincoli contraddittori**
- **Vincoli sussunti da altri più generali**: ad esempio, `Existence(A)` può rendere superfluo `Existence2(A)`, poiché quest'ultimo è una specializzazione più restrittiva del primo.
- **Vincoli meno informativi rispetto a relazioni causali**: se un vincolo `CoExistence(A, B)` è già implicato da una `Response(A, B)`, il primo può essere eliminato.

Listing 3.19: Filtraggio dei vincoli ridondanti

```
ALGORITMO filtra_vincoli_ridondanti(vincoli, relazioni_causali, N)
    ORDINA i vincoli in base alla priorit
    INIZIALIZZA liste di vincoli gi  presenti (coexistence,
        precedence, etc.)

    PER ogni vincolo nell'elenco ordinato FAI
        SE il vincolo    un "NotSuccession" ALLORA
            SCARTALO
        FINE SE

        SE esiste gi  un vincolo pi  forte per la stessa coppia
            di attivit  ALLORA
            SCARTALO
        FINE SE

        SE il vincolo    sussunto da un vincolo pi  generale
            ALLORA
            SCARTALO
        FINE SE

        ALTRIMENTI, AGGIUNGILO alla lista finale
    FINE PER

    RESTITUISCI i primi N vincoli filtrati
FINE ALGORITMO
```

Questo metodo permette di ottenere un insieme più chiaro e sintetico, riducendo il numero complessivo di vincoli ma mantenendo intatta la capacità descrittiva del modello.

### 3.4.5 Scrittura dei Vincoli Finali su File

Dopo il filtraggio e l'ordinamento, i vincoli vengono salvati in un file per essere utilizzati nelle fasi successive dell'analisi.

Listing 3.20: Scrittura dei vincoli finali su file

```
ALGORITMO scrivi_vincoli(vincoli, file_path)
    APRI il file in modalit  scrittura

    PER ogni vincolo nella lista FAI
        SCRIVI il vincolo e il suo punteggio nel file
    FINE PER

    CHIUDI il file
FINE ALGORITMO
```

Questa funzione assicura che il risultato del processo di selezione e filtraggio venga memorizzato in un formato leggibile e utilizzabile. Il file risultante conterrà l'elenco dei vincoli ordinati con il loro punteggio, permettendo di riprodurre l'analisi in fasi successive senza dover ripetere il calcolo.

L'approccio proposto garantisce una selezione dei vincoli più robusta rispetto ai metodi frequenzisti tradizionali, preservando la coerenza logica del modello. L'integrazione delle relazioni causali, la preferenza dei vincoli unari e l'eliminazione di ridondanze migliorano la qualità della rappresentazione del processo estratto dal log. Grazie a questo metodo, è possibile ottenere un insieme di vincoli più significativo migliorando la sua capacità descrittiva e predittiva.

# Capitolo 4

# Generazioni di Modelli Declare attraverso la Scoperta di Relazioni Causali tra Dati e Attività

## 4.1 Introduzione

Nel capitolo precedente abbiamo illustrato un metodo per la scoperta di vincoli Declare basato sulla relazione causale tra i dati delle attività. L'idea centrale era che se due attività condividevano dati in relazione causale, allora poteva esistere una dipendenza tra di esse. Questo approccio si è rivelato efficace nel ridurre la ridondanza e nel selezionare vincoli che riflettono le vere interazioni tra le attività del processo analizzato.

Tuttavia, un processo aziendale non è composto esclusivamente da relazioni tra dati di attività diverse. In molte situazioni reali, le scelte sulle attività da eseguire dipendono dai dati stessi. Ad esempio, in un sistema di e-commerce, l'azienda può offrire metodo di spedizione in base al valore totale dell'ordine. In questi casi, la relazione causale non è più tra dati di due attività diverse, ma tra un dato e la selezione dell'attività successiva.

In questo capitolo proponiamo un approccio alternativo, sempre basato sulla scoperta di relazioni causali, ma con una differenza metodologica sostanziale:

- Non analizziamo più solo le relazioni tra dati di attività diverse, ma osserviamo come i dati influenzano la scelta dell'attività successiva.
- Questo ci permette di modellare percorsi alternativi nel processo, creando un modello Declare più descrittivo e predittivo.

Questa metodologia è particolarmente utile nei contesti in cui le scelte operative dipendono da condizioni specifiche sui dati, permettendo di costruire modelli di processo più fedeli alla realtà.

## 4.2 Un Nuovo Approccio alla Scoperta di Relazioni Causali

Nel metodo descritto nel capitolo 3, la relazione causale tra le attività veniva stabilita analizzando le relazioni tra i loro dati. Se il valore di un dato di A influenzava il valore di un dato di B, allora si assumeva che A e B fossero collegate da una relazione causale. Questo metodo era efficace per individuare vincoli che regolano il comportamento delle attività in modo indiretto, attraverso i dati che esse manipolano.

Nel metodo proposto in questo capitolo, cambiamo prospettiva:

- Invece di confrontare i dati delle attività tra loro, analizziamo l'effetto che i dati hanno sulla sequenza delle attività nel log.
- Assegniamo un valore numerico a ciascuna attività e costruiamo un dataset strutturato, trasformando il log in una rappresentazione più facilmente analizzabile.
- Infine, applichiamo le stesse tecniche di causal learning utilizzate nel capitolo precedente, per individuare eventuali relazioni causa-effetto tra i dati e le scelte delle attività.

L'idea è che, se un certo dato influenza sistematicamente la scelta dell'attività successiva, possiamo dedurre una relazione causale tra quel dato e la sequenza degli eventi.

## 4.3 Un Esempio Introduttivo

Per comprendere meglio questo concetto, immaginiamo un semplice processo con tre attività:

- A $\rightarrow$ valore numerico 1
- B $\rightarrow$ valore numerico 2
- C $\rightarrow$ valore numerico 3

Se nel log osserviamo sequenze di attività ripetute (ad esempio, A seguito da B, oppure A seguito da C), possiamo rappresentare queste informazioni numericamente. A questo punto, possiamo verificare se i dati associati ad A influenzano la scelta tra B e C.

Questa rappresentazione ci permette di trattare le attività come variabili discrete e di individuare relazioni causali attraverso l'analisi delle dipendenze nei dati.

Nella prossima sezione, esploreremo un esempio più dettagliato per capire come questo metodo possa essere applicato a un caso reale.

## 4.4 Esempio Dettagliato: ACQUISTO e SPEDIZIONE

Per comprendere meglio il funzionamento di questo approccio, analizziamo un caso pratico tratto da un contesto di e-commerce.

### 4.4.1 Scenario

Supponiamo di avere un processo in cui un cliente effettua un **ACQUISTO**, e successivamente il sistema gestisce una **SPEDIZIONE**. Ogni attività nel log ha dei dati associati:

**ACQUISTO**

- **numero_oggetti**: quantità di prodotti acquistati.
- **prezzo_totale**: importo complessivo dell'ordine.
- **timestamp**: data e ora dell'acquisto.

**SPEDIZIONE**

- **prezzo**: costo della spedizione.
- **timestamp**: data e ora della spedizione.

Nel capitolo 3, avevamo determinato una relazione causale tra *prezzo_totale* di ACQUISTO e *prezzo* di SPEDIZIONE. Infatti, per come era stato costruito il processo, la spedizione era gratuita sopra una certa soglia di prezzo dell'ordine, mentre aveva un costo inferiore a tale soglia. Questo legame veniva individuato grazie alla correlazione tra i dati di ACQUISTO e SPEDIZIONE.

Tuttavia, questo metodo considera solo la relazione tra dati appartenenti a due attività diverse, senza modellare esplicitamente l'effetto del dato sulla sequenza degli eventi.

### 4.4.2 Un Nuovo Modello Basato sulle Attività

Nel nuovo approccio, adottiamo una rappresentazione alternativa per rendere più esplicita la relazione tra *prezzo_totale* e la scelta dell'attività successiva.

Invece di trattare **SPEDIZIONE** come un'unica attività, suddividiamo il log in due possibili esiti:

- **SPEDIZIONE_GRATIS** (quando il prezzo di ACQUISTO supera la soglia).
- **SPEDIZIONE_STANDARD** (quando il prezzo di ACQUISTO è inferiore alla soglia).

Questa nuova rappresentazione ci consente di modellare direttamente la relazione tra il valore di un dato (*prezzo_totale*) e la sequenza delle attività nel processo.

**Come Rappresentiamo il log?** Creiamo una nuova colonna che rappresenta il tipo di spedizione:

- 1 se la spedizione è gratuita (**SPEDIZIONE_GRATIS**).
- 2 se la spedizione ha un costo (**SPEDIZIONE_STANDARD**).

Ora il nostro dataset assume la seguente forma:

In questa tabella, il *prezzo_totale* non è più direttamente collegato a un altro valore numerico (come il prezzo della spedizione), ma determina la scelta dell'attività successiva.

| ID Transazione | Prezzo Totale | Tipo Spedizione |
|---|---|---|
| 001 | 120€ | 1 (Gratis) |
| 002 | 80€ | 2 (Standard) |
| 003 | 150€ | 1 (Gratis) |
| 004 | 60€ | 2 (Standard) |

Table 4.1: Rappresentazione del log con il tipo di spedizione derivato dal prezzo totale

### 4.4.3 Identificazione della Relazione Causale tra Dato e Attività

Ora che abbiamo strutturato il log in modo da rappresentare direttamente le scelte operative, possiamo applicare lo stesso approccio di causal learning usato nel capitolo 3 per verificare se il *prezzo_totale* causa la selezione tra **SPEDIZIONE_GRATIS** e **SPEDIZIONE_STANDARD**.

Per farlo, seguiamo questi passi:

1. **Costruiamo il dataset con le attività codificate numericamente**
   - Ogni transizione tra **ACQUISTO** e una delle due alternative di **SPEDIZIONE** viene rappresentata numericamente.
2. **Applichiamo tecniche di causal learning per identificare relazioni causa-effetto**
   - Se troviamo una relazione causale tra *prezzo_totale* e il valore della colonna "Tipo Spedizione", significa che il prezzo dell'acquisto determina direttamente l'attività successiva.

Nel nostro caso avendo generato una relazione di causalità basata su un threshold, se questa viene confermata, possiamo concludere che il processo segue una logica decisionale basata sui dati:

- Se *prezzo_totale* $> X \Rightarrow$ l'attività successiva sarà **SPEDIZIONE_GRATIS**.
- Se *prezzo_totale* $\leq X \Rightarrow$ l'attività successiva sarà **SPEDIZIONE_STANDARD**.

A differenza del metodo del capitolo 3, qui non ci limitiamo a scoprire vincoli tra dati di attività già esistenti, ma modelliamo percorsi decisionali alternativi, in cui i dati influenzano direttamente la sequenza degli eventi.

# Capitolo 5

# Conclusione

Il contributo principale di questo lavoro è l'introduzione di un criterio causale per la selezione dei vincoli, che consente di:

- **Superare le limitazioni dei metodi tradizionali**, migliorando l'affidabilità dei vincoli individuati.
- **Distinguere correlazioni da relazioni di dipendenza**, garantendo un modello più coerente con la logica effettiva del processo.

Inoltre, abbiamo esplorato un'estensione dell'approccio, analizzando come i dati influenzano direttamente la sequenza delle attività nel log. Questo ha permesso di individuare relazioni causali non solo tra attività, ma anche tra variabili decisionali e percorsi alternativi all'interno del processo, rendendo il modello più predittivo e descrittivo.

## Limiti e Sviluppi Futuri

Sebbene i risultati ottenuti dimostrino l'efficacia dell'approccio proposto, alcune limitazioni rimangono aperte:

- **Scalabilità del metodo**: l'analisi causale su dataset reali molto grandi potrebbe generare problemi che non sono riscontrabili negli esperimenti su dataset sintetici.
- **Espandibilità del modello**: soprattutto per quanto riguarda il secondo metodo proposto, sarebbe interessante ampliare il modello in modo da apprendere la logica decisionale per la selezione del percorso tra due o più attività.

In futuro, questo approccio potrebbe essere esteso per applicazioni più ampie, come il **monitoraggio automatico dei processi aziendali** e la **diagnosi di anomalie nei log di eventi**. L'integrazione tra causalità e scoperta dei vincoli potrebbe rappresentare un nuovo paradigma nell'analisi dei processi, permettendo di costruire modelli più interpretabili e affidabili.

# Bibliography


[1] M. Pesic, H. Schonenberg and W. M. P. van der Aalst, DECLARE: Full Support for Loosely-Structured Processes. *Proocedings of the 11th IEEE International Enterprise Distributed Object Computing Conference (EDOC 2007)*, pp. 287-287, Annapolis, MD, USA, (2008) doi.org/10.1109/EDOC.2007.14

[2] A. Alman, F.M. Maggi, M. Montali, R. Peñaloza, Probabilistic declarative process mining. *Information Systems*, 109, 102033 (2022) doi.org/10.1016/j.is.2022.102033

[3] Vespa Michela; Bellodi Elena; Chesani Federico; Loreti Daniela; Mello Paola; Lamma Evelina; Ciampolini Anna; Gavanelli Marco; Zese Riccardo, Probabilistic Traces in Declarative Process Mining. *AIxIA 2024 – Advances in Artificial Intelligence. AIxIA 2024. Lecture Notes in Computer Science*, (2024) https://dx.doi.org/10.1007/978-3-031-80607-0_25

[4] Vespa Michela; Bellodi Elena; Chesani Federico; Loreti Daniela; Mello Paola; Lamma Evelina; Ciampolini Anna, Probabilistic Compliance in Declarative Process Mining. *Proceedings of the 3rd International Workshop on Process Management in the AI Era (PMAI 2024) co-located with 27th European Conference on Artificial Intelligence (ECAI 2024)*, (2024) https://ceur-ws.org/Vol-3779/paper1.pdf

[5] Mario Alviano, Antonio Ielo and Francesco Ricca;Efficient Compliance Computation in Probabilistic Declarative Specifications *Workshop Proceedings of the 40th International Conference on Logic Programming (ICLP-WS 2024) co-located with the 40th International Conference on Logic Programming (ICLP 2024), Dallas, TX, USA, October 12th and 13th, 2024*, (2024) https://ceur-ws.org/Vol-3799/paper1PLP24.pdf

[6] Fabrizio Maria Maggi, Claudio Di Ciccio, Chiara Di Francescomarino, Taavi Kala, Parallel algorithms for the automated discovery of declarative process models. *Information Systems*, (2018) doi.org/10.1016/j.is.2017.12.002

[7] Agostinelli, Simone and Chiariello, Francesco and Maggi, Fabrizio Maria and Marrella, Andrea and Patrizi, Fabio, Process mining meets model learning: Discovering deterministic finite state automata from event logs for business process analysis. *Information Systems*, (2023) doi.org/10.1016/j.is.2023.102180

[8] Y. Zheng, B. Huang, W. Chen, J. Ramsey, M. Gong, R. Cai, S. Shimizu, P. Spirtes, K. Zhang, *Causal-learn: Causal discovery in python*, Journal of Machine Learning Research, vol. 25, no. 60, pp. 1–8, 2024.

[9] Causal-learn documentation: PC and FCI algorithms. Available at:
https://causal-learn.readthedocs.io/en/latest/search_methods_index/Constraint-based%20causal%20discovery%20methods/PC.html
https://causal-learn.readthedocs.io/en/latest/search_methods_index/Constraint-based%20causal%20discovery%20methods/FCI.html