UNIVERSITÀ POLITECNICA DELLE MARCHE

FACOLTÀ DI INGEGNERIA

Corso di Laurea Triennale in Ingegneria Gestionale

Un approccio data-driven basato sulla programmazione matematica per massimizzare l'affidabilità dei sistemi di produzione

A data-driven mathematical programming based approach for maximizing the production systems reliability

Relatore:
Dott.ssa Ornella Pisacane

Tesi di laurea di:
Letizia Ciccarè
Matr. 1080301

Correlatore:
Prof. Domenico Potena

Anno Accademico 2018/2019

Abstract

Ad oggi i costi di manutenzione rappresentano una buona percentuale di quelli di tutti gli impianti produttivi. Per cui, progettare politiche predittive di manutenzione diventa molto importante al fine di aumentare l'affidabilità dei sistemi di produzione.

Obiettivo della tesi è stato quello di studiare e progettare una politica di manutenzione predittiva, basata sull'uso combinato della programmazione matematica e di tecniche di *graph mining*, per massimizzare l'affidabilità dei sistemi di produzione. In particolare, data la lista dei componenti e delle relative rotture, si costruisce un grafo in cui a ciascun nodo è associato un componente ed esiste un arco (i,j) se dopo la rottura del componente i, si è rotto anche il componente j. Quindi, un algoritmo di *graph mining* (gSpan) estrae le sequenze di rottura più frequenti con relativo supporto, partendo dal guasto di un componente specifico. La lunghezza di queste sequenze è stabilita in base al numero di eventi che si decide di prendere in considerazione e non relativamente ad una determinata finestra temporale. Le sequenze estratte verranno utilizzate come dati per un modello di programmazione lineare intera opportunamente formulato che, ricevute in input le sequenze di rottura e il relativo supporto, seleziona i componenti che è più conveniente riparare a seguito del guasto di uno specifico, in maniera da massimizzare l'affidabilità totale del sistema (ovvero, massimizzare il supporto complessivo). La selezione del modello è vincolata al budget disponibile per le riparazioni ed anche al tempo complessivo di riparazione a disposizione.

Un'analisi sperimentale, condotta su istanze opportunamente generate, mostra i benefici di una tale politica predittiva. Tale analisi è stata condotta anche con l'obiettivo di studiare la sensitività dell'approccio a variazioni dei dati di input (ad esempio, il budget a disposizione).

Indice

Capitolo 1 - Introduzione.....**3**

1.1 - Il contesto.....3

1.2 - Politiche di manutenzione.....3

1.3 - Organizzazione della tesi.....5

Capitolo 2 - Stato dell'arte.....**6**

2.1 - Data Mining nella manutenzione predittiva.....6

2.2 - Approcci di ottimizzazione matematica per politiche di manutenzione predittiva.....9

Capitolo 3 - Approccio risolutivo.....**13**

3.1 - Approccio a grafo.....13

3.2 - gSpan.....17

3.3 - Un modello di programmazione lineare intera per la selezione dei componenti da manutenere.....24

Capitolo 4 - Risultati numerici.....**30**

4.1 - Generazione delle istanze.....30

4.2 - Risultati.....36

4.3 - Analisi di stabilità: supporto, budget e tempo di manutenzione.....38

4.3.1 - Variazione della soglia di supporto minimo.....38

4.3.2 - Variazione del budget massimo B a disposizione per la manutenzione.....41

4.3.3 - Variazione del tempo massimo T a disposizione per la manutenzione.....44

Capitolo 5 - Conclusioni e sviluppi futuri.....**48**

5.1 - Conclusioni.....49

5.2 - Sviluppi futuri.....49

Bibliografia.....51

Capitolo 1 - Introduzione

1.1 Il contesto

Ad oggi i costi di manutenzione rappresentano una buona parte dei costi operativi di tutti gli impianti produttivi. Infatti, a seconda della specifica industria, questi costi possono rappresentare tra il 15% e il 60% di tutti i costi di produzione. Inoltre, indagini specifiche evidenziano che un terzo dei costi di un'azienda sono dovuti a manutenzioni superflue o effettuate in modo non appropriato [1].

Poterli ridurre rappresenta quindi una necessità per essere competitivi sul mercato. Infatti, le manutenzioni hanno influenza sulla qualità del prodotto, sui costi di produzione e, più in generale, sul profitto dell'impresa. Un guasto in una catena produttiva potrebbe comportare rallentamenti su tutta la produzione con conseguenti perdite per l'azienda stessa.

È necessario quindi fornire metodi per ridurre o eliminare le riparazioni superflue, prevenire guasti catastrofici delle macchine e ridurre l'impatto negativo delle manutenzioni sul profitto.

1.2 Politiche di manutenzione

Per capire l'importanza di applicare una politica di manutenzione predittiva è necessario comprendere le limitazioni che altre politiche presentano.

Un approccio *run-to-failure*, cioè un approccio correttivo in cui si attende la rottura di una componente prima di ripararla, comporta un eccessivo tempo di inattività della macchina e, nei casi in cui la produzione dipenda da quella specifica macchina, anche dell'impianto stesso. Questo ovviamente limita la capacità produttiva e si corre il rischio, in caso di guasti, di non riuscire a soddisfare le commissioni dei clienti. È evidente che non essendoci alcuna politica di prevenzione, l'impianto deve essere in grado di reagire in modo tempestivo a tutti i malfunzionamenti e ciò comporta costi di gestione aggiuntivi senza però risolvere il problema dei tempi di inattività. Nel caso di ridondanze

nella macchina questi problemi risultano mitigati ma la rottura di un componente potrebbe comunque comportare prestazioni ridotte da parte del macchinario [2].

In molti impianti si è quindi applicata una politica di manutenzione preventiva: sotto l'ipotesi che tutti macchinari subiscano un degrado col passare del tempo, si possono programmare delle manutenzioni in base al tempo di vita medio di un componente o di un macchinario. Il problema di questo approccio è che questi tempi variano fortemente in base alle reali condizioni d'esercizio, cioè all'uso che viene fatto del macchinario nello specifico impianto.
Ciò comporta sprechi sia di tempo che di capitale nei casi in cui la riparazione non fosse necessaria, oppure a conseguenze analoghe a quelle della politica *run-to-failure*.

Attraverso l'uso dei dati forniti dagli stessi impianti, si può invece applicare una politica di manutenzione predittiva *data-driven* in grado di garantire il massimo intervallo tra le riparazioni e minimizzare il numero e il costo di interruzioni non programmate a causa dei guasti.
In sostanza la manutenzione predittiva, attraverso l'uso di tecniche di *data mining* per analizzare i dati generati dai processi, diventa un mezzo per migliorare la produttività, la qualità del prodotto e la sicurezza negli impianti di produzione.
Questi dati, generati in grande quantità da ogni processo, possono essere raccolti durante le normali operazioni di produzione e poi analizzati per identificare *pattern* nascosti [3].
Infine, è importante che la manutenzione predittiva non venga usata solo per prevenire tempi di inattività non previsti o guasti catastrofici, ma che venga usata come strumento di ottimizzazione delle manutenzioni. Il focus dovrebbe essere eliminare tempi di inattività non necessari sia programmati che non programmati, eliminare attività di manutenzione preventiva e correttiva, estendere la vita di sistemi critici, ridurre in generale i costi per mantenere in vita il sistema [1] e quindi migliorarne l'efficienza.

1.3 Organizzazione della tesi

L'obiettivo della tesi è applicare algoritmi di *data mining* su dati storici sui guasti dei componenti combinati all'uso della programmazione matematica per massimizzare l'affidabilità dei sistemi di produzione. Questo permette di individuare le componenti che hanno più probabilità di rompersi a seguito della rottura di un altro componente, fornendo quindi indicazioni su quali è conveniente riparare, considerando vincoli sul budget complessivamente a disposizione e sul tempo di manutenzione massimo. A tal proposito, data la lista dei componenti e delle rotture, si assocerà a questa una struttura a grafo in cui ogni nodo corrisponde ad un componente e ogni arco (i,j) ad una coppia di componenti tale che se si rompe i allora si rompe anche j. Quindi, un particolare algoritmo di *graph mining* (gSpan) estrarrà tutte le sequenze di rottura, avvenuto il guasto di un componente, ed il relativo supporto. Sequenze e supporto diventeranno i dati di input di un modello di programmazione lineare binaria in cui si vuole selezionare le componenti da riparare nel rispetto del massimo budget e del massimo tempo per la riparazione a disposizione.

La tesi sarà organizzata nel seguente modo:
Nel secondo capitolo sarà presentata una disamina sulla letteratura riguardante l'uso di tecniche di *data mining*, di modelli matematici e più in generale di approcci di ottimizzazione applicati alla manutenzione predittiva.
Nel capitolo 3 sarà descritto l'approccio usato per descrivere il problema, l'algoritmo di *data mining* usato e il modello matematico su cui si basa l'ottimizzazione.
Nel quarto capitolo verrà descritto come sono state generate le istanze usate come test, verranno presentati e discussi i risultati ottenuti e verrà fatta su di loro un'analisi di stabilità.
Infine, nel capitolo 5, vengono riportate le conclusioni sul lavoro svolto e vengono evidenziati i possibili sviluppi futuri.

Capitolo 2 - Stato dell'arte

2.1 Data Mining nella manutenzione predittiva

Nei moderni sistemi manifatturieri vengono collezionati un grande numero di dati. Questi dati contengono informazioni rilevanti (ad esempio, pattern o modelli usati per descrivere regolarità tra i dati, per fare predizioni o per raggruppare dati sulla base di caratteristiche simili) che se cercate ed elaborate con strumenti adeguati, potrebbero migliorare il processo decisionale e rendere più redditizia l'intera impresa [6]. Questi *pattern* sono generalmente difficili da individuare e vengono pertanto usati algoritmi e tecniche ad hoc per farlo.
I migliori strumenti per fare ciò sono la *Knowledge Discovery in Databases* (KDD) e il *Data Mining* (DM). KDD è il processo che ci permette di identificare pattern nuovi, validi, potenzialmente utili e comprensibili nei dati [8]. Il DM è invece uno *step* nel processo di KDD, per cui si applicano specifici algoritmi per l'estrazione di *pattern*, anomalie, strutture e relazioni fra i dati. I principali approcci di DM sono quelli basati sull'intelligenza computazionale (reti neurali, sistemi a logica *fuzzy*, algoritmi genetici) e su *machine learning* (alberi decisionali e regole associative).
L'efficacia e l'efficienza di questi strumenti ha comportato negli ultimi anni un incremento della loro applicazione nel contesto del processo di produzione e più in generale su tutte le attività delle imprese, proprio perché esse generano una grande quantità di dati e da essi è necessario estrarre informazioni utili per rendere più remunerativa l'attività [9]. La conoscenza derivabile da queste informazioni può avere molte forme ed è necessario identificare il tipo di conoscenza da estrarre. Infatti, i dati possono essere riferiti al design, ai prodotti, alle macchine, ai processi, ai materiali, agli inventari, alla pianificazione ed al controllo della produzione, alla logistica, alle prestazioni del sistema e, come per il caso di studio della tesi, alla manutenzione.

Romanowski e Nagi hanno applicato un approccio di *data mining* basato sull'uso degli alberi di decisione utilizzando un *dataset* di manutenzioni programmate e

uno delle oscillazioni di un segnale. Nella loro ricerca, individuano i sottosistemi che hanno maggiore responsabilità nella scarsa disponibilità del macchinario e sulla base di ciò vengono fornite indicazioni per programmare gli intervalli tra le manutenzioni in modo più efficiente. Inoltre, i dati sulle oscillazioni sono analizzati per trovare i sensori e le risposte in frequenza che danno più informazioni riguardo il tipo di guasto [10].
Batanov et al. propongono una ricerca sui sistemi di manutenzione basati sulla conoscenza e lo sviluppo di un prototipo, chiamato EXPERT-MM, che funziona su dati storici relativi a guasti e fornisce suggerimenti per un'appropriata programmazione delle manutenzioni predittive [11].

Yuan et al. hanno usato le reti neurali e la PCA per determinare il livello di tossicità degli effluenti in un impianto chimico, rendendo così possibile, attraverso la previsione di questi livelli, lo sviluppo di strategie per la riduzione della tossicità [12].
Anche Bevilacqua et al. hanno usato le reti neurali, in particolare il percettrone multistrato (*Multilayer Perceptron* - MLP), per valutare la probabilità di guasto su 143 pompe centrifughe di una raffineria. MLP è stato usato per ponderare la correlazione esistente tra la probabilità di guasto e le diverse condizioni operative che possono avere qualche influenza nell'avvenimento [34].

Sylvain et al. hanno usato molteplici tecniche di *data mining* come gli alberi di decisione, le reti neurali, la regressione e gli insiemi grezzi (*rough set)* per predire i guasti dei componenti sulla base dei dati raccolti dai sensori di un velivolo [13].
Nello stesso ambito Skormin et al. hanno utilizzato gli alberi decisionali per una valutazione accurata della probabilità di guasto di una qualsiasi unità avionica attraverso l'uso di dati storici, scaricati da un sistema dedicato per il monitoraggio dell'*hardware* critico di volo, riguardanti l'ambiente e le condizioni operative [14].
Yam et al. hanno presentato un intelligente sistema predittivo di supporto alle decisioni (IPDSS) per le manutenzioni basate sull'usura e il deterioramento delle

macchine e dei componenti (*Condition-Based Maintenance - CBM*). L'IPDSS usa il modello delle reti neurali ricorrenti per predire i guasti di apparecchiature critiche in una centrale elettrica, permettendo così una pianificazione appropriata delle manutenzioni [15].

Huang et al. hanno presentato un sistema integrato di supporto diagnostico che usa la teoria ibrida degli insiemi grezzi *(rough set)* e un algoritmo genetico. Questo approccio è stato applicato ad una compagnia di produzione di schede madre per scoprire regole di decisione per guasti EMI (*electro-magnetic induction*) [16].
Anche Li et al. hanno sviluppato un prototipo di sistema basato sui *rough set* per far fronte all'incompletezza e all'imprecisione dei dati nella diagnostica dei guasti applicandolo però ad una pompa centrifuga di una raffineria [20].

Antomarioni et al. hanno sviluppato una politica di manutenzione *data-driven* di supporto al processo decisionale nella scelta tra una manutenzione predittiva o correttiva sulla base di misure di probabilità e l'hanno applicata ad una raffineria. Questo risultato è ottenuto attraverso l'uso di regole associative (*Association Rules* - AR) per scoprire l'esistenza di relazioni tra l'arresto dei sotto-impianti e la rottura dei componenti. Il lavoro evidenzia che le regole estratte dipendono dal tipo di arresto e dal lasso di tempo considerato e in base ad essi vengono suggerite strategie diverse per la manutenzione [17].

Antony e Nasira hanno effettuato un'analisi dei guasti a bordo dei vagoni dei treni sia attraverso l'uso del clustering esclusivo (*hard clustering*) che non-esclusivo (*soft clustering* o *fuzzy clustering*) per individuare *pattern* di rottura ricorrenti e, basandosi sui risultati ottenuti, svolgere un'analisi predittiva da integrare alla normale pianificazione delle manutenzioni [18]. Nello stesso ambito Sammouri et al. hanno usato tecniche come *K - Nearest Neighbours* (K - NN), classificatore Bayesiano, macchine a vettori di supporto e reti neurali per

individuare *pattern* significativi dai dati forniti dai sensori installati su treni commerciali con lo scopo di predire anche i guasti più rari [19].

2.2 Approcci di ottimizzazione matematica per politiche di manutenzione predittiva

Le manutenzioni non devono essere viste solo come un costo che bisogna obbligatoriamente sostenere e da cui non si può ricavare alcun beneficio aggiuntivo, oltre che la possibilità di poter continuare ad usare il macchinario riparato [21]. Al-sultan e Duffuaa hanno infatti suggerito che una programmazione appropriata delle manutenzioni gioca un ruolo fondamentale per massimizzare non solo l'affidabilità di un sistema ma anche i benefici ricavabili dalle manutenzioni, come ad esempio un aumento di sicurezza o un miglioramento della qualità del prodotto e quindi, intrinsecamente, portando valore aggiuntivo all'impresa [22].

Per raggiungere l'obiettivo di programmare al meglio le manutenzioni, bisogna tenere in considerazione diversi fattori, spesso trascurati, come eventuali obiettivi riguardo la preservazione dell'ambiente e della salute, la sicurezza, i costi di manutenzione e i costi opportunità dovuti alla mancata produzione [23].

Un classico approccio in questo contesto è inizialmente stato quello di guardare più alla minimizzazione dei costi di manutenzione, ignorando l'affidabilità del sistema. Ovviamente la minimizzazione dei costi non comporta la massimizzazione dell'affidabilità e quindi questo approccio portava a conclusioni non attuabili e non affidabili in pratica. D'altro canto, se si considera solamente l'affidabilità del sistema si rischia di incorrere in costi elevatissimi. È quindi necessario, per ottenere prestazioni migliori dall'impianto, considerare simultaneamente sia il costo di manutenzione che le misure di affidabilità. Questo è possibile se consideriamo sistemi di manutenzione che minimizzino i costi assicurando però la soddisfazione dei requisiti di affidabilità, oppure considerando sistemi che massimizzino l'affidabilità tenendo conto dei requisiti riguardanti i costi di manutenzione.

Una manutenzione ben programmata dovrebbe comunque essere in grado di minimizzare i costi a lungo termine e i tempi di inattività, massimizzando però il profitto ricavabile da un prodotto (lo stato di salute del macchinario può influire sulla qualità del prodotto e sulla percentuale di scarti), l'efficienza dell'impianto e la sicurezza sul lavoro.
I modelli di ottimizzazione delle manutenzioni possono essere sia qualitativi, *come la Total Productive Maintenance* (TPM) o la *Reliability Centered Maintenance* (RCM), che quantitativi, come i modelli Bayesiani [7].
Tra questi, l'AHP (*Analytic Hierarchy Process*) è una tecnica matematica per il processo decisionale che permette sia considerazioni qualitative che quantitative, riducendo le decisioni complesse in una serie di paragoni uno a uno, sintetizzando poi il risultato. Questa tecnica è stata usata da Bevilacqua e Braglia per selezionare la migliore strategia di manutenzione per una raffineria [24].

L'approccio basato sulla definizione di un modello analitico (AM) usa la soluzione alle equazioni usate per descrivere i cambiamenti di un sistema. Jin et al. a tal proposito hanno sviluppato un modello analitico basato sui costi per la programmazione della produzione congiunta e della manutenzione predittiva [25], mentre Oke ne ha presentato uno per l'ottimizzazione del profitto [26].

Un approccio molto usato per la risoluzione di problemi decisionali è quello di usare la linguistica *fuzzy*. Un esempio di applicazione di questa metodologia per la scelta della strategia di manutenzione viene suggerito da Mechefske e Wang. Essi Hanno associato un insieme *fuzzy* ad ogni possibile obiettivo che l'impresa può porsi e la funzione caratteristica di questi insiemi descrive l'importanza, soggettiva, di ciascuno obiettivo. Altri insiemi fuzzy sono stati associati ad ogni strategia presa in considerazione e le loro funzioni caratteristiche descrivono la capacità di ogni strategia di soddisfare i diversi obiettivi. A partire da valutazioni soggettive si riesce quindi a individuare la migliore strategia da attuare per le

manutenzioni per raggiungere anche obiettivi difficilmente valutabili attraverso l'uso di tecniche puramente quantitative [28].

Galbraith information processing model (GIPM) descrive i fattori, sia interni che esterni, che contribuiscono all'incertezza di un'organizzazione e ai meccanismi per fronteggiare questa incertezza. Swanson definisce un GIPM per studiare come strategie differenti di manutenzione possono essere applicate per far fronte alla complessità ambientale [29].

Gli algoritmi genetici vengono generalmente implementati per trovare soluzioni eventualmente anche euristiche a problemi di ottimizzazione. Ad esempio, Marseguerra *et al.* hanno considerato sistemi multicomponente continuamente monitorati e hanno progettato un algoritmo genetico per determinare il livello ottimale di degrado oltre il quale la manutenzione preventiva può essere effettuata. Gli autori hanno usato una funzione multi-obiettivo (per il profitto e la disponibilità), ottimizzando attraverso l'uso della simulazione Monte Carlo [30].

Antomarioni et al. dopo aver individuato, attraverso l'uso di regole associative, le relazioni tra le rotture dei componenti in una raffineria, hanno formulato un modello di programmazione lineare intera in grado di selezionare i componenti da riparare, a partire dalla rottura di un componente, in modo tale da migliorare la robustezza complessiva dell'impianto alle rotture, nel rispetto del budget fornito e del tempo massimo a disposizione per le riparazioni [4].

Possono essere usati anche modelli di programmazione matematica mista intera non lineare, ovvero modelli che presentano variabili decisionali che assumono solo valori interi in cui qualche vincolo o la funzione obiettivo sono non sono lineari. Matsuoka and Muraki hanno infatti formulato un modello di questo tipo per risolvere il problema del sequenziamento delle attività di manutenzione [31].

Kianfar considera la programmazione simultanea della produzione e delle attività di manutenzione, assumendo che la probabilità di guasto sia funzione dell'età del componente. L'obiettivo principale dell'autore è massimizzare il profitto attraverso l'uso dell'equazione di Riccati [32].

Cadini et al. hanno proposto un modello basato sul metodo Monte Carlo per stimare la probabilità di guasto di un componente soggetto a degrado. Queste stime, ottenute attraverso delle simulazioni, sono state arricchite con l'uso di una politica basata sulle condizioni al contorno per la sostituzione dei componenti [33].

Nahas et al. hanno analizzato la sequenza ottimale di azioni di manutenzione preventiva che minimizza i costi e soddisfa i requisiti di affidabilità per il sistema. Gli autori hanno applicato a sistemi multistato (*Multistate System* - MSS), cioè sistemi che possono assumere un range di valori per il livello di prestazione che va da perfettamente funzionante a guasto completo, un metodo di ottimizzazione basato sull'algoritmo *extended great deluge*, cioè un algoritmo metaeuristico basato sulla ricerca locale, per trovare la soluzione che minimizza i costi rispettando i requisiti di affidabilità del sistema che viene misurata come la capacità di far fronte ad una determinata richiesta [35].

Capitolo 3 - Approccio risolutivo

In questo capitolo, si descriverà in dettaglio l'approccio *data-driven* usato per massimizzare l'affidabilità dei sistemi di produzione.

In primo luogo, i dati storici sulla rottura dei componenti, creati artificialmente, sono stati mappati su grafo. Su questo grafo, a partire dalla rottura di un componente a scelta, si individuano i sotto-grafi che rappresentano le sequenze di rotture successive a quella del componente scelto. Questa operazione viene eseguita prendendo le rotture successive a quelle del componente scelto fino ad una nuova rottura del componente da cui si era partiti o fino al raggiungimento del numero massimo di rotture, anch'esso a scelta, successive a quella del componente di partenza.

È poi possibile estrarre le sequenze più frequenti e la relativa frequenza attraverso gSpan, un algoritmo di *graph mining*.

Partendo da questi dati e inserendo informazioni sul budget e sul tempo massimo disponibile, sui costi e sui tempi di riparazione dei componenti sarà possibile utilizzare un modello di programmazione lineare intera per la scelta delle sequenze da riparare in modo da massimizzare l'affidabilità del sistema relativa alle manutenzioni.

3.1 Approccio a grafo

Data la lista dei componenti e delle rotture, si può costruire un grafo per descrivere questo *dataset*.

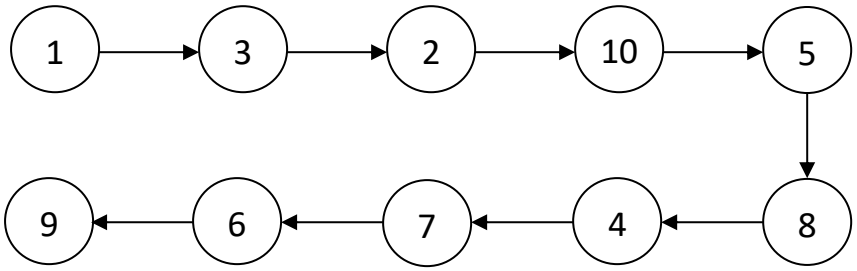
In particolare, è possibile descrivere l'evento della rottura con un nodo la cui etichetta rappresenta l'informazione sul componente soggetto a rottura. La sequenzialità tra i guasti è descrivibile tramite la creazione di un arco tra le rotture dei due componenti, formando quindi una coppia di nodi adiacenti.

Se avessimo, ad esempio, dati storici contenenti dieci eventi di rottura per 10 componenti diversi come nell'Esempio 1, sarà possibile rappresentare il dataset con un grafo di 10 nodi e l'etichetta del primo nodo sarà l'identificativo del primo

componente della sequenza (il componente 1 nell'Esempio 1), l'etichetta del secondo nodo sarà l'identificativo del componente che si è rotto successivamente al primo (il componente 3 nel caso dell'Esempio 1) e così via. La sequenzialità delle rotture, ovvero che la seconda rottura avviene in un istante di tempo successivo a quello della prima rottura viene descritta attraverso la creazione di un arco che ha come coda il nodo che rappresenta la rottura del componente temporalmente antecedente a quella che è rappresentata dal nodo di testa. Nell'Esempio 1 la rottura del componente 2 avviene in un istante di tempo successivo alla rottura del componente 3; il nodo di testa sarà quindi quello che rappresenta la rottura del componente 2 ed il nodo di coda sarà quello che rappresenta la rottura del componente 3. Il risultato di questo approccio descrittivo applicato all'esempio 1 può essere visto nella Figura 1.

Esempio 1: $1 \rightarrow 3 \rightarrow 2 \rightarrow 10 \rightarrow 5 \rightarrow 8 \rightarrow 4 \rightarrow 7 \rightarrow 6 \rightarrow 9$

Figura 1: Digrafo relativo all'Esempio 1



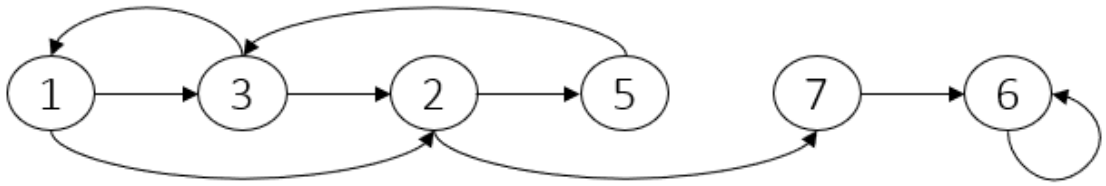
Se nella sequenza di rotture lo stesso componente compare più volte è possibile mappare il *dataset* attraverso i grafi in due modi.

Un primo metodo consiste nell'associare ai componenti soggetti a rotture un solo nodo. Avremo quindi, come nell'Esempio 2, che anche quando il componente si rompe più volte nella sequenza considerata, il componente sarà l'etichetta di un solo nodo, come si può vedere nella Figura 2. Nell'esempio, dopo la rottura del componente 1 e del componente 3, avviene una nuova

rottura del componente 1; a questa rottura non verrà associata un nuovo nodo ma verrà creato un arco che ha come nodo di coda il nodo che rappresenta la rottura del componente 3 e come nodo di testa il nodo, già creato in precedenza, che descrive la rottura del componente 1. La rottura in sequenza dello stesso componente rappresenta un caso particolare in quanto comporta la creazione di un autoanello. In questo caso il nodo di coda ed il nodo di testa coincidono. Nell'Esempio 2 l'ulteriore rottura del componente 6 dopo una prima rottura dello stesso viene rappresentata in Figura 2 con un autoanello che parte dal nodo con etichetta 6 e arriva al nodo con etichetta 6.

Esempio 2: $1 \rightarrow 3 \rightarrow 1 \rightarrow 2 \rightarrow 5 \rightarrow 3 \rightarrow 2 \rightarrow 7 \rightarrow 6 \rightarrow 6$

Figura 2: Digrafo relativo all'Esempio 2 - Un solo nodo per ogni componente

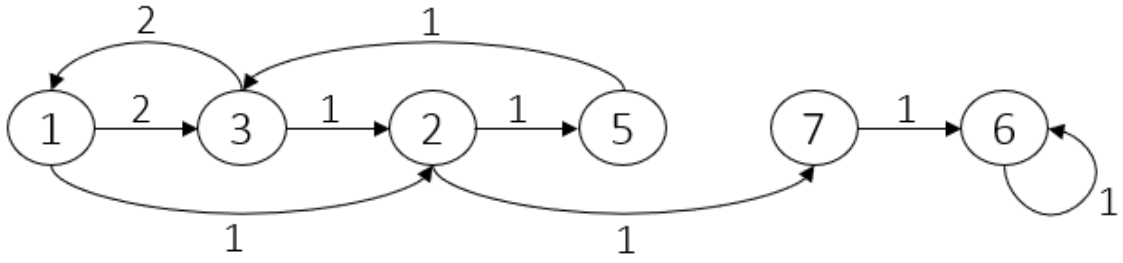


Con questo approccio è possibile rappresentare il *dataset* in modo compatto ma si corre il rischio di perdere informazioni preziose sulla sequenzialità effettiva delle rotture e sulla frequenza di rottura di un determinato componente. Per ovviare a quest'ultimo problema si può associare ad ogni arco che parte da un nodo i e arriva ad un nodo j un peso per indicare la frequenza con cui dopo la rottura del componente i è avvenuta la rottura del componente j. Ad esempio, se nella sequenza la rottura del componente 1 dopo quella del componente 3 avviene due volte, come negli Esempi 3 e 4, avrò un arco di peso 2 che parte dal nodo con etichetta 3 e arriva al nodo con etichetta 1. È comunque possibile la perdita di informazioni sulla sequenza di partenza. Infatti, il grafo in Figura 3 può descrivere sia la sequenza dell'Esempio 3 che la sequenza dell'Esempio 4.

Esempio 3: $1 \rightarrow 3 \rightarrow 1 \rightarrow 3 \rightarrow 1 \rightarrow 2 \rightarrow 5 \rightarrow 3 \rightarrow 2 \rightarrow 7 \rightarrow 6 \rightarrow 6$

Esempio 4: $1 \rightarrow 2 \rightarrow 5 \rightarrow 3 \rightarrow 1 \rightarrow 3 \rightarrow 1 \rightarrow 3 \rightarrow 2 \rightarrow 7 \rightarrow 6 \rightarrow 6$

Figura 3: Digrafo pesato relativo agli Esempi 3 e 4 - Un solo nodo per ogni componente



Il secondo metodo per la mappatura su grafo dei dati storici sulla rottura dei componenti è quello che è stato usato per lo sviluppo di questa tesi. Esso consiste nell'associare ad ogni rottura un nuovo nodo, anche se a rompersi è un componente a cui era già stato associato precedentemente un altro nodo. In questo caso, come nell'Esempio 1 in cui però tutte le rotture interessavano componenti diversi, avremo tanti nodi quanti sono gli eventi registrati nel dataset.

Gli Esempi 3 e 4 rappresentano una sequenza di 12 rotture ed alcune di queste rotture interessano uno stesso componente. La mappatura su grafo di questi due esempi sarà rappresentata quindi da due grafi ognuno composto da 12 nodi, le cui etichette non sono uniche. Mentre con il primo metodo gli Esempi 3 e 4 potevano essere mappati con un solo grafo, con questo metodo la differenza tra le sequenze è mantenuta anche dopo la rappresentazione. Si ha, infatti, che all'Esempio 3 corrisponde il grafo in Figura 4, mentre all'Esempio 4 corrisponde il grafo in Figura 5.

Figura 4: Digrafo relativo all'Esempio 3 - Un nodo per ogni rottura

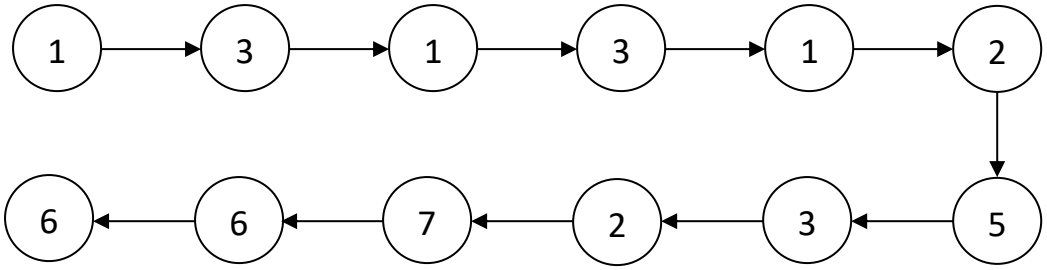
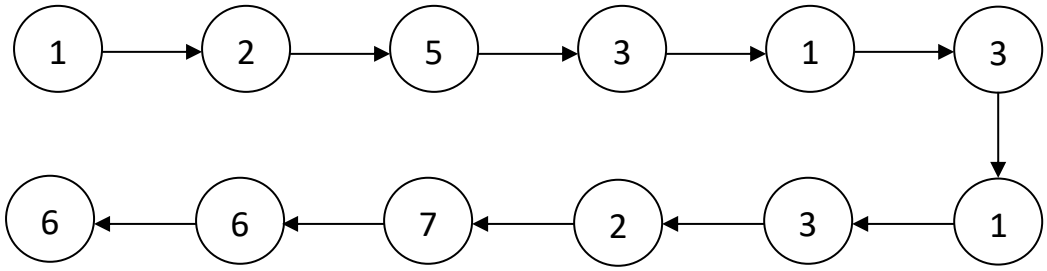


Figura 5: Digrafo relativo all'Esempio 4 - Un nodo per ogni rottura

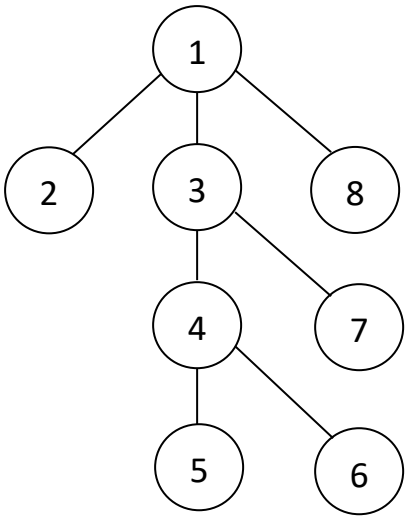


3.2 gSpan

gSpan [5] è un algoritmo di *data mining* che viene usato per l'estrazione di grafi e sotto-grafi frequenti, è cioè un algoritmo di *graph mining*.
L'algoritmo usa un nuovo ordine lessicografico tra i grafi e associa ogni grafo ad un codice DFS minimo che viene usato come etichetta canonica per identificare il grafo.
gSpan adotta un algoritmo di ricerca *Depth-First* (DFS), cioè l'esplorazione dei grafi e, successivamente, anche lo sviluppo dell'albero dei codici DFS avviene esplorando il grafo prima in profondità, cioè visitando ogni nodo di un ramo prima di tornare indietro ed esplorare l'ultimo ramo scoperto e non ancora esplorato.

Nella Figura 6 viene rappresentato un grafo esplicativo per il DFS dove l'ordine di esplorazione del grafo da parte dell'algoritmo viene evidenziato dalle etichette dei nodi.

Figura 6: Grafo esplicativo del DFS



Attraverso l'uso del DFS si esplorano i grafi e si numerano i vertici in base all'ordine di esplorazione, partendo da 0, come in Figura 7.
È possibile poi costruire un codice che viene chiamato codice DFS per ogni grafo. Ogni coppia di nodi adiacenti avrà un proprio codice che viene costruito associando il numero del vertice di partenza secondo l'ordine di scoperta al primo elemento del codice e il numero del vertice di arrivo al secondo elemento, l'etichetta del vertice di partenza al terzo elemento, l'etichetta del lato incidente ai due nodi al quarto elemento ed infine l'etichetta del vertice di arrivo al quinto elemento. Ad esempio se si vuole scrivere il codice per il grafo (a) della Figura 7 dei nodi adiacenti Y e X collegati dal lato b avremo che il primo elemento deve essere il numero del vertice secondo l'ordine di esplorazione del grafo, quindi 1, il secondo elemento il numero del secondo vertice, quindi 2, il terzo elemento l'etichetta del primo vertice quindi Y, il terzo elemento l'etichetta del lato incidente ai due quindi b e l'ultimo elemento l'etichetta del secondo vertice quindi X; in conclusione il codice sarà (1, 2, Y, b, X) come riportato anche nella Tabella 1.

Si ha però che uno stesso grafo può essere rappresentato in modi diversi come ad esempio nella Figura 7 il grafo (a) può anche essere rappresentato con la

forma dei grafi (c), e (d). Di conseguenza, anche i codici DFS di questi grafi sono diversi, seppur siano isomorfismi dello stesso grafo.
È evidente che ad uno stesso grafo può essere quindi associato più di un codice DFS.

Figura 7: Grafi isomorfi con vertici numerati secondo DFS [5]

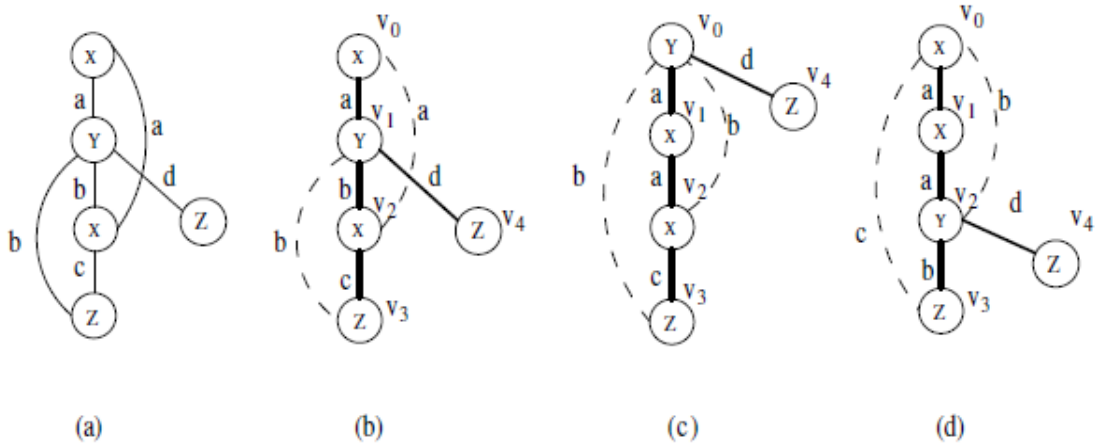


Tabella 1 - Codici DFS relativi ai grafi in Figura 7 [5]

	(a), (b)	**(c)**	**(d)**
1	(0, 1, X, a, Y)	(0, 1, Y, a, X)	(0, 1, X, a, X)
2	(1, 2, Y, b, X)	(1, 2, X, a, X)	(1, 2, X, a, Y)
3	(2, 0, X, a, X)	(2, 0, X, b, Y)	(2, 0, Y, b, X)
4	(2, 3, X, c, Z)	(2, 3, X, c, Z)	(2, 3, Y, b, Z)
5	(3, 1, Z, b, Y)	(3, 0, Z, b, Y)	(3, 0, Z, c, X)
6	(1, 4, Y, d, Z)	(0, 4, Y, d, Z)	(2, 4, Y, d, Z)

Per fare in modo che ad ogni grafo venga associato uno ed un solo codice DFS come etichetta canonica ed evitare l'effettuazione di test di isomorfismo che

risulterebbero computazionalmente molto costosi Yan e Han introducono un nuovo ordine lessicografico DFS. Se prendiamo due codici DFS a_t e b_t, individuati secondo le regole già descritte in questa sezione e formalmente descritti come segue:

$$a_t = (i_a, j_a, l_{ia}, l_{ia,ja}, l_{ja}) \qquad e \qquad b_t = (i_b, j_b, l_{ib}, l_{ib,jb}, l_{jb})$$

e se consideriamo l'insieme E_f l'insieme dei lati *forward* del grafo, cioè dei lati che collegano un vertice ad uno scoperto successivamente (ad esempio nel grafo (c) della Figura 7 il lato con etichetta a che collega il vertice 0 con etichetta Y al vertice 1 con etichetta X), e l'insieme E_b l'insieme dei lati *backward* del grafo, cioè dei lati che collegano un vertice ad uno scoperto precedentemente (ad esempio nel grafo (c) il lato con etichetta b che collega il vertice 3 con etichetta Z al vertice 0 con etichetta Y), si possono seguire delle semplici regole per individuare i codici minimi:

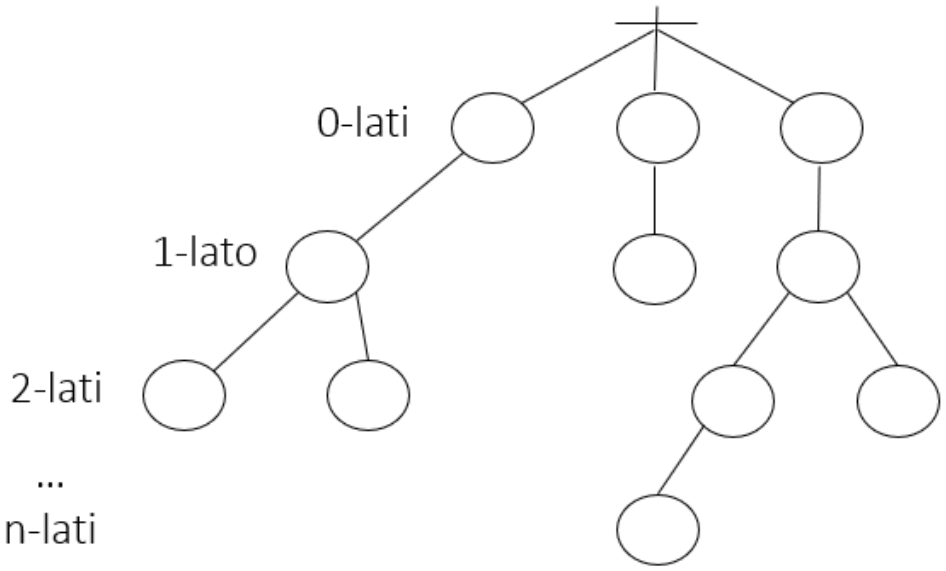
$$a_t < b_t = \begin{cases} vero\ se\ a_t \in E_b\ e\ b_t \in E_f \\ vero\ se\ a_t \in E_b\ e\ b_t \in E_b\ e\ j_a < j_b \\ vero\ se\ a_t \in E_b\ e\ b_t \in E_b\ e\ j_a = j_b\ e\ l_{ia,ja} < l_{ib,jb} \\ vero\ se\ a_t \in E_f\ e\ b_t \in E_f\ e\ i_b < i_a \\ vero\ se\ a_t \in E_f\ e\ b_t \in E_f\ e\ i_b = i_a\ e\ l_{ia} < l_{ib} \\ vero\ se\ a_t \in E_f\ e\ b_t \in E_f\ e\ i_b = i_a\ e\ l_{ia} = l_{ib}\ e\ l_{ia,ja} < l_{ib,jb} \\ vero\ se\ a_t \in E_f\ e\ b_t \in E_f\ e\ i_b = i_a\ e\ l_{ia} = l_{ib}\ e\ l_{ia,ja} = l_{ib,jb}\ e\ l_{ja} < l_{jb} \end{cases}$$

Ad esempio, preso a_t = (2, 0, Z, b, Y) e b_t = (2, 0, Z, c, X), i lati rappresentati dai due codici sono entrambi backward e quindi la regola da usare per il confronto sarà la seconda o la terza. Si valuta prima se $j_a < j_b$, nel nostro caso j_a=0 e j_b=0 quindi j_a=j_b. Va quindi usata la terza regola. valutiamo se $l_{ia,ja}$<$l_{ib,jb}$, nel nostro caso $l_{ia,ja}$=b e $l_{ib,jb}$=c quindi la condizione è verificata e possiamo concludere che $a_t < b_t$.

I codici DFS minimi, individuati attraverso l'applicazione di queste regole, saranno usati per rappresentare i grafi, come un'etichetta canonica, e per costruire l'albero dei codici DFS più frequenti (e quindi dei grafi e sotto-grafi più

frequenti). La costruzione di questo albero (Figura 8) avviene considerando prima tutti i nodi frequenti e poi aggiungendo di volta in volta ulteriori lati e nodi.

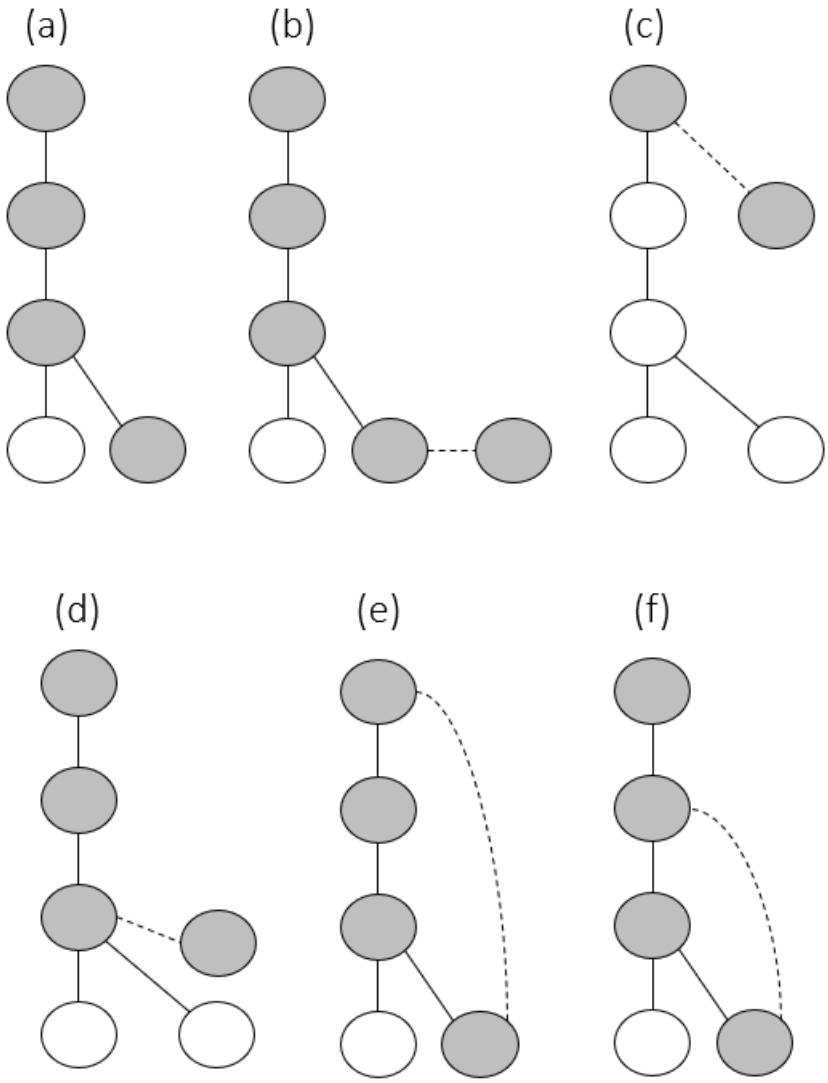
Figura 8: Struttura di un albero di codici DFS [5]



Anche la crescita dei lati è soggetta a regole ben definite. In primis è necessario individuare il cammino che partendo dalla radice del grafo arriva all'ultimo nodo scoperto durante l'esplorazione DFS del grafo in considerazione. Nella Figura 9 il cammino è quello composto dai nodi con riempimento grigio e dai lati che li collegano. Le regole impongono che un lato *forward* possa crescere ovunque da questo cammino, come si può vedere dai grafi 9(b), 9(c) e 9(d). Ovviamente l'aggiunta di un nuovo lato e di un nuovo vertice, che nella Figura 9 viene indicata con lati tratteggiati, implica che per effettuare una crescita successiva sia necessario rivalutare il cammino in quanto l'ultimo vertice scoperto sarà diventato quello appena aggiunto.

Inoltre, le regole impongono che un lato *backward* possa crescere solamente dall'ultimo vertice del cammino, come nei grafi 9(e) e 9(f).

Figura 9: Crescita dei lati da un grafo [5]



gSpan utilizza tutti gli aspetti visto fino ad ora e li unisce in un algoritmo implementativo che può essere descritto attraverso lo pseudo codice seguente [5]:

1: $Ordinamento\ dei\ vertici\ e\ dei\ lati\ nel\ GS\ in\ base\ alla\ loro\ frequenza$;
2: $Rimozione\ dei\ vertici\ e\ dei\ lati\ non\ frequenti$;
3: $Rietichettamento\ dei\ vertici\ e\ dei\ lati\ rimanenti\ in\ frequenza\ discendente$
4: $S^1 \leftarrow tutti\ i\ grafi\ frequnti\ ad\ 1\ lato$;
5: $Riordinamento\ di\ S^1\ in\ base\ all'ordine\ lessicografico\ DFS$;
6: $S \leftarrow S^1$;

7: $Per\ ogni\ grafo\ \ s \in S^1$:
8: $Subgraph\ mining\ (s, S, GS)$;
9: $GS \leftarrow GS - s$;

Si ordinano cioè tutte le etichette dei vertici e dei lati nell'insieme dei grafi *(GS)* in base alla loro frequenza e si rimuovono quelli infrequenti. Avremo infatti che se un nodo o un lato non è frequente, nemmeno il grafo che lo contiene può esserlo. Successivamente si etichettano i vertici ed i lati rimanenti secondo una frequenza discendente e si assegna all'insieme dei grafi frequenti ad 1 lato tutti i grafi del *dataset* con queste caratteristiche. Questo insieme viene ordinato secondo l'ordine lessicografico DFS e incluso nell'insieme di tutti i grafi frequenti (S). Per ogni grafo ad un lato presente nell'insieme si attiva la sotto-procedura di *subgraph mining*, al termine della quale esso viene eliminato dall'insieme dei grafi, in quanto appena valutato.

La procedura di *subgraph mining* [5] controlla che il codice DFS sia minimo, se così non fosse il grafo deve essere già stato scoperto in precedenza e non viene quindi introdotto nuovamente nell'insieme dei grafi frequenti. Se lo è si procede aggiungendolo e generando tutti i possibili figli attraverso la crescita di un lato, con le regole già viste in questa sezione. Si attiva quindi la sotto-procedura di enumerazione al termine della quale si valuta per ogni figlio se il supporto sia maggiore o uguale a quello minimo (che viene scelto dall'utente). Se lo è, il nuovo grafo ottenuto con l'aggiunta di un lato, diventa il grafo per cui viene riattivata la procedura di *subgraph mining* ciclicamente fino a quando tutto il ramo dell'albero non è stato sviluppato.

$Sottoprocedura\ di\ Subgraph\ Mining$:
1: $se\ s \neq \min(s)$:
2: termina Subgraph Mining;
3: $altrimenti$:
4: $S \leftarrow S \cup s$;

5: $generazione\ di\ tutti\ i\ potenziali\ figli\ di\ s\ aggiungendo\ un\ lato$;
6: $Enumerazione\ (s)$;
7: $per\ ogni\ figlio\ c\ di\ s$:
8: $if\ \sup(c) \geq minsup$:
9: $s \leftarrow c$;
10: $Subgraph\ Mining$;

La sotto-procedura di enumerazione [5] permette sia di contare le occorrenze di un grafo sia di prender nota dell'insieme dei grafi in cui cercare, per ogni figlio del grafo di partenza, nell'iterazione successiva del *subgraph mining*.

$Sottoprocedura\ di\ Enumerazione$:
1: $per\ ogni\ grafo\ g \in s.GS$:
2: $elenca\ la\ prossima\ occorrenza\ di\ s\ in\ g$;
3: $per\ ogni\ figlio\ c\ di\ s\ che\ compare\ in\ g$:
4: $c.GS \leftarrow c.GS \cup g$;

3.3 Un modello di programmazione lineare intera per la selezione dei componenti da manutenere

In seguito all'estrazione dei grafi e dei sotto-grafi più frequenti con l'uso dell'algoritmo appena descritto si hanno a disposizione le sequenze di rottura più frequenti dei componenti e un valore preciso per la loro frequenza, espressa come supporto.

È possibile costruire, a partire dalle sequenze e dai componenti, una matrice binaria che risulterà fondamentale nella formulazione di un modello di programmazione lineare intera per la massimizzazione dell'affidabilità del sistema. La matrice, che verrà chiamata M, avrà tante righe quante sono le sequenze e tante colonne quanti sono i componenti. Un qualsiasi elemento della matrice, a cui faremo riferimento come $m_{i,j}$, assumerà il valore 1 se il

componente j appartiene alla sequenza i, assumerà altrimenti valore 0 se non appartiene alla sequenza.

Se, ad esempio, il risultato dell'estrazione fossero tre sequenze e il numero dei componenti fosse otto avremmo una matrice 3x8, cioè tre righe per le tre sequenze e otto colonne per gli otto componenti, proprio come nell'Esempio 5. Prendendo in considerazione un qualsiasi componente j, ad esempio j=4, e valutando la presenza di questo componente in una qualsiasi sequenza i, ad esempio i=2 si può stabilire che l'elemento di posizione i,j = 2,4, cioè $m_{2,4}$, assume valore 0 in quanto nella sequenza 2 non è presente il componente 4. Prendendo in esame lo stesso componente e scegliendo invece la sequenza i=1 possiamo stabilire che l'elemento $m_{1,4}$ assume valore 1 in quanto il componente 4 è presente nella sequenza 1.

Esempio 5:

$sequenza\ 1$: $3 \rightarrow 4 \rightarrow 5$

$sequenza\ 2$: $1 \rightarrow 2$

$sequenza\ 3$: $8 \rightarrow 6 \rightarrow 7$

Tabella 2: Matrice binaria relativa all'Esempio 5

	j=1	j=2	j=3	j=4	j=5	j=6	j=7	j=8
i=1	0	0	1	1	1	0	0	0
i=2	1	1	0	0	0	0	0	0
i=3	0	0	0	0	0	1	1	1

Anche quando un componente è presente più volte in una stessa sequenza le regole per la costruzione della matrice non cambiano. La matrice resta infatti binaria ed un qualsiasi elemento $m_{i,j}$ può assumere solo il valori 1, se il componente j è presente almeno una volta nella sequenza i, o 0 altrimenti.

Nell'Esempio 6 abbiamo 3 sequenze e 5 componenti, costruiremo quindi una matrice 3x5.
Si può notare come nella sequenza 2 il componente 4 sia ripetuto due volte e come nella sequenza 3 il componente 5 sia ripetuto tre volte. In entrambi i casi l'elemento $m_{i,j}$ ($m_{2,4}$ nel primo caso e $m_{3,5}$ nel secondo) che descrive la presenza del componente j nella sequenza i assume valore 1.

Esempio 6:
$sequenza\ 1$: $3 \rightarrow 4 \rightarrow 5$
$sequenza\ 2$: $4 \rightarrow 2 \rightarrow 1 \rightarrow 4$
$sequenza\ 3$: $5 \rightarrow 5 \rightarrow 5$

Tabella 3: Matrice binaria relativa all'Esempio 6

	j=1	j=2	j=3	j=4	j=5
i=1	0	0	1	1	1
i=2	1	1	0	1	0
i=3	0	0	0	0	1

Il modello di PLI che sarà descritto in seguito riceve in input oltre che le sequenze più frequenti ed il loro supporto, i componenti, la matrice binaria costruita come appena descritto, anche i dati riguardanti il budget totale e il tempo massimo totale a disposizione per la manutenzione e quelli relativi al costo e al tempo di riparazione di ogni singolo componente.

Si possono quindi riassumere i dati di input per il modello e la notazione che verrà usata per fare riferimento ad essi nel modello stesso come segue:

- le sequenze (i=1,...,n)
- il supporto delle sequenze (sup_i)

- i componenti (j=1,...,k)
- il costo dei componenti (c_j)
- il tempo di riparazione dei componenti (t_j)
- il budget massimo totale messo a disposizione per la manutenzione (B)
- il tempo massimo totale messo a disposizione per la manutenzione (T)

Per lo sviluppo del modello viene introdotta la variabile decisionale binaria x_i, riferita alle sequenze. Questa variabile assume valore 1 se è consigliabile, per la massimizzazione dell'affidabilità del sistema, riparare la sequenza i-esima, 0 altrimenti.

$$x_i = \begin{cases} 1\ se\ riparo\ la\ sequenza\ i \\ 0\ altrimenti \end{cases}$$

Viene inoltre introdotta una seconda variabile decisionale binaria y_j, riferita ai componenti. Questa variabile assume valore 1 se il componente j-esimo afferisce almeno ad una sequenza riparata, 0 altrimenti. L'introduzione di questa seconda variabile è necessaria per poter imporre il rispetto del budget e del tempo massimo di inattività direttamente ai componenti, in modo tale da considerare il costo e il tempo di riparazione di un componente una sola volta, anche quando questo componente è presente in più sequenze riparate.

$$y_j = \begin{cases} 1\ se\ afferisce\ ad\ 1\ sequenza\ riparata \\ 0\ altrimenti \end{cases}$$

È possibile la formulazione di un altro modello di PLI senza l'introduzione di questa seconda variabile, ma questo comporterebbe la necessità di introduzione di un vincolo che imponga la presenza di ogni componente al più in una sequenza riparata per evitare di conteggiare più volte il costo o il tempo di riparazione di uno stesso componente. Questo, ovviamente, rappresenta un limite ed è consigliabile adottare il modello descritto in modo approfondito di seguito:

$$affidabilità = \max \sum_{i}^{n} x_i \, sup_i$$

$$\sum_{j}^{k} t_j \, y_j \leq T \qquad (1)$$

$$\sum_{j}^{k} c_j \, y_j \leq B \qquad (2)$$

$$\sum_{i}^{n} x_i \, m_{i,j} \leq \; y_j \sum_{i}^{n} m_{i,j} \quad \forall j = 1, \dots, k \qquad (3)$$

$$x_i, \, y_j \in \{0, \, 1\} \qquad (4)$$

La funzione obiettivo, da massimizzare, rappresenta l'affidabilità del sistema in funzione del supporto delle sequenze riparate. Il vincolo (1) impone il rispetto del tempo totale destinabile alla riparazione dei componenti mentre il vincolo (2) impone il rispetto del budget massimo totale a disposizione per la manutenzione. Il vincolo (3) esprime il legame logico tra le variabili decisionali x e y e ci consente di poter contare una sola volta, il tempo di riparazione ed il costo dei componenti presenti anche in più di una sequenza riparata. Riprendendo l'Esempio 6, possiamo rappresentare le sequenze attraverso le variabili x_i.

Esempio 6.1:

x_1: *3 → 4 → 5*

x_2: *4 → 2 → 1 → 4*

x_3: *5 → 5 → 5*

Quindi, se venissero scelte le sequenze 1 e 3 (cioè, $x_1 = x_3 = 1$), sarebbe necessario riparare tutti i componenti di queste sequenze (cioè, $y_3 = y_4 = y_5 = 1$). Riscrivendo i vincoli (3) per il componente 5, si avrà:

$$x_1 + x_3 \leq 2y_5$$

perché il componente 5 è presente sia nella prima che nella terza sequenza. Avendo scelto di riparare entrambe, si avrà:

$$2 \leq 2y_5$$

Per cui la variabile y_5 sarà correttamente valorizzata a 1. Di contro, se si scegliesse di riparare solo la prima sequenza, il vincolo verrebbe riscritto nel modo seguente:

$$1 \leq 2y_5$$

e nuovamente, tale variabile deve assumere valore unitario. Lo stesso ragionamento può essere applicato nel caso in cui ad essere scelta è la terza sequenza. Supponiamo, infine, che nessuna delle due sequenze venga selezionata per essere riparata, significa, quindi, che $x_1 = x_3 = 0$. In tal caso, il vincolo è scritto come:

$$0 \leq 2y_5$$

Affinché il vincolo sia soddisfatto, la variabile y_5 potrà assumere valore nullo. Ciò è ovviamente il risultato che ci aspettavamo poiché un componente che non appartiene ad alcuna sequenza selezionata per la manutenzione, deve avere la relativa variabile y a zero.

Infine, il vincolo (4) descrive la natura delle variabili che, come già evidenziato, sono variabili binarie e possono quindi assumere solo i valori 0 o 1.

Capitolo 4 - Risultati numerici

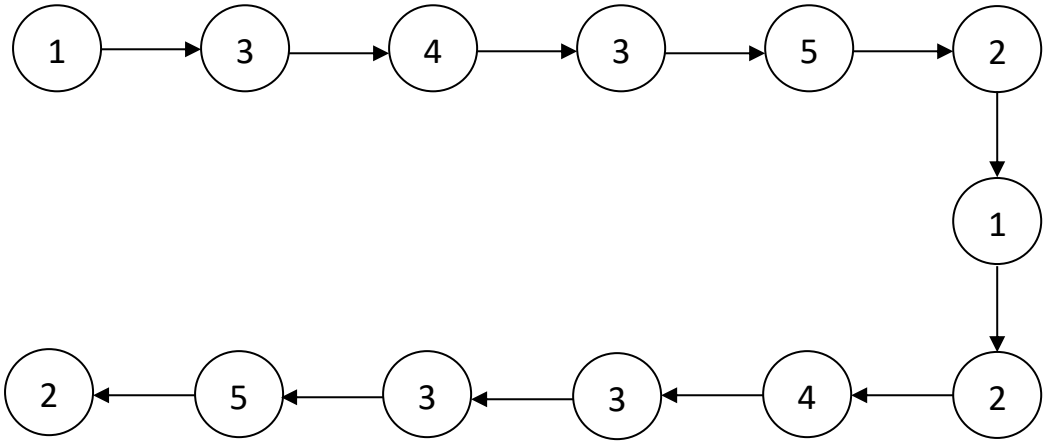
In questo capitolo, si descriverà dapprima come sono state generate le istanze utilizzate nella campagna sperimentale. Si presenteranno, inoltre, i risultati ottenuti e si effettuerà analisi di stabilità al variare di alcuni parametri ritenuti tra i più significativi per gli algoritmi progettati.

4.1 Generazione delle istanze

La generazione delle istanze si può dividere in due fasi. Nella prima fase si crea il grafo di partenza, cioè un unico grafo che contiene tutte le rotture registrate in modo sequenziale. Nella seconda fase, a partire da un componente e fissando un numero di rotture massimo, si individuano le sequenze composte dalle rotture successive a quella del componente scelto, sotto forma di sotto-grafi. In questa fase, si applica, con lo scopo di dare maggior peso alle rotture più recenti, una funzione esponenziale per il calcolo del numero di volte che ogni sotto-grafo dovrà comparire nell'insieme di tutti i sotto-grafi di input per l'algoritmo gSpan [36].

Per la prima fase è stato necessario scegliere il numero di rotture r ed il numero di componenti c del grafo affinché il codice per la generazione creasse un grafo di esattamente r vertici con i componenti, scelti in modo casuale nel *range* [1,…,c] , come etichette.
Ad esempio, scegliendo r = 13 e c = 5, si potrà rappresentare un grafo come quello della Figura 10, in cui ci sono 13 nodi corrispondenti alle 13 rotture e le etichette dei nodi sono i componenti 1, 2, 3, 4 e 5.

Figura 10: Grafo che rappresenta 13 rotture di 5 componenti



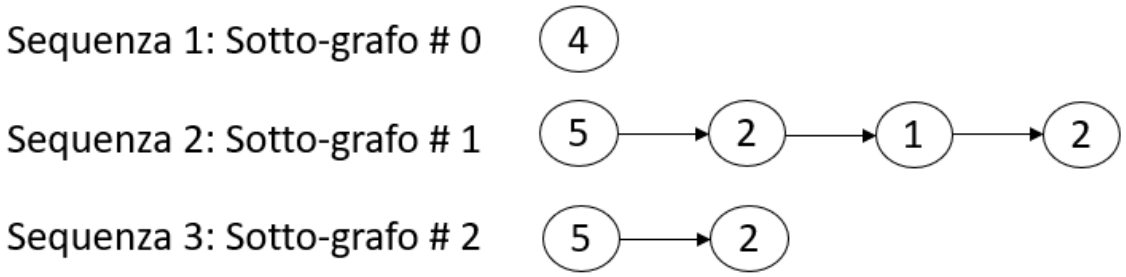
Nella seconda fase, si sceglie un componente per il quale valutare le rotture degli altri componenti a partire dai guasti di quello scelto. Inoltre, è necessario scegliere un massimo numero di rotture x, dopo quella del componente scelto, da considerare. Si creano, in base a questi dati, dei sotto-grafi che avranno come radice il nodo associato alla rottura del componente successivo a quello scelto e al più x nodi. Si sceglie, infatti, di interrompere le sequenze ad ogni nuova rottura del componente di partenza.

Se, ad esempio, si scegliesse di valutare 4 rotture successive a quella del componente 3 del grafo in Figura 10, la prima sequenza, considerando x = 4, è 4 → 3 → 5 → 2, ma si impone che ad una nuova rottura del componente di partenza (3) la sequenza termini. Avremo quindi che alla prima sequenza corrisponderà il primo sotto-grafo, cioè il sotto-grafo con identificativo 0, in Figura 11.

La sequenza 2 e il relativo sotto-grafo in Figura 11 rappresentano il caso in cui una nuova rottura del componente 3 non avviene nell'intervallo di rotture considerato e quindi in questo caso il numero di nodi è esattamente uguale a x, cioè 4.

La presenza di due rotture immediatamente successive del componente scelto nel grafo di partenza, come in Figura 10, non comporta invece la scrittura di alcun sotto-grafo.

Figura 11: Sotto-grafi del grafo in Fig.10 - 4 rotture a partire da quella del componente 3



In questa stessa fase si applica anche un metodo per poter dare più peso alle rotture più recenti. È consigliabile adottare questo approccio in quanto la rottura di un componente dipende dalle condizioni d'esercizio ed esse possono mutare nel tempo.
Ad esempio, potrebbe esserci stati cambiamenti esterni al macchinario che però ne modificano le condizioni d'uso, come un cambiamento nel personale che utilizza il macchinario, oppure un macchinario potrebbe essere sottoposto ad un diverso carico di lavoro rispetto agli inizi a causa del cambiamento dei piani di produzione. È evidente che in questi casi i primi dati sulle rotture potrebbero non essere così rilevanti come gli ultimi.

Per poter dare più peso agli ultimi dati si decide di scrivere più volte i sotto-grafi a cui corrispondono sequenze più recenti e quindi con un numero identificativo più alto.
Per fare ciò si usa la funzione esponenziale:

$$y = ae^{z/b}$$

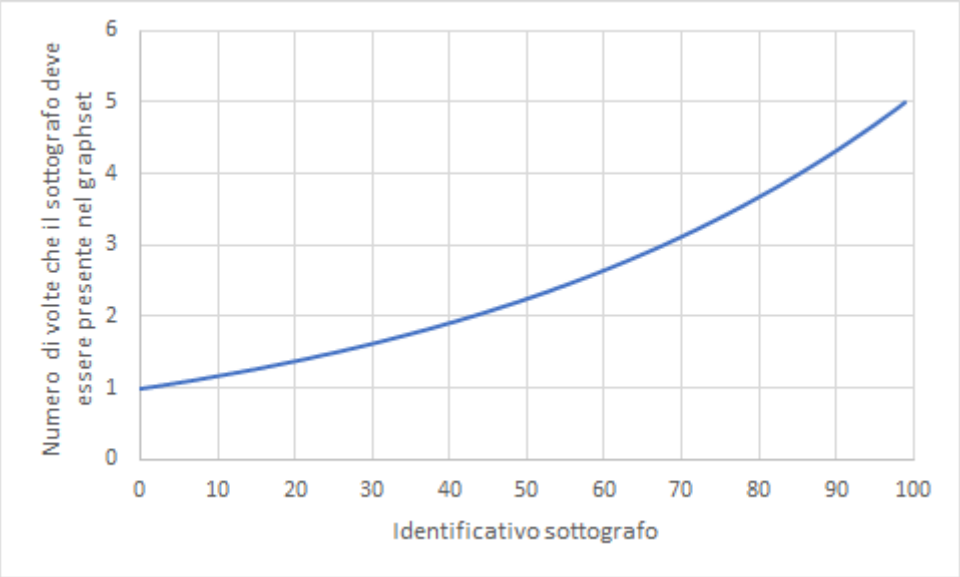
Dove y indicherà il numero di volte che il sotto-grafo sarà presente nell'insieme dei sotto-grafi di input per gSpan, a e b sono parametri a scelta e z sarà l'identificativo del grafo.
Anche se i dati meno recenti potrebbero essere meno rilevanti, essi non vanno esclusi. Anche il sotto-grafo meno recente, che sarà quello con identificativo più basso, dovrà quindi comparire 1 volta. Sapendo che l'identificativo del primo sotto-grafo è 0 e imponendo la condizione y = 1 si ricava il parametro a:

$$1 = ae^{0/b} \Rightarrow 1 = a$$

Il parametro b si può invece ricavare imponendo il numero di volte che vogliamo che compaia la sequenza più recente e quindi il sotto-grafo con identificativo più alto. Se ad esempio si individuano 100 sotto-grafi, l'identificativo più alto sarà 99. Se vogliamo che la sequenza più recente, cioè quella corrispondente al sotto-grafo 99, compaia 5 volte, come per le istanze generate in seguito, avremo:

$$5 = e^{99/b} \Rightarrow ln(5) = \frac{99}{b} \Rightarrow b = \frac{99}{ln(5)} \Rightarrow b \simeq 61.5$$

Figura 12: Rappresentazione della funzione esponenziale $y = ae^{z/b}$ per un graphset di 100 sottografi con $y_{max} = 5$



Si può così calcolare, per ogni sotto-grafo, il numero di volte in cui esso deve essere presente nell'insieme dei sotto-grafi da dare in input a gSpan. Se ad esempio volessimo sapere il numero di volte in cui il grafo 25 dovrebbe essere presente basterà procedere al calcolo di y:

$$y = e^{\frac{25 \cdot ln(5)}{99}} \Rightarrow y \simeq 1.5$$

Ovviamente un sotto-grafo non può essere presente 1.5 volte, si sceglie quindi di approssimare il risultato ottenuto all'intero inferiore: si avrà che y = 1 ed il sotto-grafo 25 sarà presente nell'insieme dei sotto-grafi di input per gSpan una sola volta.

Questa approssimazione viene effettuata con lo scopo di non incrementare oltre il valore y calcolato e quindi consigliato il numero di volte che un sotto-grafo si ripete. In questo modo non avremo sotto-grafi con peso maggiore di quanto previsto. D'altra parte, alcuni grafi verranno pesati meno del previsto (come il grafo 25), ma in ogni caso il loro peso non è trascurato: essendo il loro identificativo maggiore di 0, ne è garantita la presenza almeno una volta. Inoltre, il peso dei sotto-grafi scalerà comunque, seppure più lentamente, in base a quanto sono recenti.

Infine, l'applicazione di questo metodo di approssimazione permette di aumentare con più moderazione la cardinalità dell'insieme dei sotto-grafi di input per gSpan, garantendo anche tempi di computazione più brevi. Infatti, come si vedrà nella Sezione 4.2, questi tempi dipendono dalla cardinalità dell'insieme.

Attraverso queste due fasi sono state create 7 istanze. Per poter valutare un maggior numero di casi è stato scelto di far variare il numero di rotture (e quindi il numero di nodi del grafo di partenza) da 10000 a 200000, come si può vedere nella seconda colonna della Tabella 4. Inoltre, si è fatto variare il numero di

componenti che possono essere soggetti a rotture da 13 a 20, come nella colonna 3 della stessa tabella.

Dopo una prima scelta, sono stati mantenuti costanti il numero di volte che il sotto-grafo con identificativo più alto sarà presente nell'insieme dei sotto-grafi (y_{max} = 5), il numero di rotture successive a quella del componente di partenza (x = 10, come nella colonna 5 della Tabella 4) ed il componente stesso (4, come nella colonna 4 della Tabella 4).
Inoltre, nella stessa tabella sono stati riportati il numero di sotto-grafi individuati senza e con l'applicazione della funzione esponenziale per la pesatura, rispettivamente nelle colonne 6 e 7.

Tabella 4: Istanze generate

Istanza	Rotture	Numero di componenti	Componente di partenza scelto	Numero Rotture Scelto (x)	Sotto-grafi	Sotto-grafi dopo $y = ae^{z/b}$
1	200000	15	4	10	12416	25149
2	100000	20	4	10	4724	9569
3	100000	15	4	10	6287	12735
4	100000	13	4	10	7103	14388
5	50000	15	4	10	3154	6390
6	25000	15	4	10	1607	3256
7	10000	15	4	10	612	1240

4.2 Risultati

Sulle istanze descritte in 4.1 si fa data mining attraverso l'uso di gSpan per estrarre le sequenze più frequenti ed il loro supporto. L'algoritmo riceve in input il supporto minimo, che rappresenta la soglia, scelta dall'utente, oltre la quale un sotto-grafo viene considerato frequente e quindi, riportato in output o meno.

In questa prima fase, si decide di mantenere il supporto costante al 2% in modo da poter valutare come gli altri parametri, cioè il numero di componenti e il numero di rotture considerate, influenzino i risultati. Ad esempio, per l'istanza 1, che è composta da 25149 sotto-grafi, avremo:

$$25149 \cdot 0.02 = 502.98$$

Cioè verranno estratti tutti i sotto-grafi che compaiono almeno 503 volte nell'insieme dei sotto-grafi di input.

Allo stesso modo per l'istanza 2 viene mantenuto un supporto pari al 2% ma, in questo caso, l'insieme è composto solamente da 9569 sotto-grafi e si avrà quindi:

$$9569 \cdot 0.02 = 191.38$$

cioè si estrarranno tutti i sotto-grafi che compaiono almeno 191 volte nell'insieme.

Usando l'output dell'algoritmo si individuano le sequenze e i loro supporti ed i componenti. In base a questi dati, verrà costruita la matrice M come indicato nel Capitolo 3 necessaria per l'implementazione del modello di PLI. È infine necessario fornire i dati riguardanti il costo ed il tempo di riparazione di ciascun componente, il budget ed il tempo massimo a disposizione per la manutenzione.

Come per il supporto, si sceglie, in questa fase, di mantenere costanti questi parametri. In particolare, i risultati presentati nella Tabella 5 vengono ottenuti con budget pari a 1700 (B=1700) e tempo per la manutenzione pari a 170 (T=170).

Tabella 5: Risultati con B=1700, T=170 e costi e tempi di riparazione dei componenti costanti

Istanza	Supporto usato su gSpan	Tempo data-mining con gSpan (s)	Numero di sotto-grafi frequenti estratti	Solve Time AMPL (s)	Funzione obiettivo	Tempo totale (s)
1	2%	9.5	196	0.156	0.84902	9.656
2	2%	2.73	156	0.313	0.381545	3.043
3	2%	4.13	196	0.188	0.864389	4.318
4	2%	4.16	144	0.141	1.06123	4,301
5	2%	3.09	196	0.172	0.841784	3,262
6	2%	2.08	196	0.172	0.824939	2,252
7	2%	1.34	186	0.219	0.825806	1,559

In primo luogo, si può notare come la maggior parte del tempo totale di elaborazione sia dovuto al *data mining* mentre soltanto una piccola parte di esso per la risoluzione del modello di PLI. In particolare, i tempi di computazione di gSpan dipendono in modo direttamente proporzionale dalla cardinalità dell'insieme dei sotto-grafi generato precedentemente.

Secondariamente, analizzando le istanze 1, 3, 5, 6 e 7 delle Tabelle 4 e 5 si nota che, a seguito di una diminuzione del numero di rotture considerate, a parità di componenti, la funzione obiettivo non subisce forti cambiamenti. Il metodo sembra quindi possedere una buona scalabilità rispetto al numero di rotture considerato e garantire un'affidabilità costante anche variando la dimensione del *dataset*.

Infine, osservando le istanze 2, 3 e 4 si può notare come una diminuzione del numero di componenti presi in considerazione comporti un'affidabilità

maggiore in quanto la funzione obiettivo dell'istanza con minor componenti è maggiore di quella con più componenti.

4.3 Analisi di stabilità: supporto, budget e tempo di manutenzione

Nell'analisi dei risultati della Sezione 2 di questo capitolo è stato scelto di mantenere costanti il supporto, il budget (B = 1700) e il tempo massimo a disposizione (T=170) per la manutenzione. In questa sezione, si vedrà come variano i risultati al cambiamento di questi parametri.
Per tale scopo si sceglie l'istanza 3 e su di essa si operano i cambiamenti.

4.3.1 Variazione della soglia di supporto minimo

In questa sezione, si farà variare la soglia di supporto minimo, mantenendo costanti tutti gli altri parametri (costo e tempo di manutenzione dei componenti, budget e tempo massimo a disposizione per la manutenzione).
Si valuta la variazione della soglia di supporto minimo in un range che va dall'1% al 4% con passo 0.2%.
I risultati ottenuti si possono consultare nella Tabella 6.

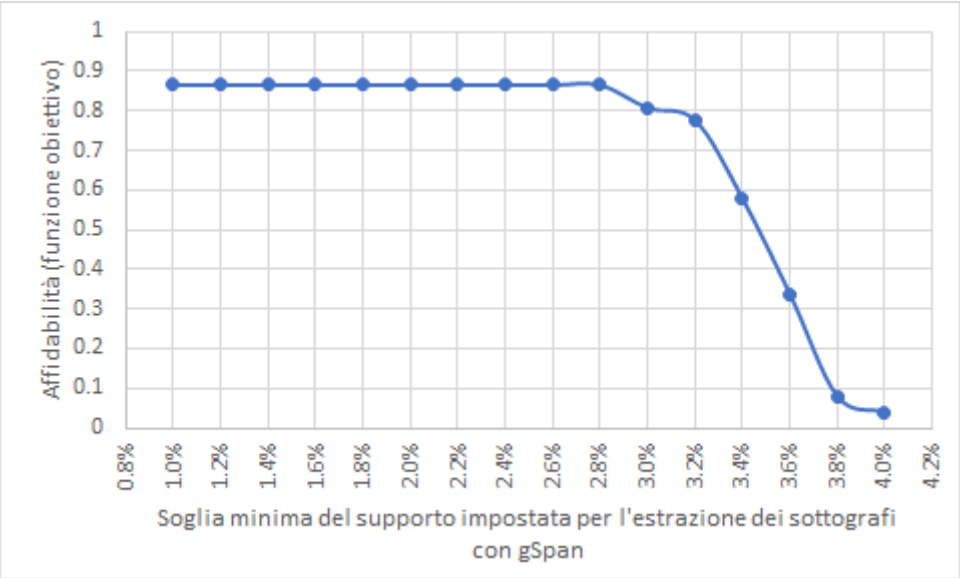
Tabella 6: Risultati per la variazione della soglia minima di supporto

Istanza	Supporto gSpan	Tempo di mining gSpan (s)	Numero di sotto-grafi frequenti estratti	Solve time AMPL (s)	Funzione obiettivo	Tempo totale (s)
3.1.0	1%	4.7	196	0.188	0.864389	4.888
3.1.1	1.2%	4.33	196	0.188	0.864389	4.518
3.1.2	1.4%	4.69	196	0.188	0.864389	4.878
3.1.3	1.6%	4.59	196	0.188	0.864389	4.778

3.1.4	1.8%	5.25	196	0.188	0.864389	5.438
3.1.5	2.0%	5.93	196	0.188	0.864389	6.118
3.1.6	2.2%	4.47	196	0.188	0.864389	4.658
3.1.7	2.4%	4.34	196	0.188	0.864389	4.528
3.1.8	2.6%	4.14	196	0.188	0.864389	4.328
3.1.9	2.8%	4.84	190	0.234	0.864389	5.074
3.1.10	3.0%	4.45	173	0.234	0.806989	4.684
3.1.11	3.2%	4.86	135	0.156	0.775658	5.016
3.1.12	3.4%	2.98	74	0.093	0.577621	3.073
3.1.13	3.6%	2.48	28	0.062	0.336553	2.542
3.1.14	3.8%	2.48	3	0.031	0.0791519	2.511
3.1.15	4.0%	2.38	1	0.031	0.0399686	2.411

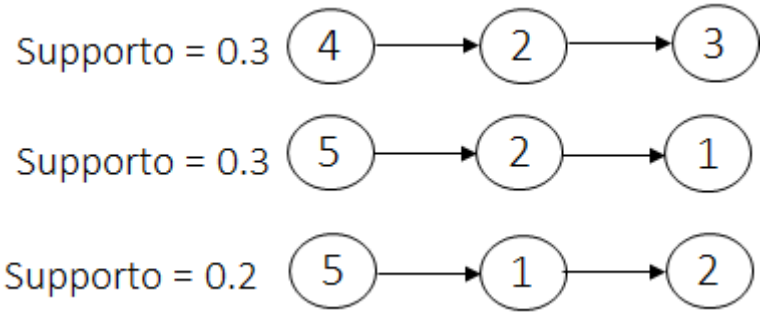
Si nota come all'aumentare della soglia minima di supporto oltre il 2.6% (colonna 2), diminuiscono, ovviamente, i sotto-grafi estratti (colonna 4) e di conseguenza anche il valore della funzione obiettivo come è evidenziato anche dalla Figura 13. In particolare, dopo la soglia del 3%, se si fa aumentare il supporto minimo dello 0.2% si ha una diminuzione di circa il 4% della funzione obiettivo, da 0.807 a 0.776. Se si decide di aumentare ancora il supporto minimo la diminuzione della funzione obiettivo supera il 25% per ogni aumento dello 0.2%, da 0.776 a 0.578 a 0.337. Questo trend termina quando la soglia è talmente alta da non permettere l'estrazione di alcun sotto-grafo.

Figura 13: Grafico che rappresenta il valore della funzione obiettivo in funzione della soglia di supporto minimo



Il secondo fenomeno è spiegabile con un esempio:
se con soglia minima = 0.2 fossero estratti i 3 sotto-grafi della Figura 13 con i relativi supporti e ipotizzassimo costi e tempi di riparazione uguali per le componenti e una disponibilità di budget e tempo massimo in grado di consentire la riparazione di soli 3 componenti, avremmo che il modello PLI, per massimizzare l'affidabilità sceglierebbe di riparare le componenti della sequenza rappresentata dal secondo grafo in quanto la loro riparazione consente di riparare anche la sequenza rappresentata dal terzo grafo. L'affidabilità sarebbe quindi pari a 0.5. Se invece la soglia minima fosse portata a 0.3, il terzo grafo verrebbe escluso e la riparazione dei componenti 5, 2 e 1 permetterebbe la riparazione completa solo della sequenza rappresentata dal secondo grafo (o quella del primo), comportando quindi una diminuzione dell'affidabilità fino allo 0.3. Non è quindi consigliabile l'eliminazione, attraverso un eccessivo innalzamento della soglia di supporto minimo, di sotto-grafi statisticamente rilevanti in quanto comporterebbe una riduzione dell'affidabilità del metodo. D'altro canto, non si dovrebbe adottare un supporto minimo troppo basso per evitare di prendere in considerazione casi poco rilevanti.

Figura 14: Grafi e relativo supporto



4.3.2 Variazione del budget massimo B a disposizione per la manutenzione

In questa sezione si analizzeranno i risultati ottenuti facendo variare il budget a disposizione per la manutenzione. Anche in questo caso si userà l'istanza 3 e si manterranno costanti il supporto minimo (2%) e il tempo massimo a disposizione per la manutenzione, che viene impostato a 220. Si sceglie di valutare gli effetti di questa variazione in un range di budget tra 200 e 10000 con passo pari a 200.

I risultati ottenuti si possono consultare nella Tabella 7.

Tabella 7: Risultati in seguito alla variazione del budget B

Istanza	Budget	Solve time AMPL (s)	Funzione obiettivo
3.2.1	200	0.031	0.031331
3.2.2	400	0.141	0.133804
3.2.3	600	0.047	0.301060
3.2.4	800	0.063	0.312367
3.2.5	1000	0.109	0.545583

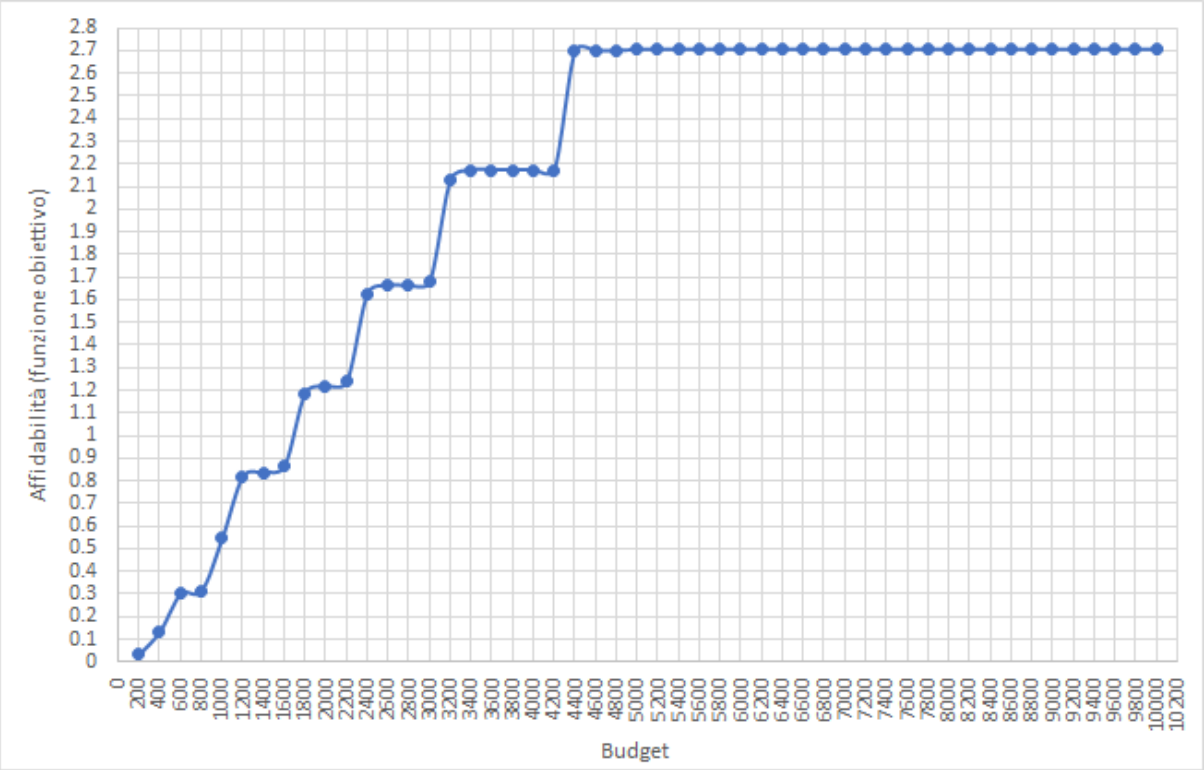
3.2.6	1200	0.125	0.819631
3.2.7	1400	0.141	0.837534
3.2.8	1600	0.188	0.864389
3.2.9	1800	0.141	1.18744
3.2.10	2000	0.188	1.2179
3.2.11	2200	0.188	1.23785
3.2.12	2400	0.141	1.62748
3.2.13	2600	0.156	1.66298
3.2.14	2800	0.219	1.66298
3.2.15	3000	0.266	1.67978
3.2.16	3200	0.219	2.12854
3.2.17	3400	0.434	2.17275
3.2.18	3600	0.422	2.17275
3.2.19	3800	0.469	2.17275
3.2.20	4000	0.453	2.17275
3.2.21	4200	0.516	2.17275
3.2.22	4400	0.171	2.6987
3.2.23	4600	0.172	2.6987
3.2.24	4800	0.188	2.6987

3.2.25	5000	0.156	2.70428
⋮	⋮	⋮	⋮
3.2.50	10000	0.141	2.70428

Si nota come, all'aumentare del budget disponibile, il valore della funzione obiettivo aumenti in quanto aumentano i componenti che si è in grado di riparare e di conseguenza le sequenze in cui questi componenti sono presenti. Questo incremento della funzione obiettivo si ferma per B ≥ 5000 e rimane, quindi, costante anche se si decide di investire un budget maggiore per la manutenzione, a parità di tempo massimo per cui il macchinario può restare inattivo.

L'andamento della funzione obiettivo è rappresentato dalla Figura 15. Dal grafico è facile vedere che la funzione obiettivo è costante a tratti, infatti l'aumento del budget permette di riparare nuove combinazioni di componenti e le sequenze in cui compaiono, permettendo quindi un aumento dell'affidabilità che rimane poi costante fino ad un aumento del budget tale da sbloccare nuove combinazioni. Se ad esempio si decide di investire B = 1800 per la manutenzione ma non si se è soddisfatti di un'affidabilità pari a 1.187 come nel caso 3.2.9, per avere un aumento rilevante, ad esempio di circa il 37%, cioè fino a 1.627 come nel caso 3.2.12, è necessario aumentare il budget fino a 2400, quindi di circa il 33%.

Figura 15: Andamento della funzione obiettivo al cambiamento del budget



4.3.3 Variazione del tempo massimo T a disposizione per la manutenzione

In questa sezione si analizzeranno i risultati ottenuti facendo variare il tempo massimo a disposizione per la manutenzione. Anche in questo caso si userà l'istanza 3 e si manterranno costanti il supporto minimo (2%) e il budget a disposizione per la manutenzione, che viene impostato a 2200. Si sceglie di valutare gli effetti di questa variazione in un range di budget tra 10 e 300 con passo pari a 10.

I risultati ottenuti si possono consultare nella Tabella 8.

Tabella 8: Risultati in seguito alla variazione del tempo a disposizione T

Istanze	T	Solve time AMPL (s)	Funzione obiettivo
3.3.1	10	0.031	0.0343149
3.3.2	20	0.047	0.126894
3.3.3	30	0.031	0.135846
3.3.4	40	0.047	0.291166
3.3.5	50	0.094	0.302159
3.3.6	60	0.094	0.523675
3.3.7	70	0.172	0.528779
3.3.8	80	0.141	0.536631
3.3.9	90	0.141	0.54095
3.3.10	100	0.188	0.832352
3.3.11	110	0.266	0.832587
3.3.12	120	0.250	0.846408
3.3.13	130	0.219	1.1872
3.3.14	140	0.234	1.1872
3.3.15	150	0.188	1.21076
3.3.16	160	0.219	1.21076

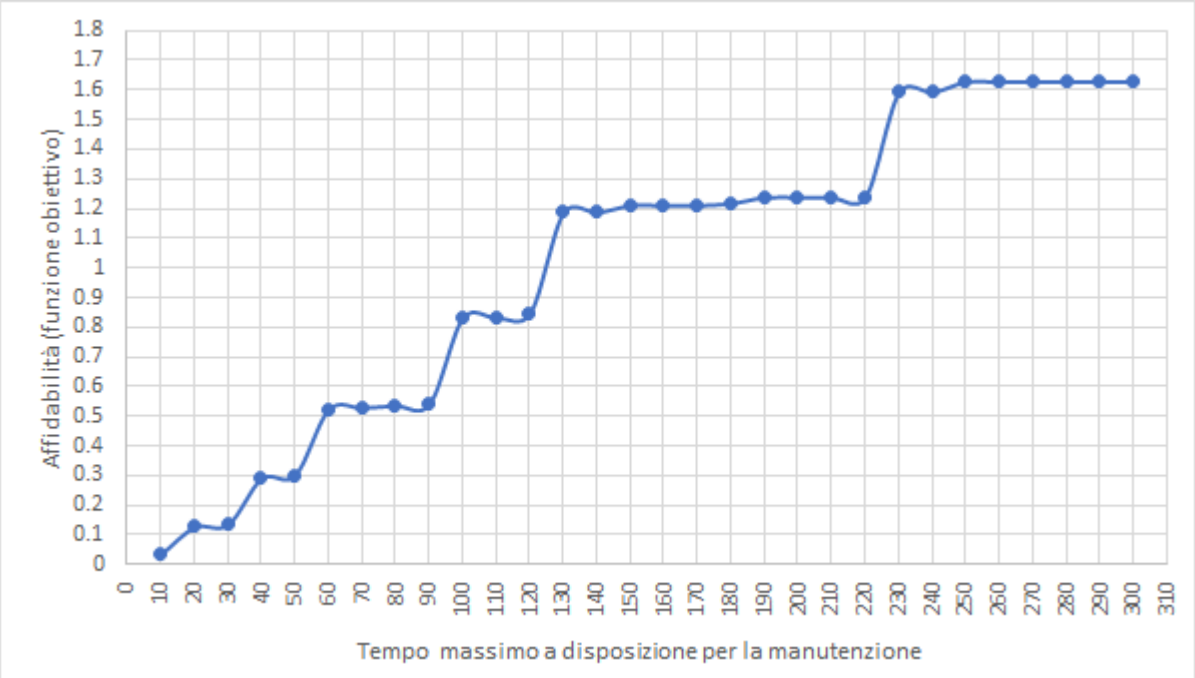
3.3.17	170	0.250	1.21076
3.3.18	180	0.250	1.2179
3.3.19	190	0.219	1.23785
3.3.20	200	0.234	1.23785
3.3.21	210	0.219	1.23785
3.3.22	220	0.250	1.23785
3.3.23	230	0.188	1.59246
3.3.24	240	0.188	1.59246
3.3.25	250	0.203	1.62701
3.3.26	260	0.188	1.62701
3.3.27	270	0.219	1.62701
3.3.28	280	0.188	1.62701
3.3.29	290	0.188	1.62701
3.3.30	300	0.188	1.62701

Come per il budget, all'aumentare tempo disponibile per la manutenzione il valore della funzione obiettivo aumenta. Anche in questo caso l'aumento è dovuto alla capacità di riparare ulteriori componenti e di conseguenza le sequenze in cui questi componenti sono presenti. La crescita dell'affidabilità si ferma per $T \geq 250$ e rimane costante anche se si decide di dedicare più tempo alla manutenzione, a parità di budget.

L’andamento della funzione obiettivo è rappresentato dalla figura 16. Dal grafico è ancora una volta possibile vedere che la funzione obiettivo è costante a tratti: l’aumento del tempo massimo a disposizione permette di riparare nuove combinazioni di componenti e le sequenze in cui compaiono, permettendo quindi un aumento dell’affidabilità che rimane poi costante fino ad un aumento del tempo tale da sbloccare nuove combinazioni.

Se ad esempio si decide di permettere che il macchinario resti inattivo per la manutenzione per T = 110 ma non si se è soddisfatti di un’affidabilità pari a 0.833 come nel caso 3.3.11, è necessario, per avere un aumento rilevante dell’affidabilità, ad esempio di circa il 42%, cioè fino a 1.187 come nel caso 3.2.13, è necessario che il macchinario resti fermo per T = 130, quindi bisogna scegliere di aumentare il tempo di inattività del 18%.

Figura 16: Andamento della funzione obiettivo al cambiamento del tempo totale per la manutenzione



Capitolo 5 - Conclusioni e sviluppi futuri

Lo sviluppo di questa tesi nasce dalla necessità, per una qualsiasi azienda, di ottimizzare le attività di manutenzione evitando continue interruzioni del flusso produttivo attraverso la riparazione di componenti a rischio prima che la loro rottura avvenga in modo tale da essere competitivi sul mercato e migliorare la sicurezza dell'impianto.

L'approccio proposto si basa sulla descrizione delle informazioni sulle rotture attraverso un grafo, su cui, a partire dalla rottura di un componente, si possono individuare le sequenze composte dalle rotture degli altri componenti sotto forma di sotto-grafi.

Ovviamente, le rotture più recenti dovrebbero riflettere in modo più fedele le condizioni d'uso del macchinario. È quindi stata necessaria l'introduzione di un metodo per poter pesare in modo differente i vari sotto-grafi e dare più peso a quelli più recenti. Esso consiste nell'applicazione di una funzione esponenziale usata per il calcolo del numero delle volte che un sotto-grafo dovrà apparire nell'insieme dei sotto-grafi di input per l'algoritmo di *graph-mining*.

Con gSpan si estraggono quindi i sotto-grafi con una frequenza maggiore di quella desiderata che insieme a costi e tempi di riparazioni dei componenti, budget e tempo massimo a disposizione per la manutenzione diventano input di un modello matematico di PLI, che ha l'obiettivo di massimizzare l'affidabilità complessiva del sistema selezionando le sequenze e quindi i componenti da riparare.

In questo modo, sfruttando la necessità di interrompere le attività produttive per la rottura di un componente, si possono aggiustare anche altri componenti, a rischio di rottura, in modo tale da prevenire ulteriori tempi di inattività e garantire, nel modo più affidabile possibile, che il flusso dei processi non venga ulteriormente interrotto.

5.1 Conclusioni

Il metodo sviluppato risulta molto efficiente, sia nell'estrazione dei sotto-grafi che, in particolare, nella soluzione del modello di PLI. Infatti, anche con il *dataset* più grande (esempio, l'istanza 1 con 200000 rotture e 15 componenti), si è in grado di ottenere risultati in meno di 10 secondi.

Inoltre, il metodo, applicato ad istanze create artificialmente, è risultato particolarmente scalabile rispetto alla cardinalità dell'insieme dei sotto-grafi di input per gSpan, garantendo un'affidabilità abbastanza uniforme a vari livelli di grandezza dello stesso.

Si è poi analizzata la stabilità delle soluzioni rispetto alla soglia di supporto minimo utilizzata per l'estrazione con gSpan, al budget e al tempo massimo a disposizione per la manutenzione sull'istanza 3. Da quest'analisi è risultato che non è consigliabile innalzare eccessivamente la soglia del supporto minimo in quanto comporterebbe l'esclusione di sotto-grafi rilevanti e quindi la diminuzione dell'affidabilità finale sistema. D'altra parte, non è consigliabile considerare tutti i sotto-grafi, anche quelli che compaiono con frequenza minima, perché ciò potrebbe falsare i risultati ottenuti, facendo aumentare il valore della funzione obiettivo.

Infine, le soluzioni ottenute in seguito alla variazione del budget e del tempo a disposizione risultano abbastanza stabili in determinati range, oltre i quali, ad una variazione dei parametri, corrisponde una notevole variazione della funzione obiettivo. Questo è particolarmente vero, nell'istanza considerata, per il parametro di tempo, per cui ad una sua piccola variazione oltre il range ne corrisponde una notevole della funzione obiettivo.

5.2 Sviluppi futuri

In futuro, si potrebbe effettuare una campagna sperimentale sulle stesse istanze generate in questa tesi ma attraverso la creazione di un grafo che ha come etichetta di ogni nodo il componente e la probabilità che la rottura del componente avvenga, come peso degli archi la probabilità di rottura del

componente (testa dell'arco) in seguito alla rottura del componente di partenza (coda dell'arco).

Come già detto, i test effettuati sono stati condotti su istanze generate artificialmente, non è quindi ancora possibile verificare la validità del metodo in una situazione reale. Per poter fare ciò è consigliabile, in futuro, applicare il metodo su di un dataset reale ed effettuare su di esso tutti gli studi, già fatti sui test artificiali.

Potrebbe essere particolarmente interessante, in particolare nei casi in cui l'inattività di un macchinario comporti lo stop di tutto l'impianto, formulare un metodo per la gestione delle manutenzioni di tutti i macchinari dell'impresa in modo da ottimizzare non solo la manutenzione sullo specifico macchinario, ma su tutto l'impianto.

Un ulteriore sviluppo per il modello potrebbe essere quello di applicare una funzione multi-obiettivo in modo da non solo massimizzare l'affidabilità, nel rispetto dei limiti di budget e di tempo, ma anche minimizzare i costi ed il tempo necessario per la manutenzione.

Bibliografia

1. Mobley, R. K. (2002). *An introduction to predictive maintenance*. Elsevier.

2. MASINI, P., & ROMANO, F. Dynamic Maintenance Management System la manutenzione dinamica per il miglioramento continuo di Trenitalia.

3. Harding, J. A., Shahbaz, M., & Kusiak, A. (2006). Data mining in manufacturing: a review. *Journal of Manufacturing Science and Engineering*, *128*(4), 969-976.

4. Antomarioni, S., Pisacane, O., Potena, D., Bevilacqua, M., Ciarapica, F. E., & Diamantini, C. (2019). A predictive association rule-based maintenance policy to minimize the probability of breakages: application to an oil refinery. *The International Journal of Advanced Manufacturing Technology*, 1-15.

5. Yan, X., & Han, J. (2002). gspan: Graph-based substructure pattern mining. In *2002 IEEE International Conference on Data Mining, 2002. Proceedings.* (pp. 721-724). IEEE.

6. Elovici, Y., & Braha, D. (2003). A decision-theoretic approach to data mining. *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans*, *33*(1), 42-51.

7. Sharma, A., Yadava, G. S., & Deshmukh, S. G. (2011). A literature review and future perspectives on maintenance optimization. *Journal of Quality in Maintenance Engineering*, *17*(1), 5-25.

8. Fayyad, U., Piatetsky-Shapiro, G., & Smyth, P. (1996). From data mining to knowledge discovery in databases. *AI magazine*, *17*(3), 37-37.

9. Feng, J. X., & Kusiak, A. (2006). Data mining applications in engineering design, manufacturing and logistics. *International Journal of Production Research*, *44*(14), 2689-2694.

10. Romanowski, C. J., & Nagi, R. (1999, May). Improving preventive maintenance scheduling using data mining techniques. In *8th Industrial Engineering Research Conference*.

11. Batanov, D., Nagarur, N., & Nitikhunkasem, P. (1993). EXPERT-MM: A knowledge-based system for maintenance management. *Artificial intelligence in engineering*, *8*(4), 283-291.

12. Yuan, B., Wang, X. Z., & Morris, T. (2000). Software analyser design using data mining technology for toxicity prediction of aqueous effluents. *Waste Management*, *20*(8), 677-686.

13. Létourneau, S., Famili, F., & Matwin, S. (1999). Data mining to predict aircraft component replacement. *IEEE Intelligent Systems and their Applications*, *14*(6), 59-66.

14. Skormin, V. A., Gorodetski, V. I., & Popyack, L. J. (2002). Data mining technology for failure prognostic of avionics. *IEEE Transactions on Aerospace and Electronic Systems*, *38*(2), 388-403.

15. Yam, R. C. M., Tse, P. W., Li, L., & Tu, P. (2001). Intelligent predictive decision support system for condition-based maintenance. *The International Journal of Advanced Manufacturing Technology*, *17*(5), 383-391.

16. Huang, C. L., Li, T. S., & Peng, T. K. (2005). A hybrid approach of rough set theory and genetic algorithm for fault diagnosis. *The International Journal of Advanced Manufacturing Technology*, *27*(1-2), 119-127.

17. Antomarioni, S., Bevilacqua, M., Potena, D., & Diamantini, C. (2019). Defining a data-driven maintenance policy: an application to an oil refinery plant. *International Journal of Quality & Reliability Management*, *36*(1), 77-97.

18. Antony, J. V., & Nasira, G. (2014). Enabling predictive maintenance strategy in rail sector: a clustering approach. *Wseas Transactions On Computers*, *13*, 118-128.

19. Sammouri, W., Côme, E., Oukhellou, L., Aknin, P., & Fonlladosa, C. E. (2014, October). Pattern recognition approach for the prediction of infrequent target events in floating train data sequences within a predictive maintenance framework. In *17th International IEEE Conference on Intelligent Transportation Systems (ITSC)* (pp. 918-923). IEEE.

20. Li, J. R., Khoo, L. P., & Tor, S. B. (2006). RMINE: a rough set based data mining prototype for the reasoning of incomplete data in condition-based fault diagnosis. *Journal of Intelligent Manufacturing*, *17*(1), 163-176.

21. Ben-Daya, M., & Duffuaa, S. O. (1995). Maintenance and quality: the missing link. *Journal of quality in maintenance engineering*, *1*(1), 20-26.

22. Al-Sultan, K. S., & Duffuaa, S. O. (1995). Maintenance control via mathematical programming. *Journal of Quality in Maintenance Engineering*, *1*(3), 36-46.

23. Vatn, J., Hokstad, P., & Bodsberg, L. (1996). An overall model for maintenance optimization. *Reliability Engineering & System Safety*, *51*(3), 241-257.

24. Bevilacqua, M., & Braglia, M. (2000). The analytic hierarchy process applied to maintenance strategy selection. *Reliability Engineering & System Safety*, *70*(1), 71-83.

25. Jin, X., Li, L., & Ni, J. (2009). Option model for joint production and preventive maintenance system. *International Journal of Production Economics*, *119*(2), 347-353.

26. Oke, S. A. (2005). An analytical model for the optimisation of maintenance profitability. *International Journal of Productivity and Performance Management*, *54*(2), 113-136.

?27. Ho, L. L., & Silva, A. F. (2006). Unbiased estimators for mean time to failure and percentiles in a Weibull regression model. *International Journal of Quality & Reliability Management*, *23*(3), 323-339.

28. Mechefske, C. K., & Wang, Z. (2001). Using fuzzy linguistics to select optimum maintenance and condition monitoring strategies. *Mechanical Systems and Signal Processing*, *15*(6), 1129-1140.

29. Swanson, L. (2003). An information-processing model of maintenance management. *International Journal of Production Economics*, *83*(1), 45-64.

30. Marseguerra, M., Zio, E., & Podofillini, L. (2002). Condition-based maintenance optimization by means of genetic algorithms and Monte Carlo simulation. *Reliability Engineering & System Safety*, *77*(2), 151-165.

31. Matsuoka, S., & Muraki, M. (2007). Short-term maintenance scheduling for utility systems. *Journal of Quality in Maintenance Engineering*, *13*(3), 228-240.

32. Kianfar, F. (2005). A numerical method to approximate optimal production and maintenance plan in a flexible manufacturing system. *Applied Mathematics and Computation*, *170*(2), 924-940.

33. Cadini, F., Zio, E., & Avram, D. (2009). Model-based Monte Carlo state estimation for condition-based component replacement. *Reliability Engineering & System Safety*, *94*(3), 752-758.

34. Bevilacqua, M., Braglia, M., Frosolini, M., & Montanari, R. (2005). Failure rate prediction with artificial neural networks. *Journal of Quality in Maintenance Engineering*, *11*(3), 279-294.

35. Nahas, N., Khatab, A., Ait-Kadi, D., & Nourelfath, M. (2008). Extended great deluge algorithm for the imperfect preventive maintenance optimization of multi-state systems. *Reliability Engineering & System Safety*, *93*(11), 1658-1672.

36. QingYing, Chen (2018). gSpan. GitHub repository https://github.com/betterenvi/gSpan.