UNIVERSITA' POLITECNICA DELLE MARCHE

FACOLTA' DI INGEGNERIA

Corso di Laurea magistrale in ingegneria meccanica

Curriculum: Sistemi produttivi e tecnologie innovative

Sviluppo di algoritmi di ottimizzazione per lo scheduling per gli ordini di produzione.

Development of optimization algorithms for scheduling for production orders

Relatore:

Prof. Ing. Ciarapica Filippo Emanuele

Tesi di Laurea di:

Del Gallo Mateo

A.A. **2021/ 2022**

Sommario

1. INTRODUZIONE

Nella seguente trattazione si andranno ad approfondire i concetti legati alla schedulazione di ordini e progetti e verranno riportati anche diversi casi di studio per passare poi alla descrizione dettagliata di algoritmi per lo scheduling.

Il testo è strutturato in modo tale che il primo capitolo fornisca una panoramica generale sulla teoria della schedulazione e le diverse tipologie e problematiche legate all'argomento. Nel secondo capitolo verrenno analizzati casi di studio attuali trovati in letteratura che utilizzano diverse tipologie di intelligenza artificiale per risolvere il problema della schedulazione.

La parte centrale della trattazione si focalizza sulla descrizione dettagliata dell'algoritmo di schedulazione realizzato in Python grazie alla libreria Pyomo che forniva una buona base di partenza per scrivere il modello matematico di riferimento. L'algoritmo realizzato ha come obiettivo quello di minimizzare il makespan sotto l'ipotesi di capacità delle macchine di lavoro infinite e la possibilità di una macchina di lavorare un solo articolo per volta.

L'algoritmo realizzato, oltre risolvere il modello, presenta diverse funzioni importanti come la possibilità di inserire i tempi di setup e il codice cliente del prodotto ma anche la possibilità di non leggere i tempi assoluti ma i tempi convertiti con la data attuale. Inoltre, l'algoritmo sviluppato permette di ricevere i dati in input non direttamente da Python ma tramite fogli di calcolo che possono essere sia formato csv che formato xlsx.

L'algoritmo è stato testato in diverse condizioni per valutarne i tempi di calcolo al variare di 3 fattori quali la varietà di articoli da produrre, il numero di macchine e la tipologia del PC. Lo scopo è quello di individuare una relazione tra i tempi di calcolo e questi 3 fattori. Eseguendo un'analisi DoE si è arrivati alla formulazione di un modello matematico che descrive l'influenza che tali parametri hanno sui tempi di calcolo.

Infine, l'algoritmo è stato testato con successo su un caso reale dell'azienda 'Matrix Modelleria'. Tale azienda, esperta nella lavorazione di materiali ceramici prevedeva la

produzione di una serie di articoli, i quali dovevano subire tutte le stesse lavorazioni e nella medesima sequenza.

2. LA SCHEDULAZIONE

La schedulazione di un ordine o di un progetto può essere intesa come una roadmap che fissa durata, sequenza di eventi e attività. Si possono intendere operazioni di schedulazione come forme di decision-making volte all'allocazione di risorse al fine di ottimizzare un dato obiettivo.

È importante non confondere la schedulazione con il sequenziamento in quanto quest'ultimo ha lo scopo di definire solamente l'ordine delle operazioni da eseguire mentre la prima si occupa anche di sincronizzare e tempificare le singole attività. La schedulazione include il sequenziamento ma non viceversa.

2.1. Introduzione alla schedulazione

I problemi di scheduling hanno come obiettivo quello di trovare la modalità di assegnamento di una risorsa per lo svolgimento di una serie di attività in modo ottimale.

Di seguito vengono definiti termini usati durante la schedulazione:

- Task: è una parte di attività, è il punto più basso di una WBS (work breakdown structure). La WBS è una rappresentazione grafica dove partendo dal progetto globale lo si scompone in sotto progetti e successivamente in attività elementari in base alle conoscenze disponibili sul modo di eseguire il progetto.
- Fase: è parte di lavoro fatta su un lasso di tempo significativo all'interno del progetto
- Eventi: rappresentano il punto di inizio e di fine di ogni attività
- Milestone: è il raggiungimento di un significativo evento all'interno di un progetto.

È importante definire le relazioni che possono esistere tra le varie task tramite l'ausilio della precedence diagramming method (PDM). È un metodo di costruzione di un project network diagram usando nodi che rappresentano attività e le linee di congiunzione fra i vari nodi indicano dei vincoli di dipendenza. Ogni elemento della WBS è rappresentato nel project network diagram. (vedi appendice 1).

2.2. Schedulazione dei lotti su macchina singola

Nel caso di lavorazione di un numero definito di lotti di articoli che devono essere lavorati sulla stessa macchina è di fondamentale importanza cercare la sequenza ottima di lavorazione al fine di poter ottimizzare uno dei seguenti parametri:

- flow time: differenza tra l'istante temporale d'ingresso e quello di uscita del singolo lotto
- Makespan: intervallo di tempo necessario al completamento di un certo numero di lotti
- Tardiness: differenza tra l'istante di completamento del lotto e quello di consegna, se è maggiore di zero allora ci sarà un ritardo di consegna.

Per determinare la data di inizio di lavorazione di un job specifico si può procedere con i seguenti due approcci:

- Forward Scheduling: in questo caso i job verranno lavorati non appena si liberi la capacità produttiva, non serve conoscere le date di consegna. Questo approccio, tuttavia, non è il più consigliato perché tende a generare un valore eccessivo di WIP (Work In Progress).
- Backward Scheduling: conoscendo la data di consegna e i tempi necessari per la realizzazione dei vari task necessari per il completamento dell'ordine si stabilisce la data di inizio produzione.

Dalla conoscenza di questi fattori si può scegliere la strategia di gestione degli ordini, ovvero decidere con che sequenza essi verranno lavorati.

1) FIFO: (first in first out) con questa strategia vengono lavorati e completati i lotti con lo stesso ordine con cui arrivano senza preoccuparsi della data di consegna. È una

strategia che genera un certo valore di tardiness poiché non è garantito che il primo prodotto in ingresso sia anche quello con una data di consegna più prossima.

2) EDD: (earliest due date) I lotti sono ordinati in base alla priorità sulla data di consegna. Con questa strategia il tempo totale di completamento dei lotti rimane lo stesso ma si riduce in maniera significativa il tardiness
3) Slack MST: i lotti sono sequenziati in ordine crescente del valore di slack. Il valore di slack è calcolato come la differenza tra la data di consegna e l'istante attuale a cui si sottrae ulteriormente la durata dell'operazione. Con questa strategia il tempo di completamento totale si riduce ma si avrà un aumento del tardiness rispetto all' EDD ma inferiore rispetto al caso FIFO
4) Critical Ratio: lotti classificati per valori crescenti del cirtical ratio. Il ciritcal ratio è calcolato come la differenza tra la data di consegna e l'istante attuale, il tutto diviso per il tempo di lavorazione. È una strategia poco utilizzata perché genera un aumento del tempo di completamento e valori di tardiness non ottimali.
5) Short Processing Time: lotti sequenziati con valori crescenti dei tempi di lavorazione richiesti. Questa è la strategia che genera un tempo complessivo di lavorazione più basso.

Esistono diversi casi in cui il sito produttivo di un'azienda può essere semplificato in un'unica macchina così da facilitare il sequenziamento dei lotti, questo avviene quando una macchina all'interno di un sistema ha una capacità produttiva più bassa rispetto alle altre e prenderà il nome di collo di bottiglia. Il collo di bottiglia è un punto importante all'interno di un sistema produttivo perché è la macchina o l'operazione che detta i ritmi produttivi alle macchine a monte; dunque, l'ottimizzazione del processo produttivo per un'azienda con più macchine può essere semplificato in un problema di schedulazione per la macchina collo di bottiglia.

C'è un altro aspetto da tenere in considerazione che non è stato affrontato, le strategie viste finora supponevano che i tempi di set-up per la produzione tra un articolo e l'altro fossero pari a zero o costanti da lotto a lotto. In molti casi i tempi necessari per effettuare l'attrezzaggio di una determinata macchina variano se ad esempio si passa dalla produzione del lotto X alla produzione del lotto Y o alla produzione del lotto Z. Il tempo di

completamento di tutti i lotti dipenderà anche dalla sequenza con cui essi verranno lavorati.

Supponiamo che in una macchina debbano essere lavorati 3 articoli (X,Y,Z) con i relativi tempi di changover per il passaggio della produzione da un lotto ad un altro:

- X -> Y = 20 minuti
- X -> Z = 15 minuti
- Y -> X = 25 minuti
- Y -> Z = 20 minuti
- Z -> X = 35 minuti
- Z -> Y 25 minuti

Per valutare qual è la sequenza che minimizza i tempi di set-up si utilizza la "changover wheel" dove, partendo da un articolo si andrà a sequenziare in successione l'articolo con tempo di changover inferiore. Ad esempio, se si decidesse di iniziare con la produzione dell'articolo X si deve seguire con il lotto Z e successivamente con l'articolo Y.

2.3. Flow-shop scheduling

Per sistemi flow-shop si intende il caso in cui si hanno un insieme di lotti che devono essere lavorati tutti su un numero finito di macchine e ognuna di esse è dedicata a fare una sola lavorazione.

Un flow-shop ha come caratteristica che ogni lavorazione richiede un dato tempo e non può essere interrotta, inoltre il flusso di lavorazione è unidirezionale e se un lotto deve essere lavorato prima su una macchina e poi su un'altra allora questo ordine di lavorazione dovrà essere rispettato per tutti i lotti. Naturalmente questa tipologia di produzione si presta bene solo per famiglie di prodotti aventi stessa sequenza di produzione, si utilizzano algoritmi di clustering come, ad esempio, quello di King per la ricerca delle famiglie di componenti.

Il problema di schedulazione per un sistema flow-shop consiste nel trovare la sequenza dei lotti che minimizzano il tempo di completamento di tutte le lavorazioni oppure il tempo di ritardo o ancora che massimizza il flow-time. Lo strumento più utilizzato per il sequenziamento è:

- Teorema di Johnson (caso di sequenziamento su 2 macchine)

  Dato il makespan di ogni lotto si andrà a mettere, come primo nella sequenza, il lotto che richiede il tempo di lavorazione minore sulla prima delle due macchine. Successivamente si andrà a sequenziare il lotto che ha il tempo di lavorazione minore su una delle due macchine. Se tale tempo è sulla prima macchina allora quel lotto sarà il secondo ad essere processato mentre se tale tempo è riferito alla seconda macchina vorrà dire che sarà l'ultimo ad essere lavorato. Si itera il ragionamento per tutti i lotti da processare.

Questo metodo è molto efficiente ma tuttavia è applicato solo nel caso di due macchine. Nel caso di lotti che devono essere lavorati su più macchine bisognerà ricondursi sempre al teorema di Johnson applicando i seguenti steps:

1) Da 4 macchine mi riconduco a 2 così da applicare il precedente teorema: la prima delle due macchine virtuali (M1*) avrà come tempi quelli della prima macchina reale (tra le 4) mentre la seconda (M2*) prenderà i tempi della quarta macchina reale. Abbiamo due macchine allora applichiamo Johnson normalmente.
2) Nel secondo sistema virtuale la prima macchina M1** avrà come tempi la somma dei tempi, per ogni lotto, presi dalle prime due macchine reali. M2** vedrà la stessa cosa ma con i tempi della terza e quarta macchina, dunque, applichiamo Johnson.
3) Nell'ultimo sistema virtuale la prima macchina virtuale che lo compone avrà i tempi dati dalla somma di quelli delle prime 3 macchine reali (M1,M2,M3) mentre la seconda lo stesso ma con le ultime 3 che dunque saranno M2,M3,M4. Otteniamo i tempi per M1*** e M2***: applichiamo Johnson.

Le 3 sequenze ottenute nelle 3 simulazioni devono essere uguali. Una volta individuata la sequenza corretta si procederà con il calcolo dei tempi di durata delle operazioni sulle diverse macchine utilizzando però i tempi reali e non quelli fittizi.

2.4. Job-shop scheduling.

Nei sistemi job-shop si avrà la necessità di individuare il giusto sequenziamento di un numero finito di lotti che dovranno essere lavorati su più macchine, la differenza con il caso precedente è che qui non si avrà un flusso unidirezionale delle lavorazioni ma ogni lotto

avrà una sequenza di lavorazione differente. È necessario caratterizzare ogni lavorazione con 3 indici, uno riferito al lotto, uno alla macchina e un altro all'operazione

Un metodo per lo scheduling dei lotti in un sistema job-shop prende il nome di:

- Work in next Queue (WINQ): si supponga di dover lavorare un numero di lotti che presentano la condizione tale per cui debbano essere lavorati tutti sulla stessa macchina per la prima operazione mentre le successive richiederanno l'impegno di una differente risorsa. Il modello schedula i lotti in base al tempo di attesa degli stessi sulla risorsa successiva alla presente, cioè saranno ordinati in modo tale che il primo sarà quello che alla lavorazione successiva attenderà in ingresso alla macchina un periodo di tempo più corto.

Tuttavia, questo non è l'unico metodo possibile, si possono preferire altre strategie di schedulazione come ad esempio:

- First in, first out (FIFO) ovvero il primo lotto in arrivo sarà il primo ad essere lavorato
- Fewset teamaining Operations (FROP)
- Slack per remaining operation (S/OPN): La priorità di un lavoro è determinata dividendo lo slack per il numero di operazioni che restano, compresa quella in programma.
- Least remaining work (LRWK)

2.4. Problemi di programmazione lineare

I problemi di schedulazione spesso sono descritti come modelli di programmazione lineare i quali sono modelli matematici che sono caratterizzati da una funzione obiettivo che rappresenta un'espressione lineare, da delle variabili decisionali e da delle equazioni o disequazione che fungono da vincolo alla funzione obiettivo. Nei seguenti due paragrafi verranno presentati due problemi molto diffusi.

2.4.1. Problema del carico macchina (Prima versione)

Il problema del carico macchina è un problema di programmazione lineare nella quale si ha come obiettivo quello di determinare il piano di produzione per realizzare le quantità stabilite dei lotti di produzione sulle macchine disponibili.

Noti:

- Produttività oraria del lotto i sulla macchina j pari a q_{ij}
- Ogni lotto va prodotto in quantità nota pari a P_i
- Ogni macchina ha a disposizione un tempo per effettuare le lavorazioni pari ad H_j

La variabile da tenere in considerazione per questo problema è il tempo t_{ij} che la macchina j dedica alla produzione del lotto i.

La funzione obiettivo in questa tipologia di problema è la funzione che minimizza la variabile critica t_{ij}.

- $\text{Min} \sum_i \sum_j t_{ij}$ la prima sommatoria va per i=1 fino ad n= numero di lotti da produrre, mentre la seconda sommatoria va da j=1 fino a m=numero di macchine

A tale funzione obiettivo vanno affiancate le equazioni di vincolo

1) $\sum_j q_{ij} * t_{ij} \geq P_i$ i = 1,2,….,n la quantità del lotto i lavorabile nella macchina j per il tempo in cui il lotto i è in lavorazione sulla macchina j deve essere maggiore o al massimo uguale alla produzione richiesta del lotto i. (ci sono tante equazione quanti sono i lotti)
2) $\sum_i t_{ij} \leq H_j$ j=1,2,….,m ovvero il tempo di lavorazione per la produzione dei vari lotti non deve superare il tempo disponibile.
3) $T_{ij} \geq 0$ per ogni i,j

		macchine			
		1	2	m	
lotti	1	q_{11}	q_{12}	q_{1m}	P_1
	2	q_{21}	q_{22}	q_{2m}	P_2
	n	q_{n1}	q_{n2}	q_{nm}	P_n
		H_1	H_2	H_m	

(Seconda versione)

In questa versione cade la possibilità di un lotto di essere lavorato su più macchine

Si ha a disposizione il tempo complessivo h_{ij} per la produzione completa del lotto i sulla macchina j, ogni macchina ha a disposizione un tempo H_j per effettuare le lavorazioni. In questo caso si vuole determinare il piano di produzione per realizzare tutti i lotti di produzione sulle macchine disponibili.

Si introduce una variabile x_{ij} binaria che indica se il lotto i viene o meno lavorato sulla macchina j; dunque, la funzione obiettivo sarà del tipo:

- Min $\sum_i \sum_j x_{ij} * h_{ij}$

Le equazioni di vincolo sono:

1) $\sum_j x_{ij} = 1$ i=1,2,….,n (ogni lotto viene lavorato su una sola macchina)
2) $\sum_i x_{ij} * h_{ij} \leq H_j$
3) X_{ij} binario per ogni i,j

		macchine		
	1	2	m	
1	h_{11}	h_{12}	h_{1m}	
2	h_{21}	h_{22}	h_{2m}	
lotti				
n	h_{n1}	h_{n2}	h_{nm}	
	H_1	H_2	H_m	

2.4.2. Problema dell'assegnamento

In questa tipologia di problema si ha un numero n di attività che devono essere svolte utilizzando un numero n di risorse per la loro esecuzione

Come input si ha il costo dell'esecuzione del task i-esimo da parte della risorsa j-esima, devono essere svolte tutte le attività e ogni risorsa è utilizzabile per l'esecuzione di una sola attività.

Anche qui si sceglierà l'uso di una variabile binaria x_{ij} che indica se l'attività i viene o meno svolta dalla risorsa j. La funzione obiettivo da minimizzare è una funzione di costo, ovvero si vuole trovare il costo minimo per lo svolgimento delle varie attività usando un numero finito di risorse

- Min $\sum_i \sum_j x_{ij} * c_{ij}$

Le equazioni di vincolo sono:

1) $\sum_j x_{ij} = 1$ i=1,2,….,n
2) $\sum_i x_{ij} = 1$ j=1,2,…,n
3) X_{ij} binaria per ogni i,j

attività \ risorse	1	2	n	
1	c_{11}	c_{12}	c_{1m}	1
2	c_{21}	c_{22}	c_{2m}	1
n	c_{n1}	c_{n2}	c_{nn}	1
	1	1	1	

2.5. SCHEDULAZIONE STOCASTICA

I modelli di programmazione lineare per la schedulazione dei sistemi produttivi assumono implicitamente delle ipotesi relative al numero di lotti, tempi di lavorazione, istante in cui i lotti sono disponibili per la lavorazione. Questi aspetti sono ipotizzati come valori noti che non variano, inoltre non si fa menzione a possibili guasti, manutenzione o difetti delle macchine ma bensì esse sono supposte come sempre disponibili lungo tutto l'orizzonte di pianificazione.

Si può estendere il ragionamento dicendo che nella realtà bisogna sempre tenere in considerazione una quota di incertezza e imprevedibilità.

Prendono il nome di sistemi stocastici i problemi di schedulazione in cui i parametri del modello non sono noti con certezza; tale incertezza viene descritta da distribuzioni di probabilità note (Es. Distribuzione BETA unimodale).

2.6. SCHEDULAZIONE DINAMICA

Nel caso in cui il sistema presenti degli elementi di incertezza tali per cui non si ha a disposizione informazioni sulla probabilità di accadimento degli eventi ma questi avvengono in maniera imprevista senza possibilità di prevederli allora si parlerà di schedulazione dinamica.

"Si definiscono dinamici tutti quei problemi di schedulazione che contemplano l'accadimento di eventi in tempo reale"

I modelli di schedulazione statistici in cui sono note e fisse le varie caratteristiche che compongono il modello e per la quale si riesce ad avere una soluzione ottimale anticipatamente non sono in grado di gestire eventi ed informazioni che giungono in tempo reale. Tali imprevisti determinano una variazione del piano di produzione rendendo quello precedente impraticabile o per lo meno non ottimale.

Questi eventi improvvisi nella realtà sono molto frequenti e possono essere di vario tipo:

- Guasto di una macchina
- Ordine di un lotto straordinario e urgente
- Assenteismo del personale
- Cambiamento della data di consegna da parte di un cliente
- Mancanza di materiale a causa di un ritardo da parte del fornitore
- Cancellazione di un ordine
- Ecc.

Si possono seguire diversi approcci per la schedulazione dinamica. Un primo approccio consiste nel non generare alcuna schedula anticipatamente ma prendere decisioni in tempo reale. Le regole di dispaccio consistono nel selezionare quale lotto lavorare tra un insieme di lotti in attesa che la macchina divenga nuovamente libera. Le regole di dispaccio più frequentemente utilizzate sono “Shortest Process Time (SPT)”, “Earliest Due Date (EDD)” , “First In First Out (FIFO)”, ecc. Questo è un approccio estremamente semplice in quanto non richiede grosse capacità di calcolo.

Un secondo approccio che viene utilizzato è quello di generare una schedula anticipatamente ma essa verrà revisionata e modificata volta per volta in risposta agli eventi che si manifestano.

2.7. SCHEDULAZIONE INTELLIGENTE

I concetti di schedulazione dinamica si sposano molto bene con le tecniche di intelligenza artificiale in particolar modo quando le decisioni da prendere al verificarsi di un evento improvviso sono molte.

I concetti di intelligenza artificiale si basano sul catturare le competenze e le esperienze passate in modo tale da poter prendere decisioni future sulla basa di situazione già affrontate. Si può far riscontro a tecniche di machine learning al fine di migliorare la prestazione della schedulazione che si basano sulle regole di dispaccio proprio perché tra tali regole non esiste una migliore dell'altra ma si possono scegliere in base alle esigenze specifiche del momento.

L'approccio del machine learning supervisionato si bassa sulla simulazione ed applicazione delle varie regole di dispaccio per una macchina a seconda dei vari casi e sulla base delle esperienze acquisite in passato.

Una volta individuato il tipo di problema bisognerà formulare il problema mediante modello matematico usando programmi informatici come ad esempio Phyton. Il modello infine viene risolto con un risolutore specifico in base alla categoria di appartenenza del problema di ottimizzazione.

3. ANALISI LETTERATURA

3.1. Hybrid of human learning optimization algrothm and particle swarm optimization algorithm with scheduling strategie for flexible job-shop scheduling problem.

Article di: Haojie Ding, Xingsheng Gu

In questo articolo viene trattato il problema di pianificazione flessibile (FJSP) con diversi tipi di algoritmi di ottimizzazione dello sciame di particelle (particle swarm optimization)

(PSO) i quali, negli ultimi dieci anni hanno portato a risultati superiori per i problemi di ottimizzazione. Tale sistema è stato implementato con l'algoritmo di ottimizzazione dell'apprendimento umano (human learning optimization) (HLO), risulta essere un tipo di algoritmo di apprendimento semplice e adattivo che ha contribuito a migliorare le prestazioni.

Lo studio propone un algoritmo ibrido HLO-PSO. Con la guida del HLO, la capacità di apprendimento individuale di ogni particella è ulteriormente migliorata sulla base del vantaggio esistente della decisione di azione collettiva. Tale sistema è stato implementato su diversi problemi e ne è risultato che l'algoritmo ibrido HLO-PSO può risolvere in maniera efficiente la maggior parte dei FJSP a obiettivo singolo. In questo particolare caso di studio si va a definire una funzione obiettivo da minimizzare, nello specifico, la minimizzazione del makespan è impostato come criterio oggettivo del FJSP.

Per la risoluzione di problemi di schedulazione ci sono diversi algoritmi ispirati alla natura , tali algoritmi meta-euristici sono progettati imitando le caratteristiche dell'evoluzione biologica, come ad esempio l'algoritmo genetico (GA) e la programmazione evolutiva (EP); alcuni di questi algoritmi invece sono progettati in base alle azioni sociali di semplici animali o insetti, come l'ottimizzazione delle colonie di formiche (ACO), la ricerca del cuculo (CS) e l'ottimizzazione dello sciame di particelle (PSO). Quest'ultimo algoritmo è stato soggetto di diversi studi che hanno portato ad un ampio utilizzo dello stesso, difatti tale approccio meta-euristico è più facile e semplice da implementare e viene usato per la risoluzione dei FJSP combinato però con altri algoritmi o criteri euristici.

L'introduzione della capacità di apprendimento e tecniche di IA e di deep learning hanno dato il via allo sviluppo di un approccio all'ottimizzazione adattativa dello sciame di particelle (APSO). Cerca le soluzioni ottimali sintonizzando in modo adattativo i parametri PSO con il problema.

Questo studio propone un algoritmo meta-euristico chiamato algoritmo HLO-PSO per cercare il makespan minimo nei problemi FJSP, ovvero si cerca di mettere insieme il comportamento di apprendimento esistente nella cognizione umana e le azioni sociali del mondo animale. L'adattabilità di ogni particella è responsabile dell'HLO e il PSO migliorato proposto è responsabile del sequenziamento delle operazioni. Tramite questi approcci si

andrà sia alla ricerca della sequenza ottima ma anche della scelta ottimale delle macchine per eseguire le singole operazioni.

Il FJSP può essere suddiviso in due categorie:

T-FJSP dove si tiene in considerazione come set di macchine utilizzabili per effettuare tutte le lavorazioni l'intero parco macchine

P-FJSP può essere considerato come un sotto insieme del T-FJSP in quanto il set di macchine da tenere in considerazione è un sotto insieme delle macchine realmente disponibili. Proprio questa seconda categoria è il principale soggetto di ricerca.

Di seguito (Figure 1) viene riportato la formulazione del problema:

$$min\ C_{max} = max\{E_{11}, \ldots, E_{1n_1}; E_{21}, \ldots, E_{2n_2}; \ldots; E_{n1}, \ldots, E_{nn_n}\} \quad (1)$$

Soggetto a:

$$S_{ij} + p_{ijk} \leqslant E_{ij} \quad (2)$$

$$E_{ij} \leqslant S_{i(j+1)} \ \ where \ \ j+1 \leqslant n_j \quad (3)$$

$$E_{in_i} \leqslant C_{max} \quad (4)$$

$$(S_{ij} + p_{ijk}) * \ y_{ijef} \leqslant S_{ef} \quad (5)$$

$$\sum_{k=1}^{m} x_{ijk} = 1 \quad (6)$$

Figure 1

Tra le ipotesi c'è che non esiste priorità tra i lavori e ogni lavoro è indipendente, tutte le macchine sono disponibili al tempo zero e nesso di essi può essere interrotto o cancellato. Ogni lavoro è costituito da una singola operazione o da un insieme di operazioni, le quali devono essere eseguite in un ordine predeterminato. Il tempo di trasporto da una macchina all'altra è trascurato e non vengono considerati i guasti nella formulazione del problema.

L' equazione (1) rappresenta la funzione obiettivo soggetta ai vincoli (2-6). Il vincolo (2) stabilisce che l'esecuzione non deve essere interrotta durante il processo; il vincolo (3) richiede che ogni operazione venga elaborata solo dopo che il suo predecessore è stato

completato; il vincolo (4) afferma che l'ora di fine di qualsiasi operazione non deve essere superiore al tempo di completamento massimo; il vincolo (5) limita che quando due operazioni consecutive devono essere assegnate alla stessa macchina il successore può essere elaborato solo dopo che il predecessore ha terminato; il vincolo (6) afferma che ogni operazione può essere eseguita una sola volta sulla macchina prescritta e non deve essere eseguita su nessun'altra macchina.

Algoritmo HLO

Per quanto riguarda l'uso delle tecniche HLO, nello studio vengono prese in considerazione tre attività di apprendimento che sono definite come apprendimento individuale (pi), apprendimento casuale (Pr) e apprendimento sociale (Ps) e sono soggette ai vincoli presenti nella figura 2:

$$\begin{cases} individual\ learning\ activity & if\ p_r \in (0, p_i) \\ free\ learning\ activity & if\ p_r \in [p_i, p_s] \\ social\ learning\ activity & if\ p_r \in (p_s, 1) \end{cases}$$

Figure 2

L'operatore di apprendimento individuale e l'operatore di apprendimento sociale delimitano il dominio di apprendimento.

Quando il valore di apprendimento casuale assume un valore compreso tra 0 e 1, l'apprendimento corrispondente dell'essere umano viene preso in base all'intervallo di caduta del valore casuale.

Algoritmo PSO

Il tradizionale algoritmo PSO si ispira al comportamento cooperativo degli uccelli in uno sciamo i quali si muovono in maniera casuale generando uno scambio di informazioni; nessuno di loro è più intelligente di un altro e il risultato di ogni azione individuale è deciso solo dall'azione collettiva dell'intero sciame. Per formulare il tradizionale problema PSO si presuppone che ogni uccello sia una particella e ogni sua posizione potrebbe

essere una possibile soluzione; quindi, ad ogni particella vengono assegnati attributi come la posizione attuale, la velocità nello spazio di ricerca e la migliore posizione individuale mai trovata dalla particella durante il processo di ricerca. Nel frattempo, la di tutte le migliori posizioni individuali è chiamata posizione ottimale globale. Il meccanismo di ottimizzazione del PSO tradizionale si basa sul fatto che ogni particella regola e ottimizza la sua posizione successiva prendendo informazioni sulla posizione ottimale globale corrente (processo di ricerca cooperativa). Per applicare il PSO all'FJSP si associa l'ordine di sequenza di tutte le operazioni al vettore di posizione della particella pertanto è possibile costruire una relazione uno-a-uno tra un elenco di operazioni e il vettore di posizione della particella.

In questo caso di studio, la capacità di apprendimento di ogni particella è promossa dal HLO e la direzione di ricerca di ogni particella viene percorsa prendendo una delle diverse combinazioni proposte dall'algoritmo. Con l'approccio HLO-PSO si adotta un nuovo schema di codifica per la pianificazione del FJSP in una catena. Tale catena presenta dei nodi con informazioni riguardanti l'operazione da eseguire e la macchina su cui farla (figura 3).

Lavoro	Operazione	M_1(Tempo di elaborazione)	M_2(Tempo di elaborazione)	M_3(Tempo di elaborazione)
1	1	3	2	–
	2	2	1	3
2	1	–	4	–
	2	–	–	5
	3	3	–	–
	4	–	4	3

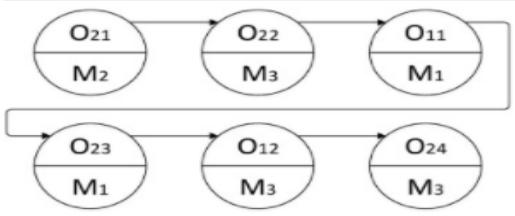


Figure 3

Dunque, una lista di operazioni è rappresentata come una lista concatenata e ogni nodo è costituito da due strutture dati, una che fa riferimento alle operazioni e una che fa riferimento alle macchine.

Con la tecnica del “puntatore” viene si creano un insieme di macchine candidate allo svolgimento delle operazioni.

Tramite processo di decodifica si stabilisce l’ora di inizio l’ora di fine di ogni attività, per individuare le operazioni critiche si esegue un processo di decodifica invertita il quale verrà avviata dall’ultimo nodo.

Nella fase di inizializzazione vengono introdotti i dati di istanza per costruire la lista delle macchine candidate di ogni operazione; nel frattempo, l’elenco delle operazioni originale viene creato con i dati dell’istanza in baso allo schema di codifica proposto e la macchina operativa predefinita di ogni nodo operativo. Quindi lo sciame di N particelle può essere costruito dando la posizione iniziale di ogni particella con una deformazione casuale dell’elenco operativo originale e bilanciando i carichi macchina iniziali di ogni particella. La deformazione casuale include anche il cambio di sequenza di operazioni e le alternanze di macchine candidate di alcune operazioni. Quando l’algoritmo HLO-PSO interviene nel processo iterativo viene creata una soluzione di pianificazione ottimale del problema con diverse combinazioni per i diversi intervalli; dunque, in base all’intervallo di tempo in cui un’attività si conclude si potrà scegliere una strada che porta ad un percorso ottimale.

Successivamente nell’articolo vengono descritti i vari parametri utilizzati nel caso di studio, i quali vengono poi implementati dall’algoritmo HLO-PSO; esso riesce a risolvere il FJSP in maniera efficiente solo nel caso di obiettivo singolo, la ricerca di soluzioni ottimali di FJSP molti-obiettivo è ancora motivo di ricerca.

3.2. Flexible job-shop scheduling with release dates, deadlines and sequence dependent setup times.

Di: Philipp Winklehner, Viktoria A. Hauder

Questo studio presenta un caso reale dove viene presentato il problema di programmazione operativa della produzione di un problema del settore automobilistico che produce pezzi stampati per iniezione. Si cerca di soddisfare i requisiti e vincoli prestabiliti come le risorse alternative, le date di rilascio, date di scadenza e i tempi di set-up dipendenti dalla sequenza. Viene proposto un approccio di programmazione a vincoli.

Il caso di studio è effettuato su un partner aziendale che lavora su 3 turni dove gli ordini di produzione sono composti da diverse operazioni, anche di manutenzione, tali task sono influenzati da date di rilascio, scadenza e rapporti di precedenza con lag temporali, tutte le operazioni richiedono tempi differenti e anche tempi di set-up dipendenti dalla sequenza con la quale essi si effettuano. Viene presentato un modello di programmazione dei vincoli (CP) fortemente basato sul ben funzionante CP Optimizer di IBM ILOG CPLEX (strumento per la risoluzione di problemi di ottimizzazione complessi) e sul lavoro correlato di Laborie er al.

Presentazione del problema

Lo studio nasce dall'esigenza di un partner aziendale con problemi di programmazione delle attività, in maniera dettagliata si parla di un'azienda produttrice di diversi articoli in ambito automobilistico ottenuto per stampa ad iniezione.

Un lotto o un ordine di produzione è descritto dalla quantità definita di un articolo specifico costituita da più fasi di lavoro $i,j \in \mathcal{I}$ dette anche attività che sono definite in una sequenza prescritta e comportano un'ultima fase di lavoro $i,j \in \mathcal{E} \subseteq \mathcal{I}$.

Tutti gli ordini vengono impartiti su base settimanale e devono essere programmati con cadenza oraria su tre turni ($t \in$ T) al giorno su macchine o risorse ($r \in \mathcal{R}$). L'obiettivo è la minimizzazione dei tempi di realizzazione di tutte le attività le quali sono soggette a una sequenza predefinita P_{ij} . la distanza temporale minima tra l'ora di inizio e quella di fine di due operazioni successive è specificata con uno sfasamento temporale Li. Data di rilascio

Fi e scadenza. D_i sono stabiliti per limitare l'assegnazione del tempo degli ordini. Le risorse per svolgere le attività sono rappresentate da S_{ir} che può associare una o più risorse per ogni attività. Tutte le attività sono soggette a tempi di elaborazione specifici del prodotto (Bi) e tempi di set-up che si verificano tra compiti successivi (Cij) la quale durata è dipendente e determinata dalla sequenza di lavorazione sulla risorsa individuale (q_{ir}). Una macchina è in grado di elaborare solo un'attività per volta (m_{ir}).

Nella prima tabella sono riportati gli ordini dove, ad esempio, l'ordine A13 è composto da 3 attività (A13/1, A13/2, ecc.) con i relativi tempi di esecuzione e la richiesta di una delle risorse alternative R1 o R3 (si fa riferimento all' attività A13/1). Tutti gli ordini hanno la stessa data di consegna (Monday 8am) e scadenza (Friday 6pm). Si definisce una maininstance A-M che richiede la risorsa R3, dopo la sua realizzazione, due intervalli temporali addizionali sono necessari prima che la risorsa sia nuovamente disponibile per processare nuovamente un ordine. Con l'inserimento obbligatorio di un tempo di preparazione di due ore all'inizio della settimana, l'orizzonte temporale di pianificazione complessivo è di 108 ore.

Ne risulta che sono presenti 8 differenti attività, 5 risorse e 9 possibili modelli risolutivi.

Attività che richiedono risorse diverse possono essere lavorate in parallelo

Table 1: Toy instance data set including all orders

Order	Activity	Processing time (h)	Release date	Deadline	Resources
A11	A11/1	40.00	Monday 8am	Friday 6pm	R1
A12	A12/1	19.44	Monday 8am	Friday 6pm	R2
A12	A12/2	4.44	Monday 8am	Friday 6pm	R5
A13	A13/1	50.01	Monday 8am	Friday 6pm	R1; R3
A13	A13/2	12.00	Monday 8am	Friday 6pm	R4
A13	A13/3	4.67	Monday 8am	Friday 6pm	R5
A14	A14/1	23.00	Monday 8am	Friday 6pm	R1
A-M	Maintenance	8.00	Monday 10am	Monday 6pm	R3

Table 1

È presenta anche una seconda tabella relativa ai tempi di changeover tra le varie attività.

Table 2: Toy instance changeover matrix for sequence dependent setup times

Activity	A11/1	A12/1	A12/2	A13/1	A13/2	A13/3	A14/1	Maintenance
A11/1	-	8	0	8	0	0	15	2
A12/1	8	-	0	8	0	0	8	2
A12/2	0	0	-	0	0	0	0	2
A13/1	8	8	0	-	0	0	8	2
A13/2	0	0	0	0	-	0	0	2
A13/3	0	0	0	0	0	-	0	2
A14/1	4	8	0	8	0	0	-	2
Maintenance	2	2	2	2	2	2	2	-

Table 2

Sulla base dei dati presenti nelle tabelle è possibile programmare manualmente le attività assegnando orario di inizio e fine, nonché risorse a tutte le task.

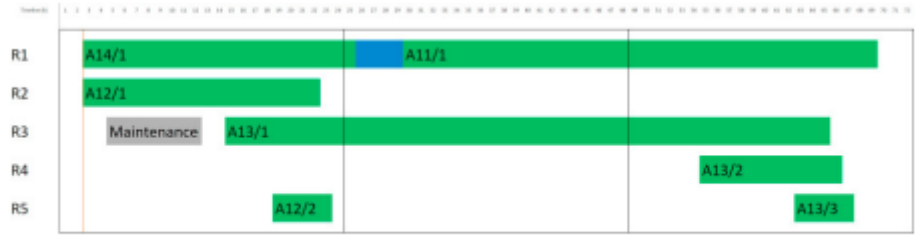


Figura 1: Programma esemplare

Tramite questa soluzione, si riesce a completare tutte le attività entro giovedì (tempo totale 72 ore)

Modello formale di programmazione

Di seguito viene riportato il modello matematico costruito per la ricerca della soluzione ottima al problema (figure 4).

$$\textit{Minimize} \qquad \sum_{i \in \mathcal{E}} \sum_{t \in \mathcal{T}} t * x_{it} \tag{1}$$

$$\textit{Subject to} \qquad P_{ij} * \sum_{t \in \mathcal{T}} (t - B_i + L_{ij}) * x_{it} \leq \sum_{t \in \mathcal{T}} (t - B_j) * x_{jt} \qquad \forall\, i,j \in \mathcal{I}, \tag{2}$$

$$\sum_{t \in \mathcal{T}} t * x_{it} \geq P_{ij} * \sum_{t \in \mathcal{T}} (t - B_j) * x_{jt} \qquad \forall\, i,j \in 1..I, \tag{3}$$

$$P_{ij} * \sum_{t \in \mathcal{T}} t * x_{it} \leq \sum_{t \in \mathcal{T}} (t-1) * x_{jt} \qquad \forall\, i,j \in \mathcal{I}, \tag{4}$$

$$\sum_{t \in \mathcal{T}} x_{it} = 1 \qquad \forall\, i \in \mathcal{I}, \tag{5}$$

$$\sum_{t \in \mathcal{T}} (t - B_i) * x_{it} \geq F_i \qquad \forall\, i \in \mathcal{I}, \tag{6}$$

$$\sum_{t \in \mathcal{T}} t * x_{it} \leq D_i \qquad \forall\, i \in \mathcal{I}, \tag{7}$$

$$\sum_{r \in \mathcal{R}} m_{ir} = 1 \qquad \forall\, i \in \mathcal{I}, \tag{8}$$

$$\sum_{r \in \mathcal{R}} S_{ir} * m_{ir} = 1 \qquad \forall\, i \in \mathcal{I}, \tag{9}$$

$$\sum_{i \in \mathcal{I}} \sum_{s=t}^{t+B_i-1} x_{is} * m_{ir} \leq 1 \qquad \forall\, r \in \mathcal{R}, t \in \mathcal{T}, \tag{10}$$

$$\sum_{t \in \mathcal{T}} t * x_{it} * m_{ir} + C_{ij} \leq \sum_{t \in \mathcal{T}} (t - B_j) * x_{jt} * m_{jr} + M * (1 - q_{ijr}) \qquad \forall\, i,j \in \mathcal{I}, r \in \mathcal{R}, \tag{11}$$

$$q_{ijr} + q_{jir} \leq 1 \qquad \forall\, i,j \in \mathcal{I}, i \neq j, r \in \mathcal{R}, \tag{12}$$

$$q_{ijr} + q_{jir} \geq m_{ir} + m_{jr} - 1 \qquad \forall\, i,j \in \mathcal{I}, i \neq j, r \in \mathcal{R}, \tag{13}$$

$$q_{ijr} \leq m_{ir} \qquad \forall\, i,j \in \mathcal{I}, r \in \mathcal{R}, \tag{14}$$

$$q_{ijr} \leq m_{jr} \qquad \forall\, i,j \in \mathcal{I}, r \in \mathcal{R}, \tag{15}$$

$$x_{it} \in \{0,1\} \qquad \forall\, i \in \mathcal{I}, t \in \mathcal{T}, \tag{16}$$

$$m_{ir} \in \{0,1\} \qquad \forall\, i \in \mathcal{I}, r \in \mathcal{R}, \tag{17}$$

$$q_{ijr} \in \{0,1\} \qquad \forall\, i,j \in \mathcal{I}, r \in \mathcal{R}. \tag{18}$$

Figure 4

La funzione obiettivo (1) minimizza il tempo complessivo di completamento di tutte le attività.

Con i vincoli (2)-(4) vengono determinate le relazioni temporali tra tutte le attività che possono essere lavorate in parallelo purché sia rispettata la sequenza. Ogni operazione deve essere programmata esattamente una volta, questo è stabilito nella relazione (5). Con le condizioni (6) e (7) il periodo di tempo per l'elaborazione dell'attività è fissato con date di rilascio e scadenze come vincoli rigidi, fornendo limiti temporali per il primo inizio e l'ultimo completamento. I vincoli (8) e (10) modellano gli aspetti relativi alle risorse.

Con i vincoli (11)-(15) si consente la rappresentazione dei tempi di configurazione dipendenti dalla sequenza che si verificano tra il set-up di due attività consecutive su una macchina.

Con i vincoli (16)-(18) le variabili decisionali sono determinate come valori booleani.

Approccio alla soluzione

L'approccio alla soluzione usando il modello FJc-RDSST è basato sulla formulazione CP permette di scrivere le varie equazioni sopra elencate in maniera più compatta e in modo tale da essere lette dal sistema software utilizzato.

Nella tabella si forniscono i risultati dell'ottimizzazione, la prima colonna mostra al relativa istanza, le colonne 2 e 3 mostrano le soluzioni del modello IP, la colonna 2 mostra il tempo di esecuzione in secondi e la colonna 3 fornisce il risultato dell'ottimizzazione (tempo di completamento totale), le colonne 4 e 5 presentano le soluzione del modello Cp e seguono la stessa logica.

È facile notare l'enorme differenza di prestazioni tra i due approcci di modellazione, per il modello IP è necessario circa un'quarto d'ora per la ricerca della soluzione ottima mentre per il modello CP sono sufficienti 0,04 secondi. La soluzione per il problema aziendale (ultima riga) mostra un dato eclatante, il modello IP non è in grado di trovare una soluzione ottima mentre con quello CP sono sufficienti 12 secondi per il calcolo della soluzione. Con questo si vuole dimostrare che il modello CP è superiore e molto più performante del modello IP questo perché il primo modello deve valutare solo 22 variabili decisionali contro le 1.850 del secondo

Table 3: Optimization results for the *FJc-RDSST*

	Integer Programming (=IP)		Constraint Programming (=CP)	
	Runtime	Solution	Runtime	Solution
Toy instance (j=8)	841.50	**196**	0.04	**196**
Enterprise instance (j=51)	T	-	12.00	**1,454**

Runtime in seconds; Solution…optimization result (bold letters indicate the optimal solution); T…time limit reached; "-"…no solution found; j…number of activities considered.

3.3. Flexible job scheduling problem with interval grey processing time.

Di: NaimingXie NanleiChen

Il seguente caso di studio nasce dall'osservazione relativa al fatto che il tempo di elaborazione del lavoro non è sempre indicato con precisione in un particolare processo di fabbricazione complesso. Si vuole studiare nuovi modelli e algoritmi basati sul tempo di elaborazione dell'incertezza in modo da risolvere i problemi di pianificazione dell'officina di lavoro. Questo modello ha l'obiettivo di elaborare un modello di pianificazione di lavoro dell'incertezza tendendo a ridurre al minimo l'incertezza di tempo per la realizzazione di attività (intervallo grigio).

Nella maggior parte delle situazioni reali i tempi per la realizzazione di un'attività non sono definiti con certezza ma si stabiliscono degli intervalli entro la quale c'è una possibilità percentuale di realizzazione del compito.

Il problema di schedulazione del job shop di prodotti complessi è un problema di scheduling flessibile del job shop (FJSP) proposto da Bunker e Schlie nel 1990. Negli ultimi anni sono stati adottati molti algoritmi euristici e metaeuristici per risolvere il problema FSJP come tabu serch, ottimizzazione dello sciamo di particelle (swarm intelligence), ecc. in particolare l'algoritmo genetico ha guadagnato molta attenzione ed è quello sviluppato nel caso in esame.

(Nel articolo vengono citati diversi studi riguardante il problema FJSP svolti da diversi studiosi)

A causa di molti nuovi prodotti, pezzi di sviluppo di prodotto complessi, possiamo solo stimare l'intervallo approssimato piuttosto che l'esatta distribuzione del tempo di elaborazione attraverso scarse informazioni dell'esperienza dei dipendenti o dati storici di

produzione. Diversamente dalla teoria fuzzy, la teoria dei sistemi grigi (GST) proposta da Deng conduce principalmente l'analisi del sistema, il processo decisionale e l'ottimizzazione sulla base del numero di Gray. Un numero grigio è definito come un numero di cui non si conosce il valore reale mentre è possibile definire i possibili valori impostati. È molto adatto per stimare i tempi di elaborazione delle attività di produzione. In particolare, è possibile definire i limiti inferiore e superiore del possibile insieme.

Questo documento mira a combinare il tempo di elaborazione del grigio e progettare un algoritmo genetico per risolvere questo problema di pianificazione con informazioni grigie.

Si parte dall'ipotesi per cui un'operazione può essere elaborata da almeno un'unità di lavoro opzionale nell'insieme delle unità di lavoro candidate, questo implica che il tempo di elaborazione dell'operazione non può essere definito esattamente ma può essere raccolto come un numero di intervallo grigio. Il modello in esame prende il nome di G-FJSP e presume che:

- Ogni lavoro non ha priorità
- Ogni unità di lavoro può avviare l'attività dal momento zero e non ha attività in quel momento ed è priva di tempo di riposo
- Una volta che l'operazione è iniziata non può essere interrotta

$J = \{ J_i \}$, i=1,2, …, n insiemi di n lavori da pianificare

$W = \{W_k\}$ k= 1,2, …,m insieme di tutte le m unità di lavoro candidate

O_{ih} si indica una delle operazioni del lavoro J_i

$W(O_{ih}) = \{W_e\}$ e= 1,2,..,l_{ih} è un insieme di l_{ih} unità di lavoro candidate di O

T è indicato come il tempo di elaborazione , intervallo grigio di O su W

S è indicato come l'ora di inizio dell' elaborazione dell'intervallo grigio di O

E indica il tempo di fine elaborazione dell'intervallo grigio di O

I è l'intervallo di tempo grigio

C_{max} indica il tempo massimo di completamento di grigio di tutti i lavori

Il problema è definito come segue (figure 5):

$$\mathrm{Min}C^{\otimes}_{max} = \max_{1\le i\le n}\left(E^{\otimes}_{i,q_i}\right)$$

$$s.t.\begin{cases} S^{\otimes}_{i,h} + x_{i,h,k} \times T^{\otimes}_{i,h,k} \le E^{\otimes}_{i,h} & (1.1)\\ E^{\otimes}_{i,h} + I^{\otimes}_{i,h+1} \le S^{\otimes}_{i,h+1} & (1.2)\\ S^{\otimes}_{i,h} + T^{\otimes}_{i,h,k} \le S^{\otimes}_{j,p} + L(1 - y_{i,h,j,p,k}) & (1.3)\\ \sum_{k=1}^{l_{i,h}} x_{i,h,k} = 1 & (1.4)\\ S^{\otimes}_{i,h} \ge 0 & (1.5)\\ i = 1,2,\cdots,n; j = 1,2,\cdots,n\\ h = 1,2,\cdots,q_i - 1; p = 1,2,\cdots,q_j\\ k \in W(O_{i,h}) \end{cases}$$

Figure 5

Le formule (1.1) e (1.2) rappresenta il vincolo di sequenza di ogni lavoro, la formula (1.3) indica che l'unità di lavoro può eseguire solo un compito alla volta, la formula (1.4) indica che le stesse operazioni possono essere elaborata solo da un'unita di lavoro contemporaneamente e la formula (1.5) mostra che l'ora di inizio di ogni lavoro non è non negativa e può iniziare al momento zero .

Lavoro	Operazione	Intervallo di tempo	Unità di lavoro		
			W_1	W_2	W_3
1	1	0	[6,8]	[2,5]	[7,8]
	2	[5,9]	–	[4,8]	[4,8]
	3	[5,8]	[7,10]	[6,8]	–
2	1	0	[3,4]	[8,11]	[6,7]
	2	[6,9]	[6,8]	–	[2,5]
	3	[6,7]	[7,11]	[6,7]	–
3	1	0	[2,3]	[7,8]	[2,4]
4	1	0	–	[3,7]	–
	2	[6,8]	[6,10]	–	[8,12]

Figure 6

In figura 6 è riportata la tabella relativa i lavori da schedulare, si nota ad esempio che la prima operazione del lavoro 1 ha tre unità di lavoro alternative e la seconda non può

essere eseguita da W1, ogni operazione su ciascuna unità di lavorazione è rappresentata dall'intervallo grigio, ovvero l'intervallo entro la quale quell'attività viene svolta; inoltre in ogni riga dell'operazione viene registrato l'intervallo di tempo di elaborazione del grigio tra un'operazione e l'operazione precedente.

Viene anche visualizzato il relativo diagramma di Gantt (figure7)

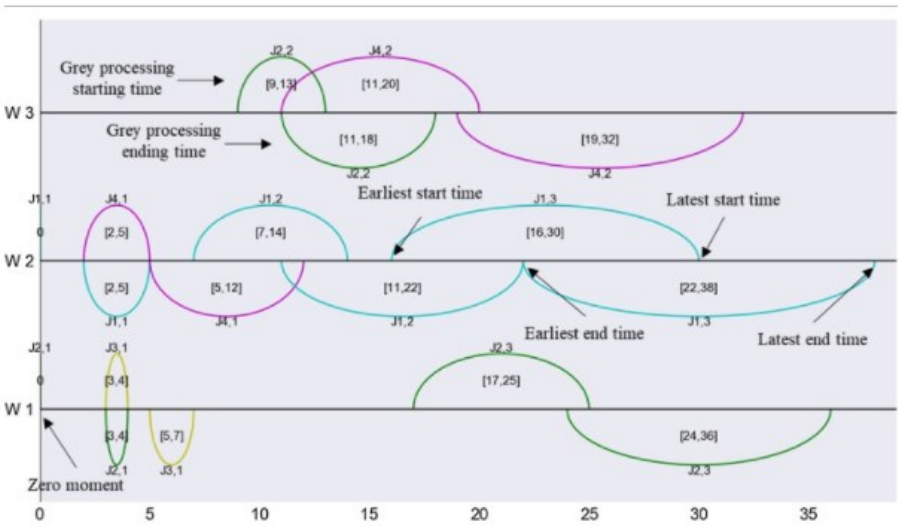


Figure 7

In un diagramma di Gantt grigio ogni unità di lavoro ha la sua linea temporale di elaborazione per rappresentare la sequenza di elaborazione.

L'algoritmo genetico (GA) è la tecnica più popolare per risolvere il FJSP dal 1990. Zhang ha proposto un algoritmo genetico ibrido per un problema FJSP multi-obiettivo, in particolare è stata adottata una memoria esterna per migliorare l'efficienza del GA salvando e aggiornando le soluzioni migliori durante il processo di ottimizzazione. Il caso di studio, quindi, mostra un caso di G-FJSP con algoritmo genetico basato sulla memoria esterna con strategia di elitismo (EGA).

Il caso di studio propone dunque uno scenario applicativo EGA per G-FJSP e sono stati progettati 9 casi simulati per testare l'algoritmo. Gli esperimenti mostrano che l'algoritmo proposto è efficace e con un'elevata velocità di ricerca della soluzione ottimale in diverse dimensioni dei dati e una riduzione notevole dei tempi di grigio. Tuttavia, è sempre difficile ottenere un valore di tempo preciso nella fabbricazione di prodotti complessi e

nella produzione personalizzata quindi risulta essere di fondamentale importanza definire degli intervalli stretti per il completamento delle diverse attività.

3.4. Hybrid teaching and learning-based optimization algorithm for distributed sand casting job-shop scheduling problem.

Di: Hongtao Tang, Bo Fang, Rong Liu, Yibing li, Shunsheng Guo

Questo articolo ha l'obiettivo di studiare un'problema reale di un'officina di produzione per fonderia che è passata dalla produzione in un unico sito a più impianti di produzione quindi si cerca di studiare il problema della pianificazione distribuita in officina flessibile (DFJSP) e si propone un modello di ottimizzazione del problema. Questo documento propone un algoritmo ibrido basato sull'insegnamento-apprendimento (HTLBO) che prevede una soluzione di codifica a tre livelli. L'HTLBO è costituito dalla fase di teacher learning, dalla fase di teaching e dalla fase di learning.

Il DFJSP si basa sulla produzione cooperativa di diverse aziende o sulla produzione distribuita di diversi stabilimenti nella stessa azienda e si concentra sulla distribuzione dei lavori tra le fabbriche e sulla sequenza di lavorazione in ciascuna fabbrica per ottimizzare gli indicatori di schedulazione. Rispetto alla pianificazione job-shop di una singola fabbrica il DFJSP prevede la selezione di fabbriche o unità di produzione adatte per ciascun lavoro da elaborare e l'assegnazione dei lavori a fabbriche diverse per generare un programma di produzione distinto.

Questo problema risulta ancora più complesso nei processi di fonderia dato che questi ultimi sono costituite da nu elevato numero di operazioni e macchine di lavorazione. Ogni azienda dispone di macchine diverse per ogni processo di colata il che rende difficile la ricerca di dati standard. Un ulteriore problema è dato dallo scarso livello di automazione dei processi il che implica alcuni vincoli pratici come il tempo di elaborazione iniziale, il

tempo di trasporto dell'prodotto; tutti aspetti che complicano la modellazione matematica del DFJSP.

Ad esempio, un lavoro effettuato sulla fabbrica 1 richiede 5 operazione mentre lo stesso lavoro effettuato sulla fabbrica 2 ne richiede 4. Il DFJSP comporta i seguenti 3 sotto problemi:

- Determinazione della fabbrica che elabora il lavoro
- Organizzare la sequenza di lavorazione appropriata per il lavoro assegnato alla stessa fabbrica
- Selezionare macchine di lavorazione per ogni operazione assegnata al lavoro della stessa fabbrica

Il modello proposta considera il vincolo del tempo di trasporto, il vincolo di tempo per il completamento delle operazioni ausiliari prima del processo e il vincolo di tempo per l'avvio della lavorazione. Viene proposto un nuovo algoritmo di ottimizzazione basato sull'insegnamento-apprendimento ibrido (HTLBO). L'algoritmo proposto viene confrontato con sei algoritmi intelligenti per valutarne la reale efficienza.

il problema è descritto da un insieme di lavori $J = \{J_1, J_2, \ldots J_n\}$ che deve essere lavorato in un insieme di stabilimenti di colata F = F_1, F_2, …. , F_q dove ogni sito produttivo è dotato di una serie di macchine M_{lk}= M_{11} , …. ,M_{1m} ,…, M_{q1} ,…, M_{qm}. Ogni lavoro può essere effettuata su entrambi i siti produttivi ma con tempi e numero di operazioni diversi quindi bisognerà assegnare la fabbrica più adatta per ogni lavoro, ordinare le operazione a causa dei diversi percorsi e inoltre bisognerà selezionare la macchina con la massima efficienza di elaborazione per ogni processi di ogni lavoro assegnato alla fabbrica.

			F_1					F_2				
		E	M_1	M_2	M_3	M_4	M_5	M_1	M_2	M_3	M_4	M_5
J_1	O_{11}		12	10	—	—	10	10	11	9	—	—
	O_{12}		8	6	—	6	—	—	—	6	8	7
	O_{13}	$[20,6^{+1}]$	—	8	6	9	—	7	—	8	—	9
	O_{14}		3	3	—	5	4	8	9	9	—	10
	O_{15}		8	6	6	—	7	—	—	—	—	—
J_2	O_{21}		—	11	—	11	10	13	—	10	11	—
	O_{22}		9	7	8	—	7	6	—	8	7	8
	O_{23}	$[20,6^{+1}]$	8	7	—	9	8	6	5	8	4	5
	O_{24}		—	—	—	—	—	5	5	4	6	—
J_3	O_{31}		12	11	13	11	10	—	12	10	11	—
	O_{32}		7	—	8	7	8	7	6	8	—	6
	O_{33}	$[20,6^{+1}]$	7	8	8	6	—	—	8	6	5	6
	O_{34}		6	—	4	5	5	5	—	5	6	4

Figure 8

Esistono vincoli sull'ordine di elaborazione per le operazioni dello stesso lavoro nello stesso sito e inoltre le lavorazioni devono proseguire senza interruzioni (no immagazzinamenti intermedi). Si parte dall'ipotesi che tutte le macchine in tutte le fabbriche sono disponibili all'ora0 e per un'operazione, una macchina può elaborare solo un lavoro alla volta e ogni task può iniziare solo dopo il completamento dell'operazione precedente. Non si tiene conto del tempo di trasporto da una macchina all'altra. Si ipotizza inoltre tempi fissi per la realizzazione delle singole attività.

Nel caso di studio reale è specificato che ogni lavoro ha rispettivamente 10 e 13 operazioni nel primo e nel secondo stabilimento; inoltre, ci sono 34 macchine nel primo sito produttivo e 40 nel secondo per l'elaborazione. Ogni operazione ha dei tempi di set-up noti e si conosce anche il tempo necessario per il trasporto del prodotto in lavorazione da una macchina all'altra. Proprio il vincolo legato alla presenza di due stabilimenti di produzione capaci di elaborare ogni tipologia di lavoro complica in maniera significativa la stesura del problema lineare, il quale ha sempre l'obiettivo di trovare l'ottima sequenza di produzione che minimizzi i tempi. Nell'articolo vengono anche riportati numerosi algoritmi riguardanti la capacità di istruzione e di apprendimento dell'algoritmo estremamente specifici per il caso in esame che risulta essere di particolare difficoltà a

causa delle numerose lavorazioni e degli ancora più elevati vincoli interconnessi tra di loro.

Poiché i parametri dell'algrotimo HTLBO possono influenzare le prestazioni dell'algoritmo, in questo documento viene adottato il metodo Taguchi per regolare i parametri

Sulla base dell'algoritmo HTLBO proposto, la soluzione ottimale al DFJSP è illustrata dal seguente diagramma di Gantt. I risultati ottenuti sono stati confrontati con i risultati provenienti da 6 diversi algoritmi di ottimizzazione, tra cui l'algoritmo di ottimizzazione del Particle Swarm e l'algoritmo genetico discusso nel caso precedente e si evince che la soluzione ricavata dall'algoritmo HTBLO fornisce ottime prestazioni nel risolvere il problema in esame

Figure 9

Tuttavia, nell'articolo viene evidenziato come tale modello non prenda in considerazione delle problematiche tipiche dei processi di fonderia quali ad esempio l'incertezza legata a presunte modifiche degli ordini, guasti casuali delle macchine e manutenzione delle stesse.

4. REALIZZAZIONE DI ALGORITMI PER LA SCHEDULAZIONE IN PYTHON

L'obiettivo di questa trattazione è quello di sviluppare un algoritmo di ottimizzazione per lo scheduling degli ordini di produzione tramite l'ausilio del software Python.

Il primo passo per la realizzazione di tale algoritmo è stato quello di ricercare una libreria che si presentasse bene per l'obiettivo. Tra le varie opzioni disponibili, la libreria Pyomo è stata quella che si prestava meglio

4.1. Introduzione

È stato scelto come caso di studio su cui testare poi l'algoritmo una configurazione di ordini inventata che prevedeva la presenza di:

- 6 articoli differenti;
- 2 macchine di lavorazioni;
- 2 differenti codici cliente;

I vari articoli hanno una sequenza e dei tempi di lavorazione differenti sulle diverse macchine (configurazione job-shop) in particolar modo si ha che:

- L'articolo 1 deve subire una lavorazione sulla macchina 1 e nessuna sulla macchina due;
- L'articolo 2 e l'articolo 4 devono essere lavorati prima sulla macchina uno e solo quando tale operazione è conclusa possono essere lavorate sulla macchina due;
- L'articolo 3 e articolo 6 devono essere lavorati prima sulla macchina due e solo quando tale operazione è conclusa possono essere lavorato sulla macchina uno;

Inoltre, è stata posta l'ipotesi che una macchina possa lavorare un solo lotto e che questo possa essere lavorato unicamente su una macchina nello stesso istante temporale.

Durante la stesura del codice, tuttavia, Pyomo non è stata l'unica libreria utilizzata ma ce ne sono state diverse di fondamentale importanza.

```
1   import matplotlib.pyplot as plt
2   import matplotlib as mpl
3   import pandas as pd
4   from pyomo.environ import *
5   from pyomo.gdp import *
6   from openpyxl import load_workbook
```

Codice. 1

Matplotlib è una libreria che permette di creare grafici e nel caso d'interesse il Gantt finale, verrà approfondito l'uso di Matplotlib nei paragrafi successivi.

Si è deciso di utilizzare anche la vastissima libreria di Pandas ma solo per poter lavorare in maniera più chiara con le matrici in modo da evitare lunghe righe di codice e semplificare così di molto il lavoro

La libreria openpyxl serve per poter leggere o scrivere file di tipo .xls molto utile per poter prendere informazioni da un foglio di calcolo.

4.2. Input

Molto importante per la realizzazione dello schedulatore tramite la libreria Pyomo è l'input che esso riceve e come esso si presenta.

L'input è un foglio di calcolo Excel dove sono presenti diversi aspetti che poi verranno letti e utilizzati dal software.

In un primo foglio sono presenti dati relativi alle operazioni da eseguire in particolar modo esso si presenta nella seguente forma:

Task (job,machine,cod)	Duration	Prerequisite Task	Domanda
('articolo1','M1','xx1')	5	None	50
('articolo2','M1','xx1')	6	None	14
('articolo2','M2','xx1')	10	('articolo2','M1','xx1')	14
('articolo3','M1','xx1')	8	('articolo3','M2','xx1')	25
('articolo3','M2','xx1')	10	None	25
('articolo4','M1','xx1')	9	None	30
('articolo4','M2','xx1')	20	('articolo4','M1','xx1')	30
('articolo2','M1','xx2')	6	None	33
('articolo2','M2','xx2')	10	('articolo2','M1','xx2')	33
('articolo6','M1','xx2')	11	('articolo6','M2','xx2')	10

('articolo6','M2','xx2')	4	None	10

Nella prima colonna è riportato il nome dell'operazione da eseguire. Si nota come il nome dell'attività è caratterizzato da 3 indici, il primo riferito al nome dell'articolo, il secondo riferito al nome della macchina dove l'articolo deve essere lavorato e il terzo riferito al codice del cliente.

Nella seconda colonna sono riportati i tempi unitari di lavorazione mentre nella terza colonna sono riportati eventuali vincoli di precedenza delle attività; il termine 'None' indica che l'attività non ha vincoli di precedenza. La quarta colonna indica la domanda per quel dato articolo.

Per poter leggere ed utilizzare tali informazioni in Python è stata creata una funzione apposita che riorganizzate il contenuto di tale foglio in modo tale da poterlo rendere compatibile con l'input del modello creato successivamente con Pyomo.

```
2   def lett_file():
3       wb = load_workbook(filename='input.xlsx', read_only=True)
4       ws = wb['Operazioni']   #lettura del foglio
5       #serie di liste vuote dove andare ad inserire i valori delle relative celle
6       name_task=[]
7       time_task=[]
8       order_task=[]
9       domanda_task=[]
10      cod_task=[]
11      #liste necessarie per inserire il valore della cella e poi modificarlo o
12      #spostarlo secondo le necessità
13      a=[]
14      b=[]
```

Codice. 2

Si definisce 'lett_file' la funzione per leggere i dati contenuti dentro ad un file Excel. Nella riga 3 si dà il comando per aprire e leggere il file Excel semplicemente digitando il nome del file stesso, con la riga 4 invece si sceglierà quale foglio del file andare ad interrogare. Con le due liste a= [] e b = [] si intende creare due liste inizialmente vuote dove andare ad inserire il valore della cella che si sta leggendo solo momentaneamente per poi poter andare ad inserire tale valore nella rispettiva lista.

```
15    for row in ws.rows:
16        for cell in row:
17            a.append(cell.value)
```

Codice. 3

con il primo ciclo for si va ad iterare su tutte le righe presenti nel foglio Excel mentre con il secondo for si andrà ad interrogare ogni cella della riga. Con il comando .append() si andrà ad inserire il valore della cella all'interno della lista vuota a[].

```
19        name_task.append(a[0])
20        time_task.append(a[1])
21        order_task.append(a[2])
22        domanda_task.append(a[3])
23        a=[]
```

Codice. 4

Così facendo si saprà già che nella prima posizione della lista troveremo sempre il valore della prima colonna, nella seconda posizione quello della seconda colonna e così via. Dato che la struttura della matrice da leggere è sempre la stessa allora si saprà che nella posizione 1 troveremo una terna di valori corrispondenti a (nome articolo, macchina, codice cliente), nella seconda colonna troveremo il valore riferito al tempo, nella terza colonna informazioni riguardanti eventuali vincoli di precedenza e nella quarta colonna i dati relativi alla domanda del cliente. Infine, si riporta la lista 'a' a lista vuota con l'ultimo comando.

Una volta interrogate tutte le celle in una riga si potrà passare alla riga successiva fino ad iterare il processo per tutte le righe disponibili.

Questa strategia presenta un piccolo problema legato al fatto che nelle liste relative ai nomi, ai tempi, alle precedenze e alla domanda il primo termine è il titolo della colonna dato che dovrà essere tolto e per farlo si procede nel seguente modo:

```
26    name_task.pop(0)
27    time_task.pop(0)
28    order_task.pop(0)
29    domanda_task.pop(0)
```

Codice. 5

successivamente si crea un dizionario vuoto che sarà poi l'input per il modello Pyomo e prende il nome di tasks:

```
37      tasks={}
38      for i in range(len(name_task)):
39          product_machine = eval(name_task[i])
40          name_task[i]= product_machine
41          cod_task.append(name_task[i][2])
42          time_task[i]= int(time_task[i])
43          domanda_task[i] = int (domanda_task[i])
44          time_task[i] = time_task[i] * domanda_task[i]
45          if order_task[i] != 'None':
46              order_prec = eval(order_task[i])
47              order_task[i] = order_prec
48          elif order_task[i] == 'None':
49              # order_task.pop(i)
50              nope= None
51              order_task[i] = nope
52          tasks [name_task[i]] = {"dur":time_task[i], "prec":order_task[i], "tsetup":{}}
53          ct = tuple(set(cod_task))
54      return(tasks , ct)
```

Codice. 6

per riempire tale dizionario si procede nel seguente modo:

Si utilizza un ciclo for che itera per un numero di volte pari alla lunghezza delle liste precedentemente create (se i punti precedenti fossero stati fatti correttamente tutte le liste dovrebbero avere la stessa lunghezza).

Dato che la lettura della cella, in special modo per quanto riguarda i valori contenuti nella prima colonna, restituisce come valore una stringa, come prima cosa bisognerà modificare tale stringa in una tupla in modo da poterla interrogare nei diversi indici. Questa operazione viene eseguita dalle righe 39 e 40.

Con il termine cod_task (riga 41) si vuole andare ad estrarre i diversi codici cliente contenuti all'interno dei vari nomi delle operazioni che ricordiamo sono caratterizzati da 3 indici, uno relativo al nome dell'articolo, uno al nome della macchina e il terzo relativo al codice cliente.

Successivamente bisogna considerare che il tempo unitario dell'operazione è irrilevante ai fini della schedulazione ma serve sapere con precisione il tempo necessario per realizzare il lotto del cliente, quindi, bisognerà moltiplicare il valore contenuto nella lista 'time_task' per il rispettivo valore nella stessa posizione contenuto nella lista 'domanda_task'.

Il ciclo if (riga 45) presente all'interno del ciclo for serve per gestire quelle righe nella quale l'attività in esame ha una precedenza con un'altra attività e allora il valore della cella sotto la colonna 'prerequiste task' sarà diverso da 'None' ma sarà a sua volta una terna di valori

e quindi anche in questo caso non va bene che la lettura di tale cella sia una stringa ma deve essere una tupla; viceversa qual ora il termine della colonna sia 'None' è necessario che Python lo legga come un NoneType e non come stringa e questo caso particolare viene gestito dalla riga 48 fino alla riga 51 con un ciclo 'elif '.

Infine, si va a creare il valore all'interno del dizionario 'tasks' in modo tale che il valore sia pari alla prima colonna del foglio Excel e quindi pari alla tupla contenente nome dell'articolo, nome della macchina e codice cliente. Come valore gli si passerà una serie di coppie chiave-valore in modo tale da avere una configurazione del tipo:

```
('articolo1', 'M1', 'xx1'):{'dur': 250, 'prec': None, 'tsetup': {}}
```

La voce relativa ai tempi di set-up verrà gestita in un'altra porzione del codice.

Nella riga 53 'ct' è una tupla che contiene tutti i codici cliente presenti nel foglio e sarà importante nella fase successiva.

Dunque, l'output di tale funzione è quella di restituire il dizionario 'tasks' che sarà l'input per la modellazione del problema e la tupla 'ct'.

4.3. Matrice dei tempi di set-up

Nelle realtà aziendali non esiste un unico tempo di set-up che caratterizza una singola macchina ma varia da articolo ad articolo. In alcuni casi invece, molto frequente nel settore alimentare, per passare dalla produzione di un prodotto A alla produzione di un prodotto B si ha un dato tempo di set-up per pulire e preparare le macchine mentre si ha un altro tempo di set-up se si passa dall'articolo B all'articolo A.

Per rappresentare questo scenario è stato utile introdurre le matrici di set-up dove vengono riportati i tempi per passare dalla produzione di un articolo all'altro e viceversa. È necessario scrivere tante matrici quante sono le macchine che vengono utilizzate quindi nel caso di studio in esame si farà riferimento a due matrici: una per la macchina M1 e l'altra per la macchina M2.

A/DA	('articolo1','M1')	('articolo2','M1')	('articolo3','M1')	('articolo4','M1')	('articolo6','M1')
('articolo1','M1')	0	4	3	6	3
('articolo2','M1')	1	0	4	6	3
('articolo3','M1')	2	4	0	4	5

('articolo4','M1')	7	5	3	0	2
('articolo6','M1')	2	1	1	1	0

(matrice dei tempi di set-up per la macchina M1)

A/DA	('articolo2','M2')	('articolo3','M2')	('articolo4','M2')	('articolo6','M2')
('articolo2','M2')	0	4	6	3
('articolo3','M2')	4	0	4	5
('articolo4','M2')	5	3	0	2
('articolo6','M2')	1	1	1	0

(matrice dei tempi di set-up per la macchina M2)

Per semplicità di lettura in Python le matrici dei tempi di set-up non sono matrice DA/A ma sono matrici A/DA.

La funzione che si crea prende il nome di 'lett_setup' (codice_6) e deve avere come input il nome del file Excel e del foglio dove sono contenute tali matrici, 'ct' è l'insieme dei vari codici cliente contenuti del foglio delle operazioni, è un valore ottenuto nel punto precedente e 'task' è il nome dell'input che poi verrà utilizzato all'interno del modello.

Lo step successivo è quello di riuscire a portare tali matrici all'interno dell'algoritmo in Python e successivamente riportarlo all'interno dell'input per la realizzazione del modello matematico tramite la libreria Pyomo.

```
7   def lett_setup(nome_file,nome_foglio,ct,task):
8       a=[]
9       b=[]
10      name_art=[]
11      name_art2=[]
```

Codice. 7

Così come è stato fatto anche per leggere i valori dal foglio contenente le varie operazione, allo stesso modo per leggere i vari valori nelle celle della matrice dei set-up si procede con un duplice ciclo for; il primo itera per tutte le righe mentre il secondo accede a tutte le celle contenute in una riga.

```
12      #creazione dizionario con le chiavi giuste
13      for row in nome_foglio.rows:
14          for cell in row:
15              a.append(cell.value)
16          name_art.append(a[0])
17          a=[]
18      name_art.pop(0) #in questo modo salvo tutti i nomi degli articoli all'interno di una lista
```

Codice. 8

Il valore della cella verrà inserito all'interno di una lista vuota che prende il nome di 'a', tale lista avrà una lunghezza pari alla lunghezza della riga nella matrice dove il primo valore è la coppia nome-macchina dell'articolo da realizzare come si può facilmente vedere nelle matrici sopra riportate. Tale valore verrà copiato in una seconda lista che prende il nome di 'name_art' e conterrà tutti i valori riportati nella prima colonna. Tuttavia nella prima colonna c'è un valore che deve essere eliminato ed è quello nella posizione (1,1) della matrice che ha come valore 'A/DA' e quindi tramite il comando .pop (riga 18) viene eliminato dalla lista.

```
20      matrix_setup={}
21      #creo il dizionario per i tempi di set-up
22      for i in range(len(name_art)):
23          product_machine = eval(name_art[i])
24          name_art[i]= product_machine
25          #serve per trasformare in stringa il nome in modo da poterci lavorare
26          ct= list (ct)
27          for j in range(len(ct)):
28              c= list(name_art[i])
29              c.append(ct[j])
30              c = tuple(c)
31              for keys in task:
32                  #serve per vedere se il valore di c esiste all'interno di task
33                  if keys == c:
34                      name_art2.append(c)
35                      matrix_setup[c] = {}
```

Codice. 9

Segue poi la creazione della matrice e per farlo si utilizza un primo ciclo for che itera per la lunghezza della lista name_art precedentemente creata. Come primo step bisogna cambiare la tipologia dell'informazione contenuta in ogni posizione della lista, questo perché quando si attinge il valore della matrice sul foglio di calcolo la coppia (nome, macchina) non viene letta come tupla ma come stringa; con il comando eval (riga 23) si forza il passaggio del valore da stringa a tupla.

Si noti come i nomi delle operazioni nel foglio contenente i valori dei tempi di setup sono indicizzati da 2 parametri che sono il nome dell'articolo e il nome della macchina diversamente dal caso precedente dove veniva riportato anche il codice cliente.

In questa fase (righe 27-30) il codice cliente viene inserito tramite il valore 'ct' che, come riportato precedentemente, è uno degli output della funzione 'lett-file' precedentemente descritta. Bisogna però ricordare che l'output è una tupla e quindi non accessibile o modificabile e quindi come primo step bisogna riportare il valore di 'ct' a lista.

Con il secondo ciclo for (riga 31) che itera per la lunghezza della lista 'ct' si andrà ad aggiungere ad ogni valore della lista 'name_art' il codice cliente così da avere un nome dell'operazione uguale alla funzione precedente e si crea così la lista 'name_art2' che si differenzia dal precedente perché ogni valore all'interno ha anche il codice cliente.

Infine, il terzo ciclo for seguito da un ciclo if serve per valutare l'esistenza della terna di valori precedentemente creata all'interno del dizionario 'task' creato nella funzione 'lett_file'.

Così facendo si inizia a creare la matrice dei setup che avrà una struttura del tipo:

```
matrix_setup = {('articolo1', 'M1', 'xx1'): {}, ('articolo2', 'M1', 'xx1'): {}, ('articolo2', 'M1', 'xx2'): {}, ('articolo3', 'M1', 'xx1'): {}, ('articolo4', 'M1', 'xx1'): {}, ('articolo6', 'M1', 'xx2'): {}}.
```

È un dizionario dove all'interno verranno riportati i nomi delle operazioni che a loro volta saranno dei dizionari dove verranno inseriti i vari tempi di setup riferiti a tutte le possibili combinazioni per passare da un articolo all'altro.

Per inserire i valori dentro i sotto dizionari bisognerà ripercorrere il codice nuovamente andando a leggere i valori dentro la matrice e per farlo si utilizzerà un doppio ciclo for che itera per le righe e per le celle di ogni riga.

```
38      #riempimento dei sotto dizionari
39      for row in nome_foglio.rows:
40          for cell in row:
41              b.append(cell.value)
42          if b[0] == 'A/DA':
43              for i in range(len(b)):
44                  b.pop(0)
45
46          if len(b)== len(name_art2) :
47              b[0]= eval(b[0])
48              for i in range(len(b)):
49                  if len(b) == len(name_art2):
50                      next_one = b[0]
51                      imp = []
```

Codice. 10

Il ciclo if (riga 42) serve per andare ad eliminare la prima riga dove non sono contenuti dati numerici relativi ai tempi di setup

Si realizza poi un ciclo for (riga48) che itera per la lunghezza di 'b' ovvero per la lunghezza della riga nella matrice dei setup e se tale lunghezza è pari a 'name_art2' precedentemente creata allora con il termine 'next_one' si andrà a intendere il valore nella posizione zero di 'b' che coincide con il nome dell'operazione da realizzare scritta come (nome articolo, macchina).

All'interno del ciclo if segue poi:

```
52                      for h in range(len (ct)):
53                          next_one = list(next_one)
54                          next_one.append(ct[h])
55                          next_one = tuple(next_one)
```

Codice. 11

Un ciclo for (riga 52) che itera per la lunghezza di 'ct' che si ricorda essere una lista contenente i codici cliente. Questo ciclo for serve per modificare il valore 'next_one' da quello precedentemente descritto ad una forma del tipo (nome articolo, macchina, codice cliente). Con il successivo ciclo if si controlla se il valore di 'next_one' è contenuto all'interno di 'name_art2' e quindi all'interno di task; se questa condizione si verifica allora tale valore verrà inserito all'interno della lista che prende il nome di 'imp'.

```
                    if imp != []:
                        if (imp[0] in name_art2) == True:
                            if len(b) >1 :
                                for keys in matrix_setup:
                                    if keys[0] == c[0] and keys[1] == c[1]:
                                        matrix_setup[keys][imp[0]] = matrix_setup[c][imp[0]]
                                    else:
                                        matrix_setup[keys][imp[0]] = b[i+1]
                                        c= keys
                                        b.pop(0) #matrice creata
                    next_one = list(next_one)
                    next_one.pop(2)
                    if len(imp) == 2:
                        for keys in matrix_setup:
                            if keys[0] == c[0] and keys[1] == c[1]:
                                matrix_setup[keys][imp[1]] = matrix_setup[keys][imp[0]]
                            else:
                                matrix_setup[keys][imp[1]] = matrix_setup[keys][imp[0]]
                                c= keys
```

Codice. 12

Infine, si ha la seguente porzione di codice che tramite una concatenazione di cicli if e cicli for mette a confronto il valore contenuto all'interno della lista 'imp' con quello contenuto del dizionario 'matrix_setup' precedentemente realizzato in modo da poter poi andare ad inserire una coppia chiave-valore all'interno dei sotto dizionari di 'matrix_setup'.

In questo modo la struttura che ne viene fuori per la realizzazione della matrice dei setup è fatta nel seguente modo:

```
matrix_setup = {('articolo2', 'M2', 'xx1'): {('articolo2', 'M2', 'xx1'): 0,
('articolo2', 'M2', 'xx2'): 0, ('articolo3', 'M2', 'xx1'): 4, ('articolo4',
'M2', 'xx1'): 5, ('articolo6', 'M2', 'xx2'): 1},

('articolo2', 'M2', 'xx2'): {('articolo2', 'M2', 'xx1'): 0, ('articolo2',
'M2', 'xx2'): 0, ('articolo3', 'M2', 'xx1'): 4, ('articolo4', 'M2', 'xx1'):
5, ('articolo6', 'M2', 'xx2'): 1},

('articolo3', 'M2', 'xx1'): {('articolo2', 'M2', 'xx1'): 4, ('articolo2',
'M2', 'xx2'): 4, ('articolo3', 'M2', 'xx1'): 0, ('articolo4', 'M2', 'xx1'):
3, ('articolo6', 'M2', 'xx2'): 1},

('articolo4', 'M2', 'xx1'): {('articolo2', 'M2', 'xx1'): 6, ('articolo2',
'M2', 'xx2'): 6, ('articolo3', 'M2', 'xx1'): 4, ('articolo4', 'M2', 'xx1'):
0, ('articolo6', 'M2', 'xx2'): 1},

('articolo6', 'M2', 'xx2'): {('articolo2', 'M2', 'xx1'): 3, ('articolo2',
'M2', 'xx2'): 3, ('articolo3', 'M2', 'xx1'): 5, ('articolo4', 'M2', 'xx1'):
2, ('articolo6', 'M2', 'xx2'): 0}}
```

Quindi si può dire che 'matrix_setup' è un dizionario dove all'interno viene riportato il nome delle operazioni da svolgere su quella data macchina, ogni operazione a sua volta è

un dizionario dove all'interno vengono riportati una serie di combinazioni di chiave-valore corrispondenti al nome di una possibile futura operazione e il relativo tempo di set-up

Si nota come l'articolo2 è riportato due volte perché ha due differenti codici clienti e inoltre si può osservare anche l'assenza dell'articolo1 questo perché tale prodotto non deve essere lavorato sulla macchina 'M2'.

Per chiarezza, data la complessità del codice, di seguito viene riportato il codice per intero:

```
def lett_setup(nome_file,nome_foglio,ct,task):
    a=[]
    b=[]
    name_art=[]
    name_art2=[]
    #creazione dizionario con le chiavi giuste
    for row in nome_foglio.rows:
        for cell in row:
            a.append(cell.value)
        name_art.append(a[0])
        a=[]
    name_art.pop(0) #in questo modo salvo tutti i nomi degli articoli
all'interno di una lista
    matrix_setup={} #creo il dizionario per i tempi di set-up
    for i in range(len(name_art)):
        product_machine = eval(name_art[i])
        name_art[i]= product_machine #serve per trasformare in stringa il
nome in modo da poterci lavorare
        ct= list (ct)
        for j in range(len(ct)):
            c= list(name_art[i])
            c.append(ct[j])
            c = tuple(c)
            for keys in task: #serve per vedere se il valore di c esiste
all'interno di task
                if keys == c:
                    name_art2.append(c)
                    matrix_setup[c] = {}


    #riempimento dei sotto dizionari
    for row in nome_foglio.rows:
        for cell in row:
            b.append(cell.value)
        if b[0] == 'A/DA':
            for i in range(len(b)):
                b.pop(0)
```

```
        if len(b)== len(name_art2) :
            b[0]= eval(b[0])
            for i in range(len(b)):
                if len(b) == len(name_art2):
                    next_one = b[0]
                    imp = []
                    for h in range(len (ct)):
                        next_one = list(next_one)
                        next_one.append(ct[h])
                        next_one = tuple(next_one)
                        if (next_one in name_art2) == True:
                            imp.append(next_one)
                        if imp != []:
                            if (imp[0] in name_art2) == True:
                                if len(b) >1 :
                                    for keys in matrix_setup:
                                        if keys[0] == c[0] and keys[1] ==
c[1]:
                                            matrix_setup[keys][imp[0]] =
matrix_setup[c][imp[0]]
                                        else:
                                            matrix_setup[keys][imp[0]] =
b[i+1]
                                            c=keys

                                            b.pop(0) #matrice creata
                        next_one = list(next_one)
                        next_one.pop(2)
                        if len(imp) == 2:
                            for keys in matrix_setup:
                                if keys[0] == c[0] and keys[1] == c[1]:
                                    matrix_setup[keys][imp[1]] =
matrix_setup[keys][imp[0]]
                                else:
                                    matrix_setup[keys][imp[1]] =
matrix_setup[keys][imp[0]]
                                    c= keys

        b=[]
    return(matrix_setup)
```

Tale codice deve essere ripetuto tante volte quante sono le macchine su cui bisogna effettuare le lavorazioni e per la quale si hanno le matrici con i tempi di setup; dunque, per

il caso di studio illustrato nel primo paragrafo tale codice deve essere ripetuto due volte dato che le macchine in considerazione sono due.

4.4. Creazione dell'input

Tramite le funzioni 'lett_file' e 'lett_setup' descritte nei due paragrafi precedenti è possibile creare la variabile di input che servirà successivamente per la modellazione del problema.

Nel dettaglio:

```
11  tasks , ct=lett_file()
12
13  nome_file = load_workbook(filename='input.xlsx', read_only=True)
14  nome_foglio1 = nome_file['M1'] #caricamento matrice setup per la prima macchina
15  nome_foglio2 = nome_file['M2'] # caricamento matrice setup per la seconda macchina
16  tsetup1= lett_setup(nome_file,nome_foglio1,ct , tasks)
17  tsetup2= lett_setup(nome_file,nome_foglio2 ,ct,tasks)
18  tsetup= {**tsetup1, **tsetup2} #unire due dizioanri
19  for keys in tasks:
20      for items in tsetup:
21          if items == keys:
22              tasks[keys]['tsetup'] = tsetup[items]
```

Codice. 13

vengono richiamate le due funzioni in modo tale da avere a disposizione l'input pronto per essere utilizzato.

Con la riga 18 (tsetup={**tsetup1, **tsetup2}) vengono uniti i contenuti informativi relativi ai tempi di setup delle due macchine e poi con i due cicli for successivi (righe 18-19) tale informazione viene aggiunta al parametro tasks.

Così facendo l'input complessivo del problema avrà la seguente forma:

```
{('articolo1', 'M1', 'xx1'): {'dur': 250, 'prec': None, 'tsetup':
{('articolo1', 'M1', 'xx1'): 0, ('articolo2', 'M1', 'xx2'): 1, ('articolo2',
'M1', 'xx1'): 1, ('articolo3', 'M1', 'xx1'): 2, ('articolo4', 'M1', 'xx1'):
7, ('articolo6', 'M1', 'xx2'): 2}}
```

dove per ogni operazione sono riportate le informazioni legate alla durata delle operazioni, ad eventuali vincoli di precedenza e i relativi tempi di setup delle macchine per passare dalla produzione del suddetto articolo ad un articolo successivo; questo ragionamento si estende poi per tutte le operazioni che devono essere schedulate.

4.5. Pyomo

Creazione del modello

La libreria Pyomo permette di creare modelli matematici per la risoluzione di problemi di programmazione lineare ricevendo come input i processi che devono essere ancora schedulati.

In particolar modo per poter creare le equazioni di vincolo e la funzione obiettivo prima bisognerà creare dei sottoinsiemi del parametro 'tasks' in modo da poter poi scrivere le equazioni e velocizzare di molto i tempi di calcolo.

```
24  def jobshop_model(tasks):
25
26      model = ConcreteModel()
27      # tasks is a two dimensional set of (j,m) constructed from the dictionary keys
28      model.tasks = Set(initialize = tasks.keys(), dimen=3)
29
30      # the set of jobs is constructed from a python set
31      model.JOBS = Set(initialize = list(set([j for (j,m,z) in model.tasks])))
32
33      # set of machines is constructed from a python set
34      model.MACHINES = Set(initialize = list(set([m for (j,m,z) in model.tasks])))
35
36      model.COD = Set(initialize = list(set([z for (j,m,z) in model.tasks])))
37
38      model.tsetup=Param(model.tasks, initialize=lambda model, j, m,z: tasks[(j,m,z)]['tsetup'])
```

Codice. 14

La funzione per la creazione del modello prende il nome di 'jobshop_model'. Come primo step bisognerà creare il contenitore dove all'interno si andranno ad inserire tutte le informazioni utili per la realizzazione del modello matematico e per la descrizione del problema e questo sarà possibile farlo con la dicitura model = ConcreteModel() (riga 26).

Le operazioni saranno viste come un insieme tridimensionale descritto dai parametri j, m, z che rispettivamente rappresentano il nome dell'articolo, nome della macchina e codice cliente. Tutte le operazioni disponibili all'interno della funzione 'tasks' verranno riportate nel parametro 'model.tasks' (riga 28).

Con 'model.JOB', 'model.MACHINES', 'model.COD' si intendono 3 sottoinsiemi del insieme 'model.tasks' dove sono contenuti all'interno rispettivamente il nome di tutti gli articoli, di tutte le macchine e tutti i codici cliente.

Tutti e 3 gli insiemi possono essere rappresentati come delle liste (non sono delle e vere proprio liste ma prendono il nome di pyomo.core), di fatti abbiamo che:

- Model. JOBS ha dimensione pari a 5 proprio come il numero di articoli da realizzare
- Model.MACHINES ha dimensione pari a 2 come il numero di macchine disponibili
- Model.COD ha dimensione pari a 2 perché si sono ipotizzati due codici cliente.

Invece 'model.tsetup' (riga 38) è un insieme di parametri relativi a tutti i tempi di setup contenuti all'interno dell'insieme 'model.tasks'.

```
40  # the order of tasks is constructed as a cross-product of tasks and filtering
41  model.TASKORDER = Set(initialize = model.tasks * model.tasks, dimen=6,
42      filter = lambda model, j, m,z, k, n,y: (k,n,y) == tasks[(j,m,z)]['prec'])
43
44  # the set of disjunctions is cross-product of jobs, jobs, and machines
45  model.DISJUNCTIONS = Set(initialize = model.JOBS * model.JOBS * model.MACHINES*model.COD*model.COD, dimen=5,
46      filter = lambda model, j, k, m,z,y : j < k and (j,m,z) in model.tasks and (k,m,y) in model.tasks )
47
48  # load duration data into a model parameter for later access
49  model.dur = Param(model.tasks, initialize=lambda model, j, m,z: tasks[(j,m,z)]['dur'])
50
51  model.tsetup=Param(model.tasks, initialize=lambda model, j, m, z: tasks[(j,m, z)]['tsetup'])
```

Codice. 15

'model.TASKORDER' (riga 41) è un sottoinsieme di 'model.tasks' dove vengono riportati tutti i legami di precedenza delle attività; in particolar modo è strutturato nel seguente modo:
('articolo2' , 'M2' , 'xx2', 'articolo2', 'M1' , 'xx1'). Quindi può essere rappresentato con 6 indici dove tre di questi rappresentano l'operazione che deve seguire l'operazione descritta dai restanti 3 indici. Tale insieme ha una lunghezza pari a 5 proprio perché, come si può vedere nella tabella di inizio capitolo, sono cinque le operazioni che devono dare precedenza ad un'altra operazione.

'Model.DISJUNCTIONS' (riga 45) è un sottoinsieme sempre di 'model.tasks' dove vengono riportati eventuali conflittualità tra le operazioni dato che se una macchina è impegnata nella lavorazione di un articolo non potrà iniziare a lavorare un nuovo articolo. Gli elementi di tale insieme si presentano nel seguente modo:

('articolo3', 'articolo4', 'M1', 'xx1', 'xx1')

Ogni elemento del sottoinsieme è caratterizzato da 5 indici che riportano nelle prime due posizioni il nome degli articoli da produrre mentre nella terza posizione il nome della

macchina che deve lavorare entrambi gli articoli, nelle ultime due posizioni vengono riportati i codici cliente. Lo scopo di tale sottoinsieme è quello di poter poi scrivere le equazioni di disgiunzione.

'Model.dur '(riga 49) e 'model.tsetup '(riga 51) invece sono due parametri che racchiudono le informazioni relative alla durata dell'attività e ai tempi di setup contenuti all'interno di 'model.tasks'.

```
53      # establish an upper bound on makespan
54      ub = sum([model.dur[j,m,z] + 100 for (j,m,z) in model.tasks])
55
56      model.makespan = Var(bounds=(0, ub))
57
58      model.start = Var(model.tasks, bounds=(0, ub))
59
60      model.objective = Objective(expr = model.makespan, sense = minimize)
61
62      model.finish = Constraint(model.tasks, rule=lambda model, j, m,z:
63          model.start[j,m,z] + model.dur[j,m,z] <= model.makespan)
```

Codice. 16

Con il termine 'ub' (riga 54) si andrà a considerare un limite massimo per il valore del makespan da non poter superare ed è dato dalla somma delle durate di tutte le attività più un fattore aggiuntivo di 100 che tiene conto di eventuali tempi di setup lunghi.

Successivamente bisognerà definire il makespan (riga 56) come un valore numerico che varia da 0 al valore massimo 'ub'.

Con 'model.start' (riga 58) si intende una variabile da associare a tutte le attività e andrà a rappresentare il tempo di inizio dell'attività stessa

Una volta giunti a questo punto si potrà definire la funzione obiettivo (riga 60) che nel caso di interessa risulta essere quella di minimizzare il makespan e si potranno andare a definire le varie equazioni di vincolo.

L'espressione di 'model.finish' (riga 62-63) la si può leggere e interpretare come se il tempo di fine di ogni attività contenuta all'interno di 'model.tasks' non debba superare il valore di makespan.

```
65    model.preceding = Constraint(model.TASKORDER, rule=lambda model, j, m, z,k,n, y:
66        model.start[k,n,y] + model.dur[k,n,y] <= model.start[j,m,z])
67
68    model.disjunctions = Disjunction(model.DISJUNCTIONS, rule=lambda model,j,k,m,z,y:
69        [model.start[j,m,z] + model.dur[j,m,z] + model.tsetup[j,m,z][k,m,y] <= model.start[k,m,y],
70         model.start[k,m,y] + model.dur[k,m,y] + model.tsetup[k,m,y][j,m,z] <= model.start[j,m,z]])
```

Codice. 17

In 'model.precending' (riga 65) si fa riferimento al contenuto creato all'interno di 'model.TASKORDER' e questa espressione serve per garantire il rispetto del vincolo di successione delle attività e quindi a garantire l'ordine giusto dello svolgimento delle attività.

Infine l'espressione 'model.disjunctions '(righe 68-70) permetterà di garantire che se una macchina è impegnata con una lavorazione non potrà iniziarne un'altra, quindi si andranno ad evitare conflitti sull'utilizzo della risorsa.

Ora è possibile andare a risolvere il modello matematico per trovare una soluzione che vada a minimizzare il makespan.

```
76  def jobshop_solve(model):
77      SolverFactory('cbc').solve(model)
78      results = [{'Job': (j,z),
79                  'Machine' : m,
80                  'Prio' :z,
81                  'Start':  model.start[j, m,z](),
82                  'Finish':  model.start[j, m,z]() + model.dur[j,m,z],
83                  'Duration': model.dur[j,m,z] ,
84                  }
85                 for j,m,z in model.tasks]
86      return results
```

Codice. 18

'cbc' è un risolutore di programmazione lineare di interi misti open source scritto in C++.

Con la seguente funzione si andrà a risolvere il modello creato precedentemente in modo tale da avere in output una strutta del tipo:

```
{'Job': ('articolo1', 'xx1'), 'Machine': 'M1', 'Prio': 'xx1', 'Start': 791.0,
'Finish': 1041.0, 'Duration': 250}
```

Ora si hanno informazioni relative al tempo di inizio e fine dell'operazione; tuttavia, la struttura dati che ne viene fuori risulta essere complessa e difficile da manipolare quindi è stato utile l'uso della libreria Pandas che permette di eseguire operazioni su strutture matriciali.

```
scheda = pd.DataFrame(results)
```

Con la seguente dicitura si avrà una struttura dati organizzata del tipo:

	Job	Machine	Prio	Start	Finish	Duration
0	(articolo1, xx1)	M1	xx1	791.0	1041.0	250
1	(articolo2, xx1)	M1	xx1	0.0	84.0	84
2	(articolo2, xx1)	M2	xx1	254.0	394.0	140
3	(articolo3, xx1)	M1	xx1	477.0	677.0	200
4	(articolo3, xx1)	M2	xx1	0.0	250.0	250
5	(articolo4, xx1)	M1	xx1	203.0	473.0	270
6	(articolo4, xx1)	M2	xx1	627.0	1227.0	600
7	(articolo2, xx2)	M1	xx2	0.0	198.0	198
8	(articolo2, xx2)	M2	xx2	254.0	584.0	330
9	(articolo6, xx2)	M2	xx2	585.0	625.0	40
10	(articolo6, xx2)	M1	xx2	678.0	788.0	110

Questa risulta essere la soluzione del problema di schedulazione in esame tuttavia tale soluzione può essere migliorata. Si noti come i tempi sono riportati in termini assoluti, in una realtà industriale è molto meglio poter riportare i risultati dei tempi in termini specifici (codice 19).

```
97  #converto i tempi con date reali
98  for i in range(len(results)):
99      results[i]['Start'] = {'Anno': 2022, 'Mese': 5, 'Giorno':6 ,'Ora':8 , 'Minuti': int(results[i]['Start'])}
100     results[i]['Finish'] = {'Anno':2022,'Mese': 5, 'Giorno':6 ,'Ora':8 , 'Minuti': int(results[i]['Finish'])}
101     results[i]['Duration'] = {'Anno':2022, 'Mese': 5, 'Giorno':6 ,'Ora':0 , 'Minuti': int(results[i]['Duration'])}
102     #cambio minuti --> ore
103     for items in results[i]:
104         if items == 'Start' or items == 'Finish' or items == 'Duration':
105             while results[i][items]['Minuti'] >= 60 :
106                 results[i][items]['Ora'] = results[i][items]['Ora'] +1
107                 results[i][items]['Minuti'] = int(results[i][items]['Minuti']) - 60
108             while results[i][items]['Ora'] >= 24:
109                 results[i][items]['Ora'] = results[i][items]['Ora'] -24
110                 results[i][items]['Giorno'] = results[i][items]['Giorno'] + 1
111             for keys in results[i][items]:
112                 if results[i][items][keys] < 10:
113                     results[i][items][keys] = str(results[i][items][keys])
114                     results[i][items][keys] = '0' + results[i][items][keys]
115
116             results[i][items] =  '2022'+ '-' + str(results[i][items]['Mese']) + '-' + str(results[i][items]['Giorno'])
117 schedule = pd.DataFrame(results)
118 schedule.to_csv('schedule.csv',index=False)
```

Codice. 19

Con la seguente parte di codice è possibile convertire i tempi riportati nella schedulazione in date, si è scelto come data d'inizio il 6 Maggio alle ore 08:00. Con la riga 118 si andrà a creare un file .csv dove verrà riportata la schedulazione in formato tabellare.

```
125   #Read Data from schedule.csv
126   df =pd.read_csv('schedule.csv')
127   df.head()
128   #Convert dates to datetime format
129   df.Start=pd.to_datetime(df.Start)
130   df.Finish=pd.to_datetime(df.Finish)
131   df.Duration=pd.to_datetime(df.Duration)
132
133   df=df.sort_values(by='Start', ascending=True)
```

Codice. 20

Infine, con le seguenti righe di codice si andranno a convertire le date precedentemente create in un formato chiamato 'datatime'. Con tale formato sarà possibile fare operazioni di somma e sottrazione con le date.

4.6. Realizzazione del Gantt

Risulta essere molto importante ai fini della comprensione dei risultati avere anche un riscontro visivo tramite la realizzazione di un diagramma a barre o diagramma di Gantt.

Il diagramma di Gantt è uno strumento di supporto estremamente diffuso nelle realtà industriali per la sua estrema facilità nell'interpretarlo. Nel caso specifico è stato deciso di realizzare un doppio diagramma di Gantt dove in ascissa ci sarà sempre una scala di tempi mentre nelle ordinate del primo diagramma si avrà il nome degli articoli e nel secondo il nome delle macchine. Così facendo si avrà un'idea della schedulazione lato macchina e lato articolo.

Per la realizzazione di tale diagramma è stata utilizzata la libreria Matplotlib.

```
9  def visualize(df,scheda):
10     # df = pd.DataFrame(data_xm)
11     JOBS = sorted(list(df['Job'].unique()))
12     MACHINES = sorted(list(df['Machine'].unique()))
13     makespan = df['Finish'].max()
14
15     bar_style = {'alpha':1.0, 'lw':25, 'solid_capstyle':'butt'}
16     text_style = {'color':'white', 'weight':'bold', 'ha':'center', 'va':'center'}
17     colors = mpl.cm.Dark2.colors
18
19     scheda.sort_values(by=['Job', 'Start'])
20     scheda.set_index(['Job', 'Machine'], inplace= True)
21     df.set_index(['Job', 'Machine'], inplace=True)
```

Codice. 21

La funzione prende il nome di ‘visualize’ e riceve in input i parametri ‘scheda’ e ‘df’ creati nel paragrafo precedente.

‘JOBS’ e ‘MACHINES’ sono due liste dove sono contenuti rispettivamente i nomi degli articoli e i nomi delle macchine per eseguire le lavorazioni. Si andranno a settare poi dei parametri per la realizzazione del diagramma come lo stile delle barre, stile del testo e i colori. Importante anche definire subito il valore del makespan.

I dati contenuti dentro il parametro scheda sono disordinati, con i comandi .sort e .set_index si ordinano le operazioni da quella che inizia all’istante zero fino all’ultima da realizzare.

```
fig, ax = plt.subplots(2,1, figsize=(12, 5+(len(JOBS)+len(MACHINES))/4))
```

Con la seguente riga di codice si andrà a creare la base su cui poi costruire il diagramma, in particolare si scelgono una serie di parametri come la dimensione del diagramma.

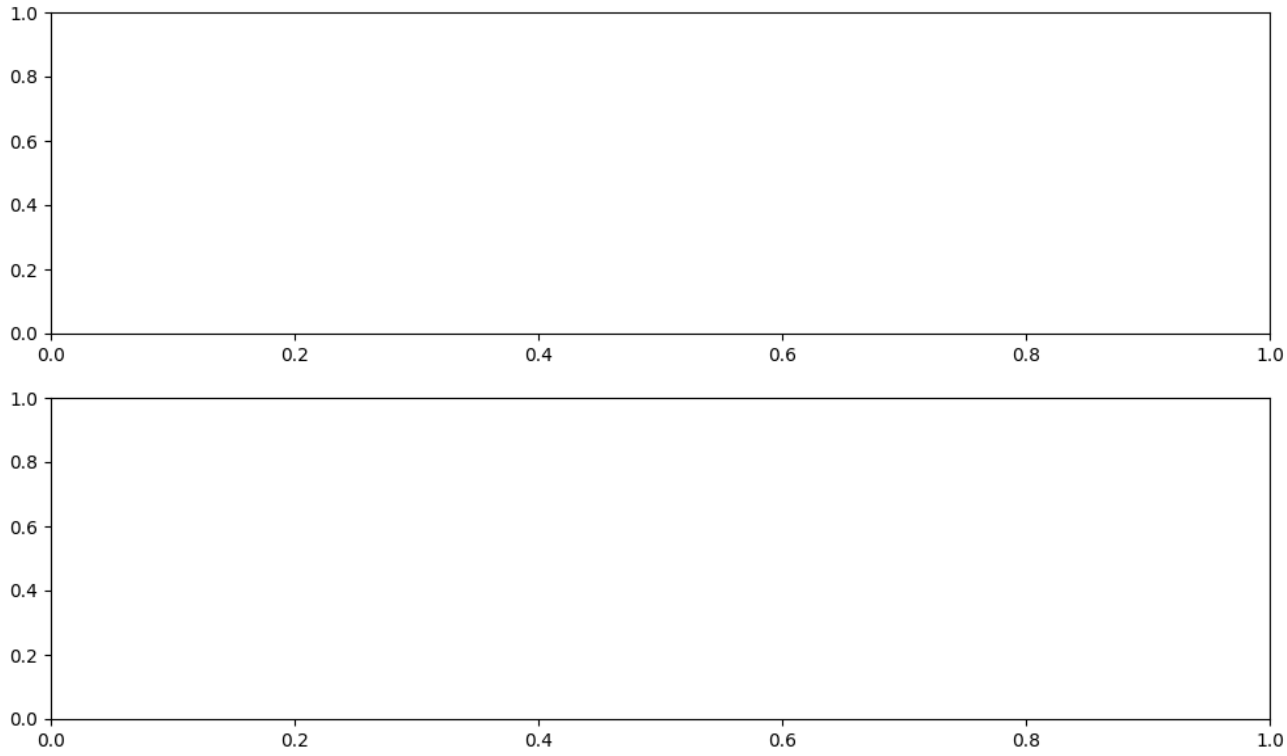


Figure 10

Questa risulta quindi essere una sorta di foglio bianco su cui andare a costruire passo dopo passo il Gantt.

```
    for jdx, j in enumerate(JOBS, 1): # j= nome art ldx un contatore
        for mdx, m in enumerate(MACHINES, 1): #m = nome macchin mdx contatore
            if (j,m) in df.index:

                xs = df.loc[(j,m), 'Start'] #xs = inizio dell'asse x per ogni attività
                xf = df.loc[(j,m), 'Finish'] #xf fine dell'asse x per ogni attività
                c= literal_eval(j)
                xs1= scheda.loc[(c,m), 'Start']
                xf1=scheda.loc[(c,m), 'Finish']
                xm =(xs1 +xf1)/2
                data_xm={'Mese': 5, 'Giorno':6 ,'Ora':8 , 'Minuti':xm}
                data_xm['Minuti'] = int(data_xm['Minuti'])
```

Codice. 22

Per realizzare i blocchi del diagramma bisognerà usare un doppio ciclo for dove l'indice 'j' rappresenta una stringa contenente il nome dell'articolo e il codice cliente mentre l'indice 'm' sta ad indicare il nome della macchina.

Se la combinazione j,m esiste all'interno di 'df' allora sarà sicuramente un'operazione esistente.

Con 'xs' ed 'xf' si andranno ad indicare gli istanti temporali di inizio e fine dell'attività j,m; è importante conoscere anche il valore medio perché servirà successivamente per scrivere sulla barra il nome dell'operazione e quindi servirà come riferimento per aggiungerci un scritta. Trovare il valor, medio non è semplice quanto si pensa dato che il tempo è espresso in formato 'DataTime' e quindi non è un valore numerico assoluto e per farlo allora si utilizza la schedulazione originaria 'scheda', quella ottenuta direttamente dal Solver che presentava i tempi assoluti.

'scheda' differisce da 'df' dal fatto che il valore di j non è una stringa ma una tupla e dunque bisognerà convertire il formato e nel codice in esame prende il nome di 'c' la nuova variabile.

Una volta individuata l'operazione in esame all'interno di 'scheda' si potrà attingere al valore di inizio e fine dell'attività così da ricavarsi il valor medio.

```
35          data_xm={'Mese': 5, 'Giorno':6 ,'Ora':8 , 'Minuti':xm}
36          data_xm['Minuti'] = int(data_xm['Minuti'])
37          while data_xm['Minuti'] >= 60 :
38              data_xm['Ora'] = data_xm['Ora'] +1
39              data_xm['Minuti'] = int(data_xm['Minuti']) - 60
40          while data_xm['Ora'] >= 24:
41              data_xm['Ora'] = data_xm['Ora'] -24
42              data_xm['Giorno'] = data_xm['Giorno'] + 1
43          for keys in data_xm:
44              if data_xm[keys] < 10:
45                  data_xm[keys] = str(data_xm[keys])
46                  data_xm[keys] = '0' + data_xm[keys]
47
48          data_xm['Minuti'] = str(data_xm['Minuti'])
49
50          dx=  str('2022'+ '-' + str(data_xm['Mese']) + '-' + str(data_xm['Giorno']) + '\t' +
51              str(data_xm['Ora']) + ':' + str(data_xm['Minuti']) + "." + "00")
52
53          data_xm=pd.to_datetime(dx)
```

Codice. 23

Con la seguente porzione di codice si andrà a riportare il valore di 'xm' (valor medio della durata dell'attività' in formato 'DataTime' andando ad eseguire le dovute conversioni dal valore assoluto ad un formato anno/mese/giorno/ora/minuti.

Ora si hanno informazioni riguardanti all'inizio dell'attività, la fine e il tempo medio. Questi dati dovranno essere riportati sul grafico nel seguente modo:

```
54          ax[0].plot([xs, xf], [jdx]*2, c=colors[mdx%7], **bar_style)
55          ax[0].text( data_xm ,jdx, m, **text_style)
56          ax[1].plot([xs, xf], [mdx]*2, c=colors[jdx%7], **bar_style)
57          ax[1].text( data_xm, mdx, j, **text_style)
```

Codice. 24

Con la riga ax[0].plot si andranno ad inserire i dati di inizio e fine attività sul primo grafico mentre con il comando .text il nome della macchina su cui dovrà essere eseguita l'operazione.

Allo stesso modo, con ax.[1].plot si inseriranno nel secondo diagramma i tempi di inizio e fine operazione e con il comando .text il nome dell'articolo e relativo codice cliente.

Il risultato che ne viene fuori è il seguente:

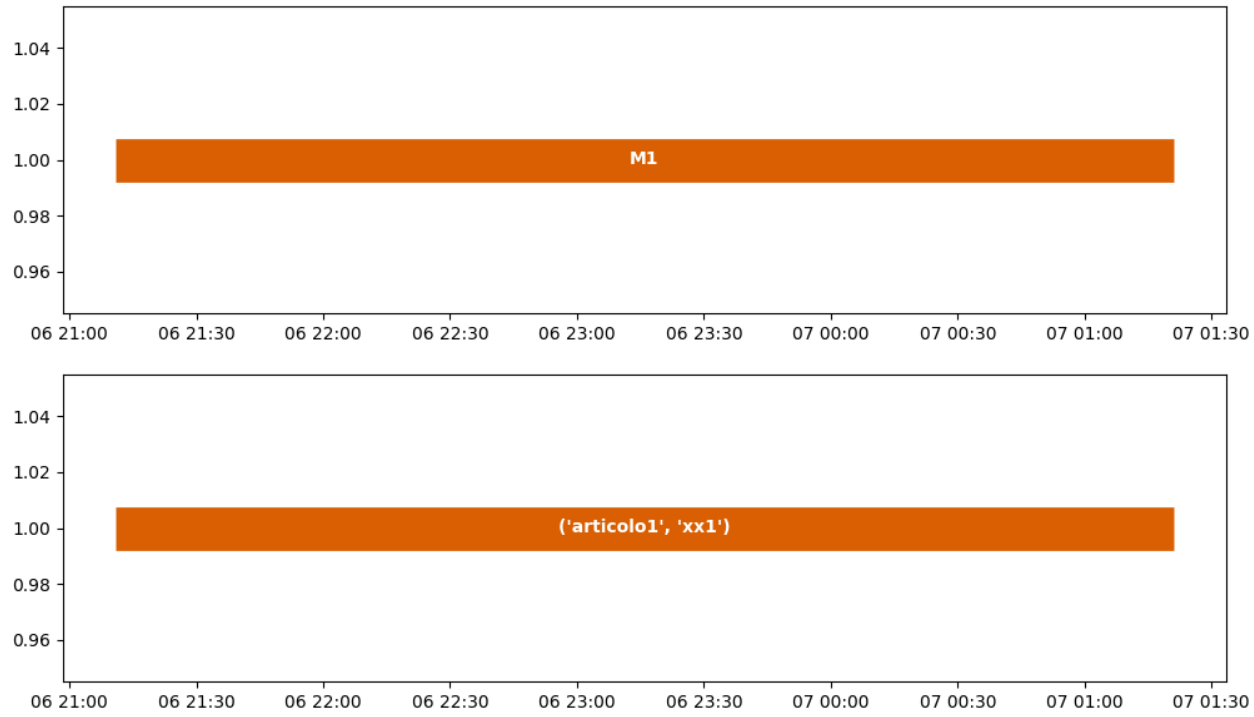


Figure 11

Si noti come l'asse delle ascisse è cambiata e riporti la data con l'ora e i minuti di inizio e fine. queste due porzioni di codice dovranno essere ripetute per tutte le attività fino a completare il Gantt.

```
59   ax[0].set_title('Job schedule')
60   ax[0].set_ylabel('Job')
61   ax[1].set_title('Machine schedule')
62   ax[1].set_ylabel('Machine')
```

Codice. 25

Per maggiore chiarezza nella lettura del grafico è importante inserire i titoli sui due grafici e anche sull'asse delle ordinate.

```
63  for idx, s in enumerate([JOBS, MACHINES]):# itera per i due grafici
64    #idx = 0    ---- s = ['articolox' , 'articoloy' , articoloz']
65    #idx = 1    ---- s = ['M1' , 'M2']
66        ax[idx].set_ylim(0.5, len(s) + 0.5)
67        ax[idx].set_yticks(range(1, 1 + len(s)))
68        ax[idx].set_yticklabels(s)
69        ax[idx].plot([makespan]*2, ax[idx].get_ylim(), 'r--')
70        ax[idx].grid(True)
71
72    fig.tight_layout()
73    return()
```

Codice. 26

Infine, questo ciclo for serve per avere una migliore lettura dei grafici, in particolare si andranno a modificare i valori in ascissa e saranno inseriti solo i valori utili per la lettura.

Con la penultima riga si andrà a disegnare una linea tratteggiata su entrambi i grafici che rappresenterà il makespan mentre con l'ultima si aggiungerà una griglia sullo sfondo per avere una lettura più chiara del Gantt.

Il risultato finale avrà la seguente forma:

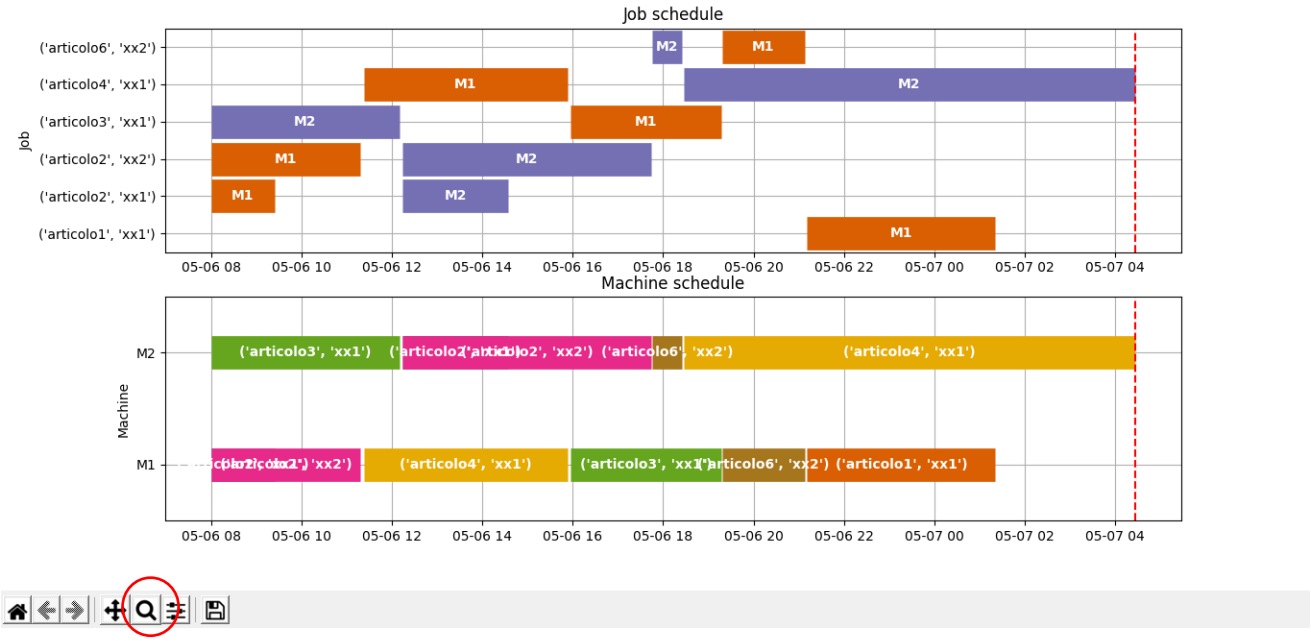


Figure 12

Si noti come l'asse delle ascisse è cambiato nuovamente, con '05-06' si intende la data del 6 Maggio. Per avere maggiore dettaglio sull'ora o il minuto di inizio o fine attività si potrà usare la funzione di ingrandimento (cerchiata in rosso).

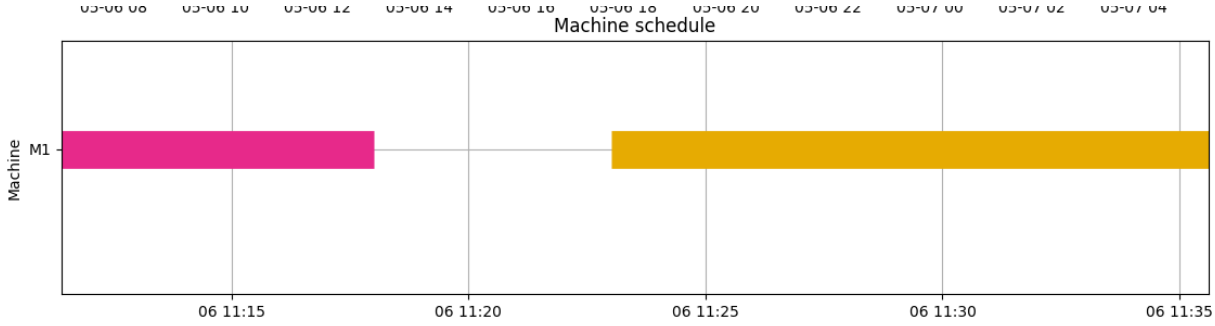


Figure 13

Così facendo la scala delle ascisse si riadatterà e si potranno avere informazioni più dettagliate (Figure 13).

5. VARIAZIONE DEI TEMPI DI CALCOLO AL VARIARE DI DIVERSI PARAMETRI.

Un aspetto importante da tenere in considerazione quando si parla di algoritmi matematici per la schedulazione è quello di considerare come varia il tempo di calcolo in funzione di diversi parametri. Per calcolare i tempi di calcolo in maniera efficace è stata usata la libreria 'Time' che fornisce il risultato espresso in secondi.

Per valutare la relazione dei tempi di calcolo con i diversi fattori si sono scelti diversi scenari, i tempi sono stati calcolati come media di 3 prove.

5.1. Relazione del tempo al variare del numero di macchine.

Per valutare come varia il tempo di processamento in funzione del numero di macchine disponibili sono stati creati diversi scenari:

- Scenario 1: 15 articoli da lavorare su 4 macchine per un totale di 36 operazioni;
- Scenario 2: 15 articoli da lavorare su 6 macchine per un totale di 47 operazioni;
- Scenario 3: 15 articoli da lavorare su 8 macchine per un totale di 47 operazioni
- Scenario 4: 15 articoli da lavorare su 10 macchine per un totale di 47 operazioni.

Tutte le prove sono state effettuate su un PC con le seguenti caratteristiche:

- Processore: AMD Ryzen 3 2200U with Radeon Vega Mobile Gfx 2.50 GHz,
- Ram: 12,0 GB.

Nel primo scenario ci sono state diverse problematiche dato che i tempi di calcolo superavano di gran lunga le due ore nonostante il numero delle operazioni fosse inferiore rispetto agli scenari successivi, mentre questi ultimi hanno fornito dei dati molto significativi; i dati sono riportati nella seguente tabella.

Scenario1 (15x4)	Scenario2 (15x6)	Scenario3 (15x8)	Scenario4 (15x10)
no	757,52	119	35
no	730	98	24
no	726	77	23

#VALORE!	737,78	97,9593333	27,17733333

Tramite tali dati si è provato a creare una curva che rappresenti al meglio l'andamento del tempo al variare del numero di macchine escludendo però il primo scenario.

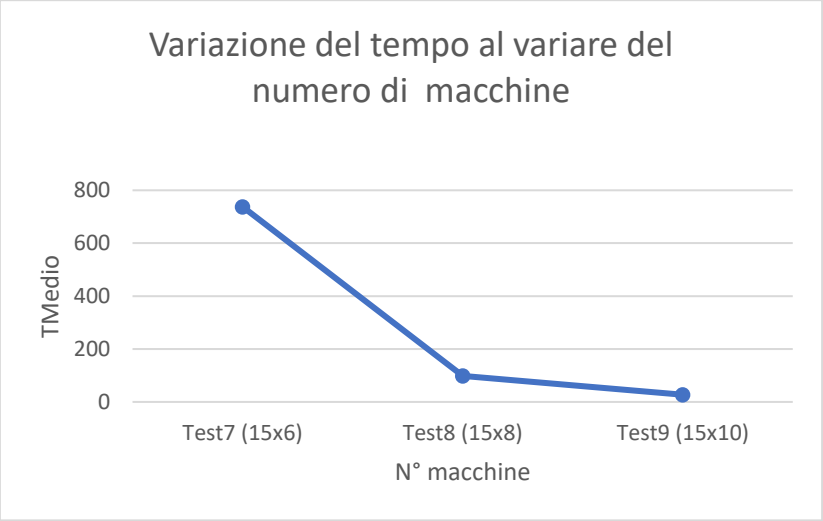


Figure 14

Dalla curva in figura 14 si nota come all'aumentare del numero di macchine c'è una notevole diminuzione dei tempi di calcolo.

Questo andamento è stato riscontrato anche in un secondo test eseguito su 4 scenari che prevedevano:

- Scenario 1: 10 articoli da lavorare su 2 macchine;
- Scenario 2: 10 articoli da lavorare su 4 macchine;
- Scenario 3: 10 articoli da lavorare su 6 macchine;
- Scenario 4: 10 articoli da lavorare su 8 macchine.

Anche per questo studio si è deciso di prendere il tempo medio fatto su un campione di 3 prove.

	Test3 (10x2)	Test4 (10x4)	Test5 (10x6)	test10 (10x8)
tempo1	1.799	15	53	7
tempo2	1.796	15	56	7
tempo3	1.798	15	50	7

Tmedio[s]	1797,713	15,0366667	53,1197	6,988666667

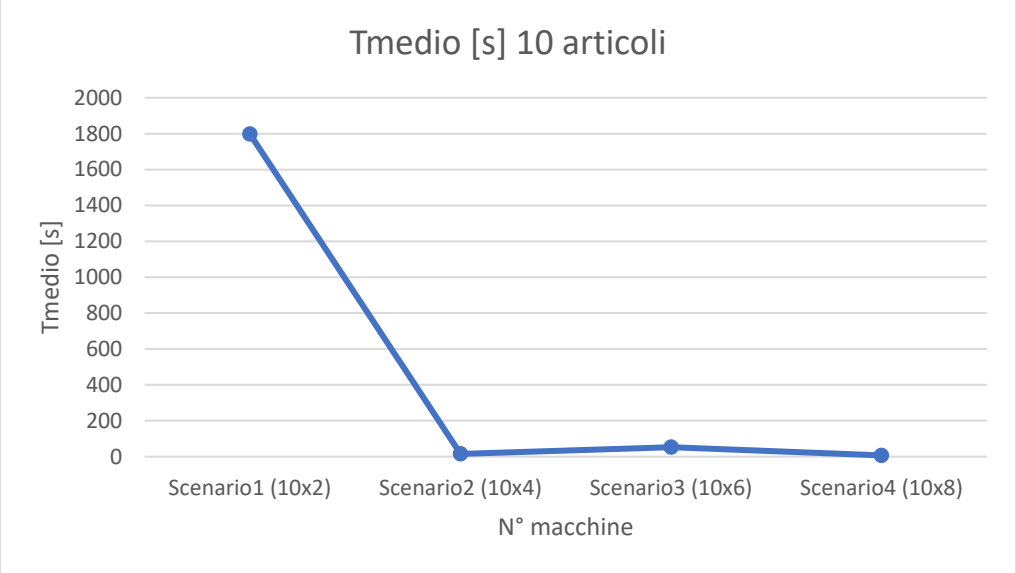


Figure 15

Si noti come anche in questo caso come nel precedente c'è una diminuzione dei tempi di calcolo all'aumentare del numero di macchine tenendo sempre in considerazione che ogni articolo lavora su un ordine e su delle tipologie di macchine differenti dagli altri.

5.2 Relazione del tempo al variare del numero di macchine.

Si è cercato di trovare una relazione tra il tempo di calcolo necessario per risolvere l'algoritmo in funzione del numero di articoli e per farlo si sono ipotizzati 3 scenari differenti:

- Scenario 1: 5 articoli da lavorare su 6 macchine per un totale di 17 operazioni
- Scenario 2: 10 articoli da lavorare su 6 macchine per un totale di 30 operazioni
- Scenario 3: 15 articoli da lavorare su 6 macchine per un totale di 34 operazioni

Ogni articolo di cui si fa riferimento ha un ordine di lavorazione sul parco macchine diverso dagli altri.

Anche per queste prove è stato usato un PC con le stesse caratteristiche di quello descritto nel paragrafo precedente.

Per ogni scenario sono state effettuate 3 misurazioni e si è preso in considerazione il tempo medio espresso in secondi come riportato nella tavella sottostante.

	Test2 (5x6)	Test5 (10x6)	Test7 (15x6)
tempo1	6	53	757,52
tempo2	6	56	730
tempo3	6	50	726
Tmedio	5,942	53,12	737,78

Una volta ottenuti i dati è stato possibile tracciare una curva che rappresentava l'andamento del tempo in funzione del numero di articoli da processare (figure 15).

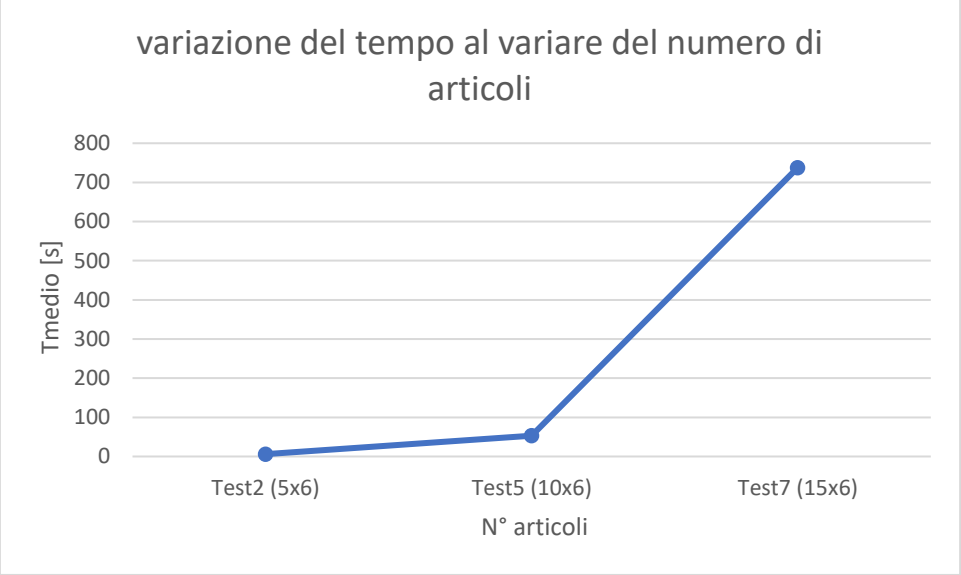


Figure 16

Si noti come, a differenza del caso precedente, in questo caso si ha un aumento esponenziale dei tempi di calcolo all'aumentare della numerosità di articoli. È bene specificare che i vari articoli da processare sono tutti diversi perché qual ora si parlasse di articoli uguali ma con codice cliente differente allora il tempo di calcolo non varia.

5.3. Relazione del tempo al variare del tipo di computer utilizzato.

Di particolare interesse risulta essere confrontare i tempi di calcolo usando due computer con differenti prestazioni e valutarne eventuali differenza.

Le caratteristiche dei due computer sono le seguenti:

- Computer 1:
    - Processore: AMD Ryzen 3 2200U with Radeon Vega Mobile Gfx 2.50 GHz;
    - RAM: 12,0 GB;
- Computer 2:
    - Processore: InterCore i7-7700K, 4,20 GHz
    - RAM: 31,0 GB

Sono state effettuate dieci prove differenti modificando il numero e la varietà di articoli da lavorare e il numero di macchine disponibili; ogni prova presenta le seguenti caratteristiche:

- Test 0: 5 articoli su 2 macchine
- Test 1: 5 articoli su 4 macchine
- Test 2: 5 articoli su 6 macchine
- Test 3: 10 articoli su 2 macchine
- Test 4: 10 articoli su 4 macchine
- Test 5: 10articoli su 6 macchine
- Test 6: 10 articoli su 8 macchine
- Test 7: 15 articoli su 6 macchine
- Test 8: 15 articoli su 8 macchine
- Test 9: 15 articoli su 10 macchine

I tempi (in secondi) riportati nella tabella sottostante fanno riferimento ad una media di 3 misurazioni.

	Test0 (5x2)	Test1 (5x4)	Test2 (5x6)	Test3 (10x2)	Test4 (10x4)	Test5 (10x6)	Test6 (10x8)	Test7 (15x6)	Test8 (15x8)	Test9 (15x10)
Tmedio1	6	6	6	1.798	15	53	7	738	98	27
Tmedio2	2,343	2,304	2,19	1349,509	5,888	21,122	3,864	436,159	48,358	13,646

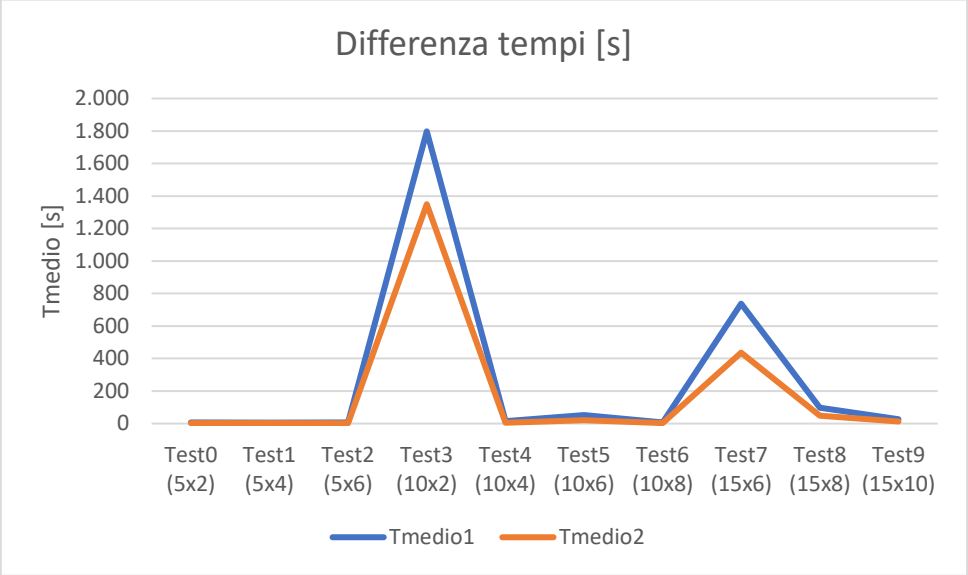


Figure 17

La linea azzurra in figura rappresenta i tempi di calcolo riferiti al primo computer mentre quella arancio al secondo.

Si noti come con il secondo computer, più performante, si ha una diminuzione media dei tempi di circa il 52%.

5.4 Analisi DoE

L'analisi DoE (Design of Experiment) è un potente strumento statistico molto usato nelle realtà industriali per valutare l'influenza di diversi parametri sul risultato finale.

Nel seguente caso tale analisi è stata effettuata per valutare e quantificare l'influenza che hanno tre fattori sul tempo di calcolo. Questi 3 fattori sono il numero di articoli, il numero di macchine e il numero di operazioni. L'obiettivo è quello di capire come varia il tempo di calcolo al variare di tali fattori e poter realizzare un modello matematico che descrive tale comportamento.

L’analisi è stata effettuata su un campione di 8 test caratterizzati dalla combinazione dei valori riportati nella tabella sottostante.

Code	N°articoli	N°macchine	N°operazioni
-1	10	6	31
1	15	8	47

Table 2

Per poter effettuare l’analisi tali valori devono essere codificati, in particolar modo abbiamo che con il codice -1 andremmo ad intendere 10 articolo o 6 macchine o 31 operazioni mentre con il codice 1 si andrà a considerare i restanti valori.

Dalla combinazione di questi parametri (2^3) si ottengono le 8 prove su cui è stata condotta l’analisi (vedi tabella3).

Valori codificati				Valori reali		
			tempo [s]	N°articoli	N°macchine	N°operazioni
-1	-1	-1	53	10	6	31
-1	-1	1	128	10	6	47
-1	1	-1	15	10	8	31
-1	1	1	25	10	8	47
1	1	1	98	15	8	47
1	1	-1	8	15	8	31
1	-1	1	737	15	6	47
1	-1	-1	1000	15	6	31

Table 3

Con riferimento alla tabella 3, nella colonna centrale colorata di rosa sono riportati i tempi di calcolo, espressi in secondi, per ogni prova.

Ai fini di un'analisi più completa bisogna considerare anche il contributo combinato di 2 fattori (tabella 4).

T	N°articoli	N°macchine	N°operazioni	art*oper	art*macch	macch*oper
128	-1	-1	1	-1	1	-1
737	1	-1	1	1	-1	-1
15	-1	1	-1	1	-1	-1
10	1	1	-1	-1	1	-1
53	-1	-1	-1	1	1	1
25	-1	1	1	-1	-1	1
98	1	1	1	1	1	1
900	1	-1	-1	-1	-1	1

Table 4

Con lo strumento 'Analisi Dati – Regressione' presente su Excel è stato possibile ricavare il modello matematico che descrive il comportamento di tali parametri.

	A	B	C	D	E	F	G	H	I
1	OUTPUT RIEPILOGO								
2									
3	*Statistica della regressione*								
4	R multiplo	0,99303957							
5	R al quadrato	0,98612759							
6	R al quadrato corretto	0,90289312							
7	Errore standard	111,722871							
8	Osservazioni	8							
9									
10	ANALISI VARIANZA								
11		*gdl*	*SQ*	*MQ*	*F*	*Significatività F*			
12	Regressione	6	887289,5	147881,5833	11,84758719	0,218805862			
13	Residuo	1	12482	12482					
14	Totale	7	899771,5						
15									
16		*Coefficienti*	*Errore standard*	*Stat t*	*Valore di significatività*	*Inferiore 95%*	*Superiore 95%*	*Inferiore 95,0%*	*Superiore 95,0%*
17	Intercetta	245,75	39,5	6,221518987	0,101457679	-256,1450871	747,6450871	-256,1450871	747,6450871
18	N°articoli	190,5	39,5	4,82278481	0,130158118	-311,3950871	692,3950871	-311,3950871	692,3950871
19	N°macchine	-208,75	39,5	-5,28481013	0,119054594	-710,6450871	293,1450871	-710,6450871	293,1450871
20	N°operazioni	1,25	39,5	0,03164557	0,979860526	-500,6450871	503,1450871	-500,6450871	503,1450871
21	art*oper	-20	39,5	-0,50632911	0,701617541	-521,8950871	481,8950871	-521,8950871	481,8950871
22	art*macch	-173,5	39,5	-4,39240506	0,142507493	-675,3950871	328,3950871	-675,3950871	328,3950871
23	macch*oper	23,25	39,5	0,588607595	0,661317889	-478,6450871	525,1450871	-478,6450871	525,1450871

Figure 18

In figura sono riportati i dati ottenuti dall'analisi, in particolar modo il valore di 'R al quadrato' pari a 0,92 convalida il modello matematico.

La colonna 'Valore di significatività' aiuta a comprendere qunato un fattore influisca sul risultato finale, in particolar modo se tale valore è inferiore a 0,15 (righe evidenziate in rosa) allora il contributo di tale parametro è importante.

È stata eseguita nuovamente l'analisi prendendo in considerazione solo i parametri evidenziati in modo da avere un modello più completo.

Il risultato è riportato nell' immagine 19:

	A	B	C	D	E	F	G	H	I
1	OUTPUT RIEPILOGO								
2									
3	*Statistica della regressione*								
4	R multiplo	0,96145984							
5	R al quadrato	0,92440503							
6	R al quadrato corrett	0,82361173							
7	Errore standard	185,947125							
8	Osservazioni	8							
9									
10	ANALISI VARIANZA								
11		*gdl*	*SQ*	*MQ*	*F*	*Significatività F*			
12	Regressione	4	1268439	317109,75	9,171294913	0,049604419			
13	Residuo	3	103729	34576,33333					
14	Totale	7	1372168						
15									
16		*Coefficienti*	*Errore standard*	*Stat t*	*Valore di significatività*	*Inferiore 95%*	*Superiore 95%*	*Inferiore 95,0%*	*Superiore 95,0%*
17	Intercetta	283	65,74223655	4,304690787	0,023073191	73,77886219	492,2211378	73,77886219	492,2211378
18	N°articoli	227,75	65,74223655	3,464287374	0,040513865	18,52886219	436,9711378	18,52886219	436,9711378
19	N°macchine	-246,5	65,74223655	-3,74949215	0,033127849	-455,7211378	-37,27886219	-455,7211378	-37,27886219
20	N°operazioni	36	65,74223655	-0,54759317	0,622081061	-245,2211378	173,2211378	-245,2211378	173,2211378
21	art*macch	-211,25	65,74223655	-3,21330717	0,048832828	-420,4711378	-2,028862189	-420,4711378	-2,028862189

Figure 19

Nella colonna 'Coefficenti' sono riportati i coefficienti moltiplicativi per la realizzazione del modello matematico.

L'equazione che ne viene fuori risulta pertanto essere:

T= 283+ 227,75N°articoli -246,5N°macchine +36N°operzioni -211,25art*macchine

Tale equazione risulta confermare quanto detto nei paragrafi precedenti. In particolar modo si noti come i tempi di calcolo aumentino all'aumentare del numero di articoli ma diminuiscono con l'aumentare del numero di macchine.

Con lo strumento 'risolutore' presente all'interno di Excel è stato possibile risolvere tale equazione per tutte le 8 prove effettuate in precedenza.

Infine, i dati ottenuti dal risolutore sono stati confrontati con quelli reali per valutarne l'errore su ogni test effettuato (tabella 5).

t reale [s]	t previsto [s]	residuo
53	54,5	-1,5
128	126,5	1,5
15	16	-1
25	56	-31
98	89	9
10	17	-7
737	1004,5	-267,5
900	932	-32

Table 5

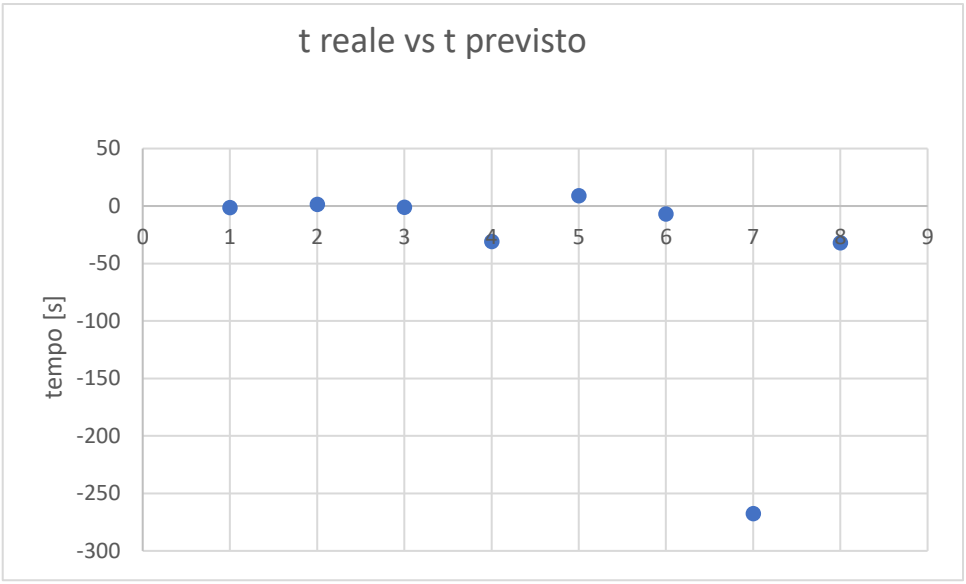


Figure 20

In conclusione, si noti come il modello matematico creato approssima in maniera buona i dati reali tranne per il caso 7 che risulta essere un anomalia.

6. Caso di studio azienda Matrix Modelleria

L'algoritmo è stato testato anche su un caso reale dell'azienda Matrix Modelleria. Tale azienda è specializzata nella produzione di articoli in ceramica, il caso di studio si differenza da quello trattato prima perché in questo caso si fa riferimento ad un produzione di tipo flow-shop, ovvero tutti gli articoli devono essere lavorati sulle stesse macchine e con la stessa sequenza.

In particolar modo la schedulazione è stata effettuata per un ordine di produzione che prevedeva 4 articoli differenti che devono subire 6 lavorazioni in diverse stazioni.

Tali stazioni sono:

- Fase di colaggio
- Fase di rifinitura
- Fase di smaltatura
- Fase di carico/scarico
- Fase di ritocco
- Fase di imballaggio

Non era stato specificato il tempo di produzione unitario ma veniva fornito direttamente il tempo necessario alla produzione dell'intero lotto relativo ad un articolo.

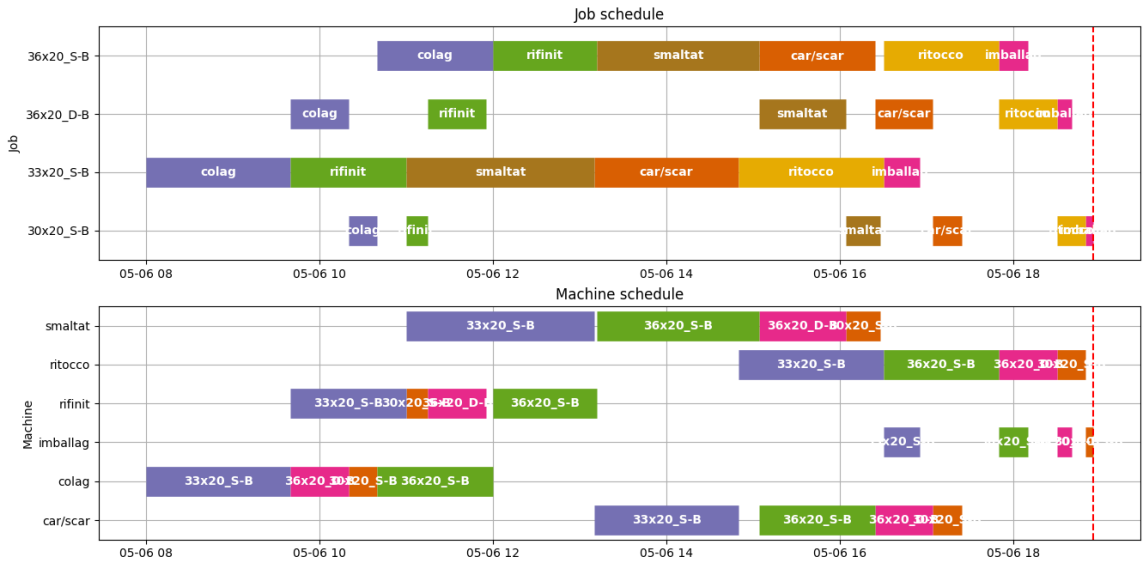
I tempi di set-up per il passaggio di produzione da un articolo all'altro non era stato specificato e dunque è stato supporto pari a 0.

Table 6

Task (job,machine,cod)	Duration	Prerequisite Task	Domanda
('30x20_S-B','colag','155_01_0054')	20	None	1
('30x20_S-B','rifinit','155_01_0054')	15	('30x20_S-B','colag','155_01_0054')	1
('30x20_S-B','smaltat','155_01_0054')	24	('30x20_S-B','rifinit','155_01_0054')	1
('30x20_S-B','car/scar','155_01_0054')	20	('30x20_S-B','smaltat','155_01_0054')	1
('30x20_S-B','ritocco','155_01_0054')	20	('30x20_S-B','car/scar','155_01_0054')	1
('30x20_S-B','imballag','155_01_0054')	5	('30x20_S-B','ritocco','155_01_0054')	1
('33x20_S-B','colag','155_01_0054')	100	None	1
('33x20_S-B','rifinit','155_01_0054')	80	('33x20_S-B','colag','155_01_0054')	1
('33x20_S-B','smaltat','155_01_0054')	130	('33x20_S-B','rifinit','155_01_0054')	1
('33x20_S-B','car/scar','155_01_0054')	100	('33x20_S-B','smaltat','155_01_0054')	1

('33x20_S-B','ritocco','155_01_0054')	100	('33x20_S-B','car/scar','155_01_0054')	1
('33x20_S-B','imballag','155_01_0054')	25	('33x20_S-B','ritocco','155_01_0054')	1
('36x20_S-B','colag','155_01_0054')	80	None	1
('36x20_S-B','rifinit','155_01_0054')	72	('36x20_S-B','colag','155_01_0054')	1
('36x20_S-B','smaltat','155_01_0054')	112	('36x20_S-B','rifinit','155_01_0054')	1
('36x20_S-B','car/scar','155_01_0054')	80	('36x20_S-B','smaltat','155_01_0054')	1
('36x20_S-B','ritocco','155_01_0054')	80	('36x20_S-B','car/scar','155_01_0054')	1
('36x20_S-B','imballag','155_01_0054')	20	('36x20_S-B','ritocco','155_01_0054')	1
('36x20_D-B','colag','155_01_0054')	40	None	1
('36x20_D-B','rifinit','155_01_0054')	40	('36x20_D-B','colag','155_01_0054')	1
('36x20_D-B','smaltat','155_01_0054')	60	('36x20_D-B','rifinit','155_01_0054')	1
('36x20_D-B','car/scar','155_01_0054')	40	('36x20_D-B','smaltat','155_01_0054')	1
('36x20_D-B','ritocco','155_01_0054')	40	('36x20_D-B','car/scar','155_01_0054')	1
('36x20_D-B','imballag','155_01_0054')	10	('36x20_D-B','ritocco','155_01_0054')	1
('36x20_D-B','imballag','155_01_0054')	10	('36x20_D-B','ritocco','155_01_0054')	1

L'algoritmo è stato testato con successo, difatti ha restituito una schedulazione degli ordini di produzione con l'obiettivo di minimizzare il makespan in soli 5.2 secondi.



7. CONCLUSIONI

Il presente studio si è posto come obiettivo quello di realizzare un algoritmo di schedulazione degli ordini di produzione minimizzando il makespan seguendo un modello matematico.

Tale obiettivo è stato possibile realizzarlo in Python grazie all'ausilio della libreria Pyomo che fornisce una buona base di partenza per la realizzazione del modello matematico. A questo sono state aggiunte diverse funzioni come:

- La possibilità di inserire i tempi di set-up quando si passa dalla produzione di un articolo all'altro;
- La possibilità di leggere i tempi sul Gantt contestualizzati alla data attuale;
- La possibilità di usare come input un formato più semplice come quello Excel o CSV rispetto il linguaggio Python;
- La possibilità di inserire un codice cliente per ogni articolo in modo tale da avere una visione più chiara di ciò che si andrà a produrre.

Una volta scritto l'algoritmo per la schedulazione è stato importante andarne a valutare le prestazioni in quanto i tempi per la risoluzione del problema devono essere contenuti al fine di poterlo utilizzare in una realtà industriale.

Come prima cosa si è andato a valutare l'impatto della varietà di articoli e del numero di macchine disponibili sul tempo totale e si è visto come a parità di macchine un aumento degli articoli causa un aumento dei tempi di calcolo significativo. Viceversa, a parità di articoli un aumento del numero di macchine genera una diminuzione del tempo il che si traduce in un maggiore semplicità nella risoluzione del problema. Sono risultati particolarmente problematiche le configurazioni con un varietà alta di articoli da produrre su un numero contenuto di macchine.

Tali risultati sono stati confermati dall'analisi DoE svolta grazie all'utilizzo di Excel che ha permesso di realizzare un modello matematico in grado di rappresentare l'andamento del tempo di calcolo in funzione del numero di articoli, numero di macchine e numero di operazioni.

Successivamente si è voluto andare a valutare la differenza dei tempi di calcolo in funzione del tipo di computer utilizzato e si è notato come passando da un computer con processore AMD Ryzen 3 ad uno con processore Intel core i7 i tempi sono diminuiti mediamente del 52%.

7.1. Sviluppi futuri

Durante la realizzazione dello schedulatore sono emerse numerose problematiche che possono essere argomento di studi futuri quali ad esempio:

- Introdurre un calendario di disponibilità delle risorse indicando giorno per giorno il numero di turni di lavoro disponibili e se il giorno è lavorativo o meno al fine di eliminare il vincolo di capacità delle risorse infinite.
- Tenere in considerazione, durante la risoluzione del problema, anche la data di consegna dell'articolo in modo da evidenziare eventuali ritardi e cercarli di diminuirli.
- Avere la possibilità di schedulare con priorità eventuali ordini non lavorati o lavorati parzialmente nel turno precedente.
- Creare un interfaccia user-friendly per l'inserimento dei dati in input

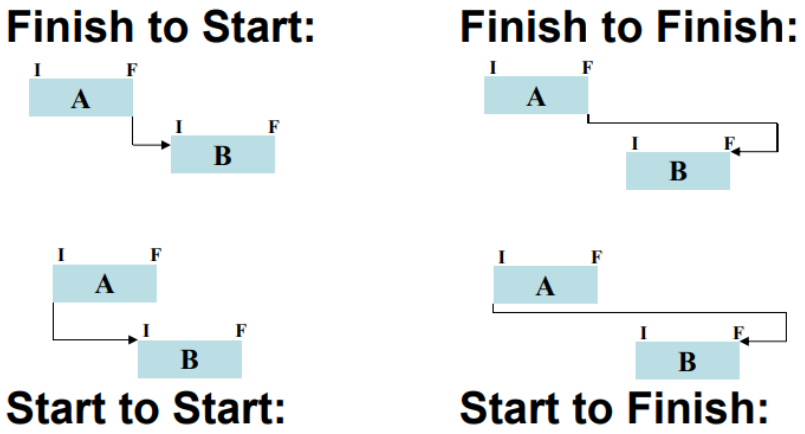
Un ulteriore sviluppo molto interessante è quello di poter associare all'algoritmo creato un algoritmo di intelligenza artificiale in modo da poter abbattere i tempi di calcolo in modo significativo in base ad esperienze pregresse.

APPENDICE 1

CREAZIONE DEL PROJECT NETWORK DIAGRAM e CRITICAL PATH METHOD

Per rappresentare un project network diagram, una volta individuate le varie task bisognerà stabilire le relazioni tecniche tra le attività stesse e possono essere di diverso tipo:

- Finish to start: l'attività successive può iniziare solo se l'attività precedente è finita
- Start to start: l'attività successive può iniziare solo se è iniziata l'attività precedente
- Finish to finish: l'attività successiva può finire solo se l'attività precedente è finita
- Start to finish: l'attività successive può finire solo se è iniziata l'attività precedente
- Lag temporali: ad ogni legame di precedenza può essere associato un ritardo o un anticipo.



È fondamentale, nella realizzazione del project network diagram, che ogni attività abbia un almeno un predecessore o un successore; dunque, ogni task deve avere una linea di connessione e non ci devono essere dei loop. Importante in questa fase è l'individuazione

del critical path: un'attività si definisce critica quando un ritardo dell'attività precedente determina un ritardo della attività successiva, l'insieme delle attività critiche determina il sentiero critico (critcal path) mentre le attività che non fanno parte di questa categoria possono ammettere uno scorrimento.

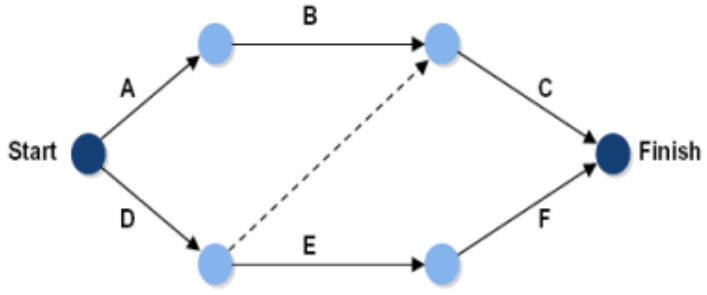
Per la realizzazione di un project network diagram è necessario avere già stipulato la Work Breakdown Structure (WBS) nella quale, a partire dal progetto da realizzare, si definiscono i vari sotto progetti e attività necessarie a garantire un buon controllo del progetto in corso di realizzazione.

L'unità di lavoro è una funzione elementare che prende il nome di Work Packege (WPs) dove per ognuna di esse va definito:

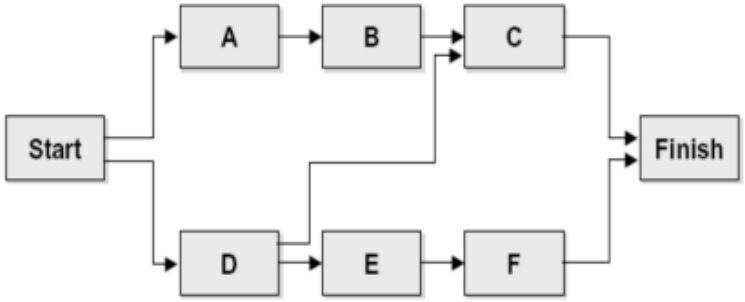
- Descrizione delle attività da svolgere, comprese anche quelle relative al controllo qualità
- Gli input attesi eventualmente da altre attività
- Gli output previsti
- Mesi uomo richiesti
- Vincoli
- Evento di chiusura

Una volta ottenute queste informazioni per ogni singola attività si potrà iniziare a redigere il Project network diagram a partire dall'attività senza vincoli di precedenza seguendo due approcci differenti:

- Activity on Arrow: dove le frecce indicano un'attività da svolgere e i nodi rappresentano degli istanti temporali



- Activity on Node: dove le frecce rappresentano i vincoli tra le varie attività che sono rappresentate sui nodi



In entrambe le metodologie di realizzazione lo scopo è quello di andare a definire il sentiero critico, ovvero la sequenza di attività che non ammette scorrimento o ritardi sulla realizzazione, per farlo si segue un approccio su 2 steps:

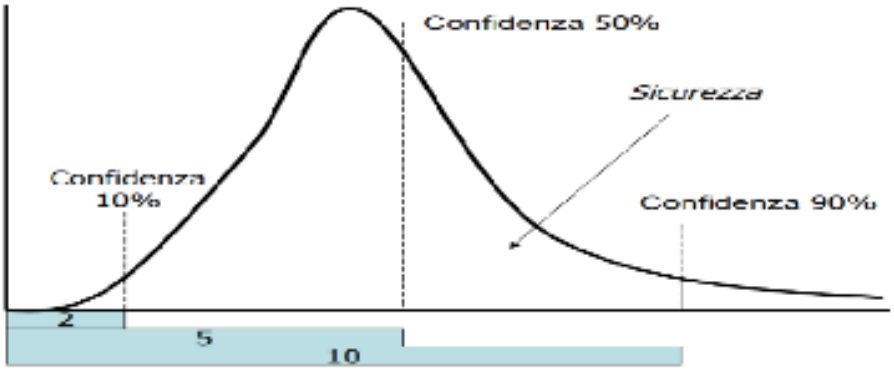
1) Forward pass: usato per calcolare le date di inizio al più presto (ES) e di fine al più presto (EF) delle singole attività. Partendo dalla prima attività in cui si suppone come istante iniziale ES=0 si andrà a calcolare la data di fine al più presto come: EF = ES + durata. La data di EF della prima attività sarà di conseguenza la data di ES dell'attività che lo segue. Si itera il seguente metodo fino al raggiungimento dell'ultima attività
2) Backward pass: usato per calcolare le date di inizio al più tardi (LS) e fine al più tardi (LF) di tutte le attività. Queste sono le date più posticipate che si possono fissare senza impattare le date di fine del progetto. Il processo per il calcolo di questi due valori è simile a quello visto nel primo step. Si parte dall'ultima attività nella quale si suppone che il valore di LF = EF e si ripercorre il reticolo al ritroso fino ad arriviare alla prima attività. Qui il valore di LS della singola attività sarà LS = LF – durata. La data di LF di una specifica attività sarà pari al valore minimo di LS di una delle attività a monte.

Questa metodologia ci permette di andare ad individuare il sentiero critico come la sequenza di attività che hanno valore di scorrimento = 0. (scorrimento ottenuto come LS – ES)

Intorno al percorso critico si svolgeranno altre attività che invece ammettono di subire una variazione in termini di data di inizio o di fine. Se scegliessimo di effettuare delle ispezioni a intervalli per la verifica della corretta esecuzione delle attività del progetto potremmo incorrere nella situazione per cui, in un certo momento, un'attività del percorso critico non sia giunta al livello di completamento progettato dunque bisognerà agire repentinamente andando a prendere da un'attività non facente parte del critical path (che si sta svolgendo parallelamente) le risorse da far fluire verso la prima attività (ovviamente nell'ipotesi che le risorse siano condivisibili e compatibili). Quest'operazione però avrà un effetto ritardante sull'attività che non fa parte del percorso critico ma è comunque accettabile se non vada oltre la data di late finish concessa.

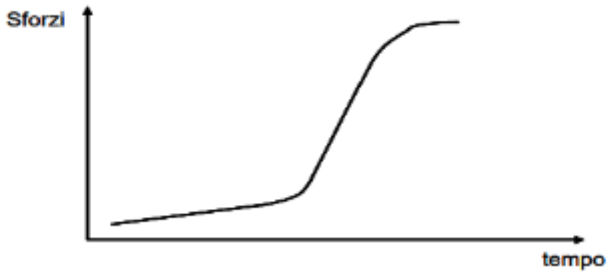
L'applicazione di questa metodologia richiede la presenza del project network diagram dove sono evidenziati eventuali legami di dipendenza tra le attività e della WBS e dove sono rappresentate tutte le task che devono svolgersi con i relativi tempi. Questa tecnica richiede una base storica di dati legati al tempo necessario per svolgere una singola attività.

Un momento critico di questa metodologia è la fase di contrattazione delle risorse con i diversi gestori delle attività di un progetto: storicamente, e in base all'esperienza, chi coordina una certa attività del progetto, conoscendo quelli che sono i tempi che statisticamente sono necessari al completamente dell'operazione, tenderà a domandare al responsabile o al project manager di avere a disposizione una quantità di risorse tempo maggiore di quella effettiva così da introdurre un certo fattore di sicurezza tale per cui al 90% si avrà la certezza di concludere le operazioni nei tempi previsti. Ma se ogni coordinatore domanderà questo tipo di richieste sarà molto probabile che la conclusione del progetto vedrebbe uno slittamento non ammissibile.

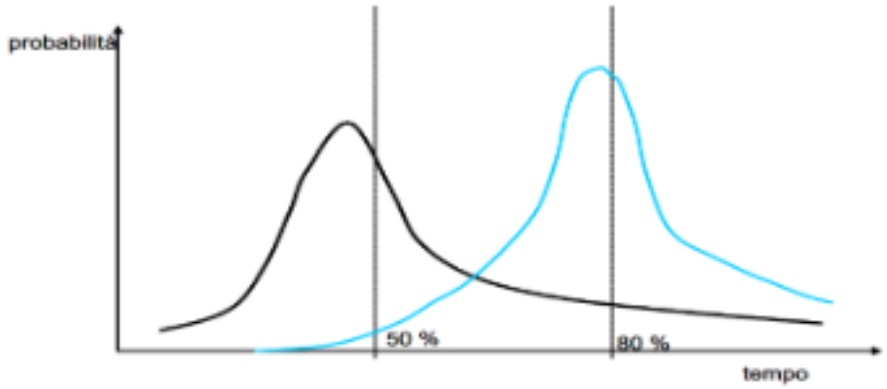


(l'immagine riporta la variazione della durata sulle ascisse in funzione del coefficiente di sicurezza richiesto che, solitamente, si attesta al 90%)

Ovviamente si vuole aggiungere tempo di sicurezza poiché si terrà conto dell'esperienza più negativa che si è avuto in passato, dunque, pure sapendo che la stima proposta subirà un taglio, si chiederà una quantità di tempo che è comunque superiore anche dopo il taglio. Nonostante tutta questa sicurezza introdotta nella maggior parte dei casi si finisce per completare in ritardo il progetto a causa dell'effetto chiamato "sindrome dello studente" che dimostra come le risorse, sicure del loro tempo a disposizione, procederanno per la maggior parte del tempo a lavorare a ritmi ridotti per poi aumentare la produttività in prossimità della data di consegna.



La motivazione di questa richiesta di sicurezza può essere motivata anche con la "legge di Parkinson": chi tende a terminare in anticipo un'operazione si vedrà tagliare il tempo a disposizione la prossima volta dato che ha dato dimostrazione che ne è capace, dunque ecco perché si usano sempre tutte le risorse a disposizione e si chiedono delle sovrastime.



Si creano così delle situazioni conflittuali tali per cui si vorrebbe garantire l'esecuzione in sicurezza dei vari task ma in contempo assicurare tempi di consegna brevi ai clienti.

Una soluzione molto utilizzata è quella di ridurre dal 90% al 50% l'intervallo di confidenza richiesto per lo svolgimento delle varie attività e quel tempo sottratto aggiungerlo ad un safty buffer da cui possono attingere le varie task che sono in ritardo.

APPENDICE 2

Codice apertura file:

```
from openpyxl import load_workbook
def lett_file():
    wb = load_workbook(filename='input.xlsx', read_only=True)
    ws = wb['Operazioni']   #lettura del foglio
    #serie di liste vuote dove andare ad inserire i valori delle relative celle
    name_task=[]
    time_task=[]
    order_task=[]
    domanda_task=[]
    cod_task=[]
    #liste necessarie per inserire il valore della cella e poi modificarlo o
    #spostarlo secondo le necessità
    a=[]
    b=[]
    for row in ws.rows:
        for cell in row:
            a.append(cell.value)

        name_task.append(a[0])
        time_task.append(a[1])
        order_task.append(a[2])
```

```
        domanda_task.append(a[3])
        a=[]
    #con questa serie di comandi si elimina il primo valore della lista in modo tale da evitare
    # la prima riga del foglio
    name_task.pop(0)
    time_task.pop(0)
    order_task.pop(0)
    domanda_task.pop(0)

    k= len( time_task )
    name_task.pop(k-1)
    time_task.pop(k-1)
    order_task.pop(k-1)
    domanda_task.pop(k-1)

    tasks={}
    for i in range(len(name_task)):
        product_machine = eval(name_task[i])
        name_task[i]= product_machine
        cod_task.append(name_task[i][2])
        time_task[i]= int(time_task[i])
```

```
        domanda_task[i] = int (domanda_task[i])
        time_task[i] = time_task[i] * domanda_task[i]
        if order_task[i] != 'None':
            order_prec = eval(order_task[i])
            order_task[i] = order_prec
        elif order_task[i] == 'None':
            # order_task.pop(i)
            nope= None
            order_task[i] = nope
        tasks [name_task[i]] = {"dur":time_task[i], "prec":order_task[i], "tsetup":{}}
        ct = tuple(set(cod_task))
    return(tasks , ct)

```

Codice lettura matrice set-up:

```
#lettura matrice set-up
from openpyxl import load_workbook
nome_file = load_workbook(filename='input.xlsx', read_only=True)
# nome_foglio1 = nome_file['M1']#si crea un foglio per ogni tipologia di macchina
# nome_foglio2 = nome_file['M2']
from apertura_file import lett_file
def lett_setup(nome_file,nome_foglio,ct,task):
    a=[]
    b=[]
    name_art=[]
    name_art2=[]
    #creazione dizionario con le chiavi giuste
    for row in nome_foglio.rows:
        for cell in row:
            a.append(cell.value)
        name_art.append(a[0])
        a=[]
    name_art.pop(0)
    #in questo modo salvo tutti i nomi degli articoli all'interno di una lista
    matrix_setup={}
    #creo il dizionario per i tempi di set-up
```

```
    for i in range(len(name_art)):
        product_machine = eval(name_art[i])
        name_art[i]= product_machine
        #serve per trasformare in stringa il nome in modo da poterci lavorare
        ct= list (ct)
        for j in range(len(ct)):
            c= list(name_art[i])
            c.append(ct[j])
            c = tuple(c)
            for keys in task:
                #serve per vedere se il valore di c esiste all'interno di task
                if keys == c:
                    name_art2.append(c)
                    matrix_setup[c] = {}


    #riempimento dei sotto dizionari
    for row in nome_foglio.rows:
        for cell in row:
            b.append(cell.value)
        if b[0] == 'A/DA':
```

```
            for i in range(len(b)):
                b.pop(0)

        if len(b)== len(name_art2) :
            b[0]= eval(b[0])
            for i in range(len(b)):
                if len(b) == len(name_art2):
                    next_one = b[0]
                    imp = []
                    for h in range(len (ct)):
                        next_one = list(next_one)
                        next_one.append(ct[h])
                        next_one = tuple(next_one)
                        if (next_one in name_art2) == True:
                            imp.append(next_one)
                        if imp != []:
                            if (imp[0] in name_art2) == True:
                                if len(b) >1 :
                                    for keys in matrix_setup:
                                        if keys[0] == c[0] and keys[1] == c[1]:
                                            matrix_setup[keys][imp[0]] = matrix_setup[c][imp[0]]
```

```
                                        else:
                                            matrix_setup[keys][imp[0]] = b[i+1]
                                            c= keys
                                            b.pop(0) #matrice creata
                        next_one = list(next_one)
                        next_one.pop(2)
                        if len(imp) == 2:
                            for keys in matrix_setup:
                                if keys[0] == c[0] and keys[1] == c[1]:
                                    matrix_setup[keys][imp[1]] = matrix_setup[keys][imp[0]]
                                else:
                                    matrix_setup[keys][imp[1]] = matrix_setup[keys][imp[0]]
                                    c= keys

        b=[]
    return(matrix_setup)
```

Codice principale:

```python
import time
start = time.time()
import matplotlib.pyplot as plt
import matplotlib as mpl
import pandas as pd
from pyomo.environ import *
from pyomo.gdp import *
from openpyxl import load_workbook

from Gantt import visualize
from mtrx_setup import lett_setup
from apertura_file import lett_file
tasks , ct=lett_file()

nome_file = load_workbook(filename='input.xlsx', read_only=True)
nome_foglio1 = nome_file['M1'] #caricamento matrice setup per la prima macchina
nome_foglio2 = nome_file['M2']
tsetup1= lett_setup(nome_file,nome_foglio1,ct , tasks)
tsetup2= lett_setup(nome_file,nome_foglio2 ,ct,tasks)
tsetup= {**tsetup1, **tsetup2} #unire due dizioanri
for keys in tasks:
    for items in tsetup:
        if items == keys:
            tasks[keys]['tsetup'] = tsetup[items]

def jobshop_model(tasks):

    model = ConcreteModel()
    # tasks is a two dimensional set of (j,m) constructed from the dictionary keys
    model.tasks = Set(initialize = tasks.keys(), dimen=3)
```

```python

    # the set of jobs is constructed from a python set
    model.JOBS = Set(initialize = list(set([j for (j,m,z) in model.tasks])))

    # set of machines is constructed from a python set
    model.MACHINES = Set(initialize = list(set([m for (j,m,z) in model.tasks])))

    model.COD = Set(initialize = list(set([z for (j,m,z) in model.tasks])))

    model.tsetup=Param(model.tasks, initialize=lambda model, j, m,z: tasks[(j,m,z)]['tsetup'])

    # the order of tasks is constructed as a cross-product of tasks and filtering
    model.TASKORDER = Set(initialize = model.tasks * model.tasks, dimen=6,
        filter = lambda model, j, m,z, k, n,y: (k,n,y) == tasks[(j,m,z)]['prec'])

    # the set of disjunctions is cross-product of jobs, jobs, and machines
    model.DISJUNCTIONS = Set(initialize = model.JOBS * model.JOBS * model.MACHINES*model.COD*model.COD, dimen=5,
        filter = lambda model, j, k, m,z,y : j < k and (j,m,z) in model.tasks and (k,m,y) in model.tasks )

    # load duration data into a model parameter for later access
    model.dur = Param(model.tasks, initialize=lambda model, j, m,z: tasks[(j,m,z)]['dur'])

    model.tsetup=Param(model.tasks, initialize=lambda model, j, m, z: tasks[(j,m, z)]['tsetup'])

    # establish an upper bound on makespan
    ub = sum([model.dur[j,m,z] + 100 for (j,m,z) in model.tasks])

    model.makespan = Var(bounds=(0, ub))

    model.start = Var(model.tasks, bounds=(0, ub))

```

```
    model.objective = Objective(expr = model.makespan, sense = minimize)

    model.finish = Constraint(model.tasks, rule=lambda model, j, m,z:
        model.start[j,m,z] + model.dur[j,m,z] <= model.makespan)

    model.preceding = Constraint(model.TASKORDER, rule=lambda model, j, m, z,k,n, y:
        model.start[k,n,y] + model.dur[k,n,y] <= model.start[j,m,z])

    model.disjunctions = Disjunction(model.DISJUNCTIONS, rule=lambda model,j,k,m,z,y:
        [model.start[j,m,z] + model.dur[j,m,z] + model.tsetup[j,m,z][k,m,y] <= model.start[k,m,y],
         model.start[k,m,y] + model.dur[k,m,y] + model.tsetup[k,m,y][j,m,z] <= model.start[j,m,z]])


    TransformationFactory('gdp.hull').apply_to(model)
    return model

def jobshop_solve(model):
    SolverFactory('cbc').solve(model)
    results = [{'Job': (j,z),
                'Machine' : m,
                'Prio' :z,
                'Start':  model.start[j, m,z](),
                'Finish':  model.start[j, m,z]() + model.dur[j,m,z],
                'Duration': model.dur[j,m,z] ,
                }
               for j,m,z in model.tasks]
    return results


def jobshop(tasks):
    return jobshop_solve(jobshop_model(tasks))
```

```
results = jobshop(tasks)
results

scheda = pd.DataFrame(results)

#converto i tempi con date reali
for i in range(len(results)):
    results[i]['Start'] = {'Anno': 2022, 'Mese': 5, 'Giorno':6 ,'Ora':8 , 'Minuti': int(results[i]['Start'])}
    results[i]['Finish'] = {'Anno':2022,'Mese': 5, 'Giorno':6 ,'Ora':8 , 'Minuti': int(results[i]['Finish'])}
    results[i]['Duration'] = {'Anno':2022, 'Mese': 5, 'Giorno':6 ,'Ora':0 , 'Minuti': int(results[i]['Duration'])}
    #cambio minuti --> ore
    for items in results[i]:
        if items == 'Start' or items == 'Finish' or items == 'Duration':
            while results[i][items]['Minuti'] >= 60 :
                results[i][items]['Ora'] = results[i][items]['Ora'] +1
                results[i][items]['Minuti'] = int(results[i][items]['Minuti']) - 60
            while results[i][items]['Ora'] >= 24:
                results[i][items]['Ora'] = results[i][items]['Ora'] -24
                results[i][items]['Giorno'] = results[i][items]['Giorno'] + 1
            for keys in results[i][items]:
                if results[i][items][keys] < 10:
                    results[i][items][keys] = str(results[i][items][keys])
                    results[i][items][keys] = '0' + results[i][items][keys]

            results[i][items] =  '2022'+ '-' + str(results[i][items]['Mese']) + '-' + str(results[i][items]['Giorno']) + '\
schedule = pd.DataFrame(results)
schedule.to_csv('schedule.csv',index=False)


```

```python
import pandas as pd
import matplotlib.pyplot as plt
import datetime as dt
#Read Data from schedule.csv
df =pd.read_csv('schedule.csv')
df.head()
#Convert dates to datetime format
df.Start=pd.to_datetime(df.Start)
df.Finish=pd.to_datetime(df.Finish)
df.Duration=pd.to_datetime(df.Duration)

df=df.sort_values(by='Start', ascending=True)

visualize(df,scheda)
end= time.time()
print( "il tempo di calcolo è.... " , end- start )
plt.show()
```

Codice Gantt:

```python
def visualize(df,scheda):
    # df = pd.DataFrame(data_xm)
    JOBS = sorted(list(df['Job'].unique()))
    MACHINES = sorted(list(df['Machine'].unique()))
    makespan = df['Finish'].max()

    bar_style = {'alpha':1.0, 'lw':25, 'solid_capstyle':'butt'}
    text_style = {'color':'white', 'weight':'bold', 'ha':'center', 'va':'center'}
    colors = mpl.cm.Dark2.colors

    scheda.sort_values(by=['Job', 'Start'])
    scheda.set_index(['Job', 'Machine'], inplace= True)
    df.set_index(['Job', 'Machine'], inplace=True)

    fig, ax = plt.subplots(2,1, figsize=(12, 5+(len(JOBS)+len(MACHINES))/4))
    from ast import literal_eval
    for jdx, j in enumerate(JOBS, 1): # j= nome art ldx un contatore
        for mdx, m in enumerate(MACHINES, 1): #m = nome macchin mdx contatore
            if (j,m) in df.index:

                xs = df.loc[(j,m), 'Start'] #xs = inizio dell'asse x per ogni attività
                xf = df.loc[(j,m), 'Finish'] #xf fine dell'asse x per ogni attività
                c= literal_eval(j)
                xs1= scheda.loc[(c,m), 'Start']
```

```
                xf1=scheda.loc[(c,m), 'Finish']
                xm =(xs1 +xf1)/2
                data_xm={'Mese': 5, 'Giorno':6 ,'Ora':8 , 'Minuti':xm}
                data_xm['Minuti'] = int(data_xm['Minuti'])
                while data_xm['Minuti'] >= 60 :
                    data_xm['Ora'] = data_xm['Ora'] +1
                    data_xm['Minuti'] = int(data_xm['Minuti']) - 60
                while data_xm['Ora'] >= 24:
                    data_xm['Ora'] = data_xm['Ora'] -24
                    data_xm['Giorno'] = data_xm['Giorno'] + 1
                for keys in data_xm:
                    if data_xm[keys] < 10:
                        data_xm[keys] = str(data_xm[keys])
                        data_xm[keys] = '0' + data_xm[keys]

                data_xm['Minuti'] = str(data_xm['Minuti'])

                dx=  str('2022'+ '-' + str(data_xm['Mese']) + '-' + str(data_xm['Giorno']) + '\t' +
                    str(data_xm['Ora']) + ':' + str(data_xm['Minuti']) + "." + "00")

                data_xm=pd.to_datetime(dx)
                ax[0].plot([xs, xf], [jdx]*2, c=colors[mdx%7], **bar_style)
                ax[0].text( data_xm ,jdx, m, **text_style)
                ax[1].plot([xs, xf], [mdx]*2, c=colors[jdx%7], **bar_style)
                ax[1].text( data_xm, mdx, j, **text_style)

    ax[0].set_title('Job schedule')
    ax[0].set_ylabel('Job')
    ax[1].set_title('Machine schedule')
    ax[1].set_ylabel('Machine')
```

```
    for idx, s in enumerate([JOBS, MACHINES]):# itera per i due grafici
    #idx = 0    ---- s = ['articolox' , 'articoloy' , articoloz']
    #idx = 1    ---- s = ['M1' , 'M2']
        ax[idx].set_ylim(0.5, len(s) + 0.5)
        ax[idx].set_yticks(range(1, 1 + len(s)))
        ax[idx].set_yticklabels(s)
        ax[idx].plot([makespan]*2, ax[idx].get_ylim(), 'r--')
        ax[idx].grid(True)

    fig.tight_layout()
    return()
```

BIBLIOGRAFIA

[1] Materiale del corso di 'Gestione degli impianti industriali' del Prof. Maurizio Bevilacqua.

[2] Haojie Ding, Xingsheng Gu, 'Hybrid of human learning optimization algrothm and particle swarm optimization algorithm with scheduling strategie for flexible job-shop scheduling problem'.

[3] Philipp Winklehner, Viktoria A. Hauder, 'Flexible job-shop scheduling with release dates, deadlines and sequence dependent setup times'.

[4] NaimingXie NanleiChen, 'Flexible job scheduling problem with interval grey processing time'.

[5] Hongtao Tang, Bo Fang, Rong Liu, Yibing li, Shunsheng Guo, 'Hybrid teaching and learning-based optimization algorithm for distributed sand casting job-shop scheduling problem'.

SITOGRAFIA

[1] https://it.wikipedia.org/wiki/Teoria_della_schedulazione

[2] https://pyomo.readthedocs.io/en/latest/modeling_extensions/gdp/modeling.html