UNIVERSITÀ DEGLI STUDI DI PADOVA

Dipartimento di Tecnica e Gestione dei Sistemi Industriali

Corso di Laurea Magistrale in Ingegneria Gestionale

Tesi di Laurea

Schedulazione della produzione: ottimizzazione basata sulla tecnica della programmazione a vincoli

Relatore *Ch. mo Prof. Roberto Panizzolo*

Laureando *Girlanda Stefano*

Anno Accademico 2016-2017

Sommario

La schedulazione della produzione è un processo che si occupa di assegnare le risorse produttive alle operazioni da eseguire per fornire ai reparti produttivi un piano dettagliato in modo da ottimizzare le performance di processo.
La schedulazione della produzione o production scheduling si colloca in un Manufacturing Planning and Control System (MPCS) all'interno della fase di Shop-Floor Scheduling and Control (SFC). Le attività tipiche della schedulazione della produzione rientrano in 4 categorie: assegnazione dei carichi, sequenziamento delle attività, tempificazione delle attività, monitoraggio e controllo.
I metodi utilizzati per affrontare i problemi di scheduling sono numerosi, e possono essere collocati in due macro categorie: i metodi euristici e i metodi esatti. I metodi euristici si basano su l'utilizzo di regole di priorità, come la weighted shortest processing time first o la earliest due date, o algoritmi più complessi come l'approccio bottleneck, la regola di Johnson per i flow shop o lo shifting bottleneck per i job shop. I metodi esatti comprendono la programmazione matematica e la programmazione a vincoli, che a partite da formulazioni simili ma elaborazioni diverse, permettono di ottenere una soluzione ottima. La programmazione a vincoli permette di risolvere contraint satisfaction problems (CSP), in cui si deve assegnare un valore numerico a delle variabili soggette ad una serie di vincoli. Se è presente una funzione obiettivo da ottimizzare si parla di contraint optimization problems (COP). Tale metodo si è dimostrata molto efficace nella risoluzione di problemi di scheduling. La programmazione a vincoli sfrutta il backtracking associato a una serie di tecniche di tipo look back o look ahead volte a ridurre lo spazio di ricerca, come la propagazione dei vincoli o il backjumping.
L'utilizzo di un software prodotto dall'azienda MBM che sfrutta le potenzialità della programmazione a vincoli per la schedulazione della produzione, ha permesso al gruppo Danieli di ridurre i ritardi di consegna e di migliorare le performance aziendali.

Indice

Lista figure

Introduzione

Il seguente lavoro approfondisce il tema della schedulazione della produzione e dei principali metodi utilizzati per la risoluzione di tale problema, soffermandosi in particolare sulla programmazione a vincoli e presentando un caso studio in cui l'applicazione di tale metodo al processo produttivo ha portato miglioramenti in termini di performance.

Nel primo capitolo viene data una definizione di schedulazione della produzione la quale viene poi collocata all'interno di un manufacturing production and control system (MPCS) tipico aziendale proposto da Vollmann, Berry e Whybark in Manufacturing planning and control systems. Vengono poi presentate le attività tipiche del processo di schedulazione e la notazione a tre campi utilizzata per classificare tali problemi (Pinedo 2008). Il capitolo termina con la classificazione dei metodi di risoluzione, come le tecniche euristiche e le tecniche di ottimizzazione che sfruttano la programmazione matematica e a vincoli, per le quali vengono messe in luce le principali differenze.

Nel secondo capitolo, seguendo la classificazione a tre campi definita nel capitolo precedente, vengono presentati i problemi di scheduling a macchina singola, a macchine in parallelo, flow shop, job shop e open shop e vengono esposti alcuni possibili algoritmi euristici per la loro risoluzione (Pinedo 2008) (Bruno 2012).

Nel terzo capitolo viene presentata la programmazione a vincoli come tecnica di ottimizzazione utilizzabile per la risoluzione dei problemi di scheduling. Vengono esposti gli algoritmi che vengono utilizzati dai motori di ottimizzazione che sfruttano la programmazione a vincoli per ottenere la soluzione di un problema, tra cui il backtracking e la propagazione dei vincoli (Kumar 1992) (Russel e Norvig 2009).

Il quarto capitolo presenta un caso studio dove l'utilizzo di un software per la schedulazione della produzione che sfrutta la programmazione a vincoli prodotto dall'azienda MBM e applicato negli stabilimenti del gruppo Danieli ha portato a una diminuzione dei ritardi di consegna e dei tempi di set up.

Capitolo 1 - La schedulazione della produzione

In questo capitolo viene introdotto il tema della schedulazione della produzione. Si parte prima dando una definizione di schedulazione della produzione specificandone le caratteristiche e mostrando dove si colloca all'interno di un sistema aziendale di pianificazione e controllo della produzione (MPCS). Dopodiché vengono delineate le attività che compongono lo scheduling e viene data una classificazione delle tipologie di problemi di scheduling tramite la notazione a tre campi. Si delineano poi le tecniche di risoluzione più usate nello scheduling della produzione che verranno poi approfondite nel capitolo 2.

1.1 La schedulazione all'interno del sistema di pianificazione della produzione aziendale

La schedulazione è un processo decisionale che consiste nell'allocazione delle risorse alle attività su un intervallo di tempo in modo da ottimizzare uno o più obiettivi di performance come costo, tempi di set up, ritardi di consegna o utilizzo dei macchinari. Le risorse possono essere costituite da macchinari in un centro di lavoro, operatori, unità di calcolo all'interno di un computer o piste di atterraggio in un aeroporto. Le attività possono essere operazioni all'interno di un processo produttivo, atterraggi e decolli in un aeroporto, fasi di un progetto, pazienti in un ospedale o elaborazioni da eseguire in un calcolatore. La schedulazione della produzione in particolare si occupa quindi di assegnare le risorse come macchinari e operatori alle operazioni o compiti da eseguire, in modo da ottimizzare le performance di processo.

Il livello di dettaglio è elevato poiché si va a specificare quale operazione dovrà effettuare la singola risorsa in un dato momento e per quanto tempo. La necessità di un livello di dettaglio elevato implica un orizzonte temporale delle decisioni limitato al giorno o alla settimana.

In figura 1.1.1 viene fornito un esempio di sistema di pianificazione e controllo della produzione aziendale, o manufacturing planning and control system (MPCS), proposto da Vollmann, Berry, Whybark, 1997.

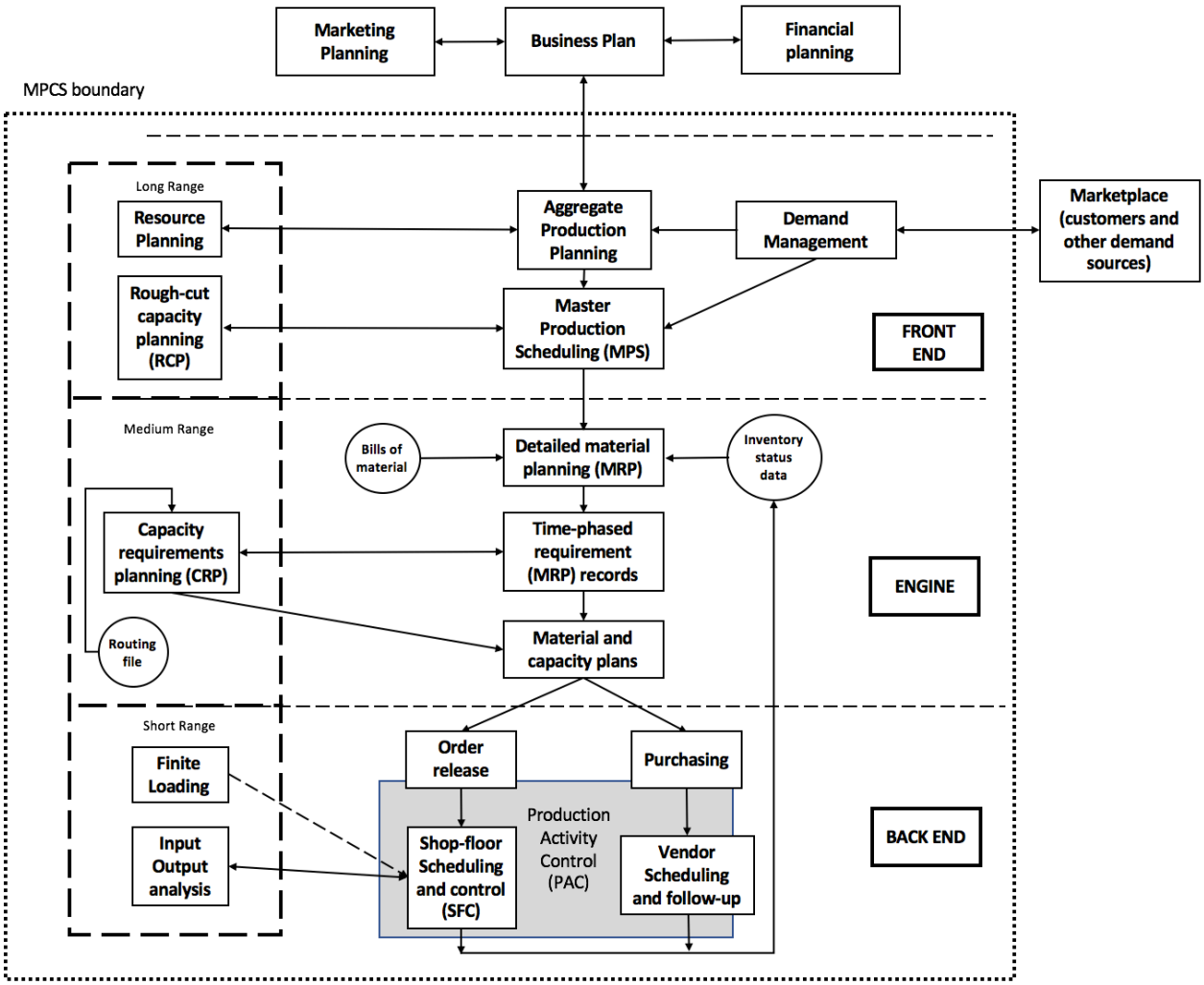


Fig. 1.1.1 Schema Manufacturing Planning and Control System (adattato da Vollmann, Berry, Whybark, 1997)

L'MPCS si divide in tre parti; la prima, o front end, costituisce l'insieme di tutte le attività per l'impostazione della direzione complessiva, ossia per stabilire gli obiettivi aziendali per la pianificazione e controllo della produzione. Le attività di front end operano su un orizzonte temporale elevato e di conseguenza è necessario

che le informazioni siano aggregate e ad un basso livello di dettaglio. Demand management si occupa di prevedere la domanda del consumatore per il prodotto finale al fine di coordinare tutte le attività che devono soddisfare questa domanda.
Il piano aggregato di produzione (aggregate production planning) consiste nel determinare il livello di produzione necessario a soddisfare la domanda attesa futura, bilanciandolo con la capacità attuale, in modo da perseguire gli obiettivi stabiliti dalla strategia aziendale, o business plan. È necessario in questa fase aggregare le informazioni da processare in modo da far fronte all'incertezza, per questo il piano aggregato di produzione opera di solito su un'unità aggregata di produzione ossia un prodotto con caratteristiche medie tra quelli venduti dall'azienda in termini di peso, volume, valore, tempo di produzione. Il piano aggregato di produzione viene poi disaggregato nel master production schedule (MPS) che prevede di stabilire per ogni prodotto la quantità da produrre in ogni periodo. Il piano delle risorse, o resource planning, è il piano più aggregato e di lungo termine per le decisioni di pianificazione della capacità. Esso consiste nel determinare se la capacità produttiva attuale è sufficiente per far fronte alla domanda attesa o se sono necessari ampliamenti in termini di macchine, spazio, uomini, capitale etc.
Il rough-cut capacity planning (RCCP) permette di pianificare in modo grossolano la capacità necessaria ad attuare tale piano. Esso è la versione più disaggregata del resource planning e necessita delle informazioni fornite dal MPS.
La seconda parte è quella di engine; l'orizzonte temporale su cui vengono prese le decisioni comincia a diminuire (medio-lungo termine) mentre il livello di dettaglio aumenta. La fase di engine riceve come input il prodotto finale della fase di front end, ossia l'MPS. L'MPS alimenta direttamente il detailed material planning che rappresenta il sistema centrale nella fase di engine. Molte aziende utilizzano la logica MRP nella pianificazione dettagliata dei materiali. L'MRP, o material requirements planning, determina periodo per periodo i piani per tutti i componenti e materie prime necessarie a produrre i prodotti definiti nell'MPS.

L’MRP utilizza come input, oltre all’ MPS, anche le distinte base dei prodotti, o bills of material, e i livelli attuali di scorte, o inventory status data. Le distinte base dei prodotti forniscono indicazioni sulle parti necessarie a produrre un certo prodotto e quindi permettono all’MPS di esplodere il prodotto finale per determinare il fabbisogno di componenti e materie prime. I livelli attuali di scorte sono necessari poiché è necessario sapere quante delle parti da produrre o acquistare ho già ad esempio a magazzino, o in produzione o ordinate.
I dati dell’MRP permettono quindi di costruire un piano dettagliato e time-phased dei fabbisogni aziendali. L’MRP quindi traduce i piani complessivi di produzione in informazioni dettagliate necessarie per sviluppare i piani di capacità e per guidare le attività di produzione.
Utilizzando l’MRP è inoltre possibile strutturare un piano dettagliato della capacità necessaria (capacity requirement plan, o CRP). La CRP riceve inoltre come input informazioni su work-in-process, routing, ordini pianificati, in modo da determinare i fabbisogni di capacità necessari. Per resource planning, RCP e CRP le frecce sono bidirezionali poiché è necessario che ci sia una corrispondenza tra la capacità necessaria a eseguire un determinato piano dei materiali e la capacità disponibile per eseguire i piani.
L’ultima parte è quella di back end che rappresenta le attività che si occupano dell’esecuzione dei piani dettagliati dei materiali (MRP). Le attività operano su breve periodo e con grado elevato di dettaglio. Order release è la fase che autorizza il lancio degli ordini e fornisce la documentazione necessaria.
Purchasing consiste nella gestione degli acquisti dove sono stabiliti il flusso informativo, le relazioni, i termini e le condizioni di acquisto.
Tuttavia la parte principale della fase di back end è rappresentata dalla production activity control (PAC). La PAC si occupa dell’esecuzione, monitoraggio e controllo dei piani di produzione definiti ai livelli precedenti. L’attività di vendor scheduling and follow-up si occupa di mantenere gli ordini di acquisto allineati con le date di consegna stabilite dal piano dei materiali (MRP). L’altra attività da cui è costituita la PAC è lo shop-floor scheduling and control (SFC).

È proprio la fase di shop-floor scheduling and control quella di cui fa parte la schedulazione della produzione. L'SFC si occupa di stabilire le priorità per gli ordini di produzione per ogni centro di lavoro in modo che gli ordini possano essere pianificati al fine di ottimizzare degli obiettivi di performance.
Tale fase si occupa di emettere gli ordini, sequenziarli e assegnarli ad un livello di dettaglio di singola macchina o operatore.
L'attività di finite loading è una decisione sulla capacità ma è visibile inoltre come una tecnica di shop-floor scheduling poiché consiste nell'allocare ogni risorsa di capacità produttiva ai singoli ordini di produzione. Essa perciò può essere vista meglio all'interno delle attività di SFC e rappresenta in maniera chiara la relazione fra scheduling e disponibilità di capacità.
Dunque l'attività di schedulazione della produzione è collocabile in un sistema MPCS aziendale all'interno della fase di shop-floor scheduling and control (SFC) e si pone l'obiettivo di guidare le attività operative per raggiungere i piani elaborati ai livelli precedenti.
Infine l'input output analysis si occupa di monitorare il consumo attuale di capacità produttiva e l'avanzamento degli ordini in modo da indicare la necessità di modifica dei piani in caso le performance dello shop-floor si discostino da quelle attese (Vollmann, Berry e Whybark 1997).

1.2 Attività che costituiscono la schedulazione della produzione

Il problema della schedulazione si occupa quindi di, dati una serie di ordini da processare in un certo sistema produttivo, assegnarli e sequenziarli, rispettando determinati vincoli, in modo che siano ottimizzati dei criteri di performance.
Le attività tipiche della schedulazione della produzione rientrano in 4 categorie (Slack, et al. 2013):

- assegnazione dei carichi

- programmazione con le date di inizio e fine di ciascuna operazione (spesso viene utilizzato il diagramma di Gantt)
- definizione della sequenza dei lavori
- monitoraggio e controllo

L'assegnazione dei carichi consiste nell'assegnazione della quantità di lavoro alle singole parti del processo produttivo. Essa può tener conto della capacità del processo oppure non tenerne conto; nel primo caso si parla di assegnazione a capacità finita, nel secondo caso si parla di assegnazione a capacità infinita.
Il caricamento, ossia l'assegnazione, a capacità finita alloca il carico al centro di lavoro entro un certo limite predefinito. Questo limite è determinato dalla capacità massima del centro di lavoro che definisce un vincolo al caricamento. Il caricamento a capacità infinita assegna i carichi ai centri di lavoro non limitandone l'accettazione (ad esempio un pronto soccorso di un ospedale non può rifiutare i pazienti da visitare).
Oltre all'assegnazione dei carichi di lavoro è necessario definire la sequenza dei lavori da eseguire, ossia il sequencing, decidendo quali lavori eseguire prima e quali dopo.
Il monitoraggio e controllo infine permette di confrontare lo stato attuale delle operazioni con quello programmato e in caso di intervenire con una riprogrammazione. In questa fase risulta utile la possibilità di effettuare analisi di tipo what if, ossia, cosa succede se modifico la schedulazione in qualche maniera.
L'output del processo di schedulazione è il piano schedulato il quale contiene tutte le informazioni relative all'utilizzo delle risorse produttive e il programma di lavoro di ogni reparto. La schedulazione deve operare ad un livello di dettaglio più elevato possibile e deve tener conto dei vincoli, presenti anche a livello di singola macchina, come capacità produttive, calendari aziendali, turni, precedenza fra attività, indisponibilità di risorse per un determinato periodo e date di consegna richieste.

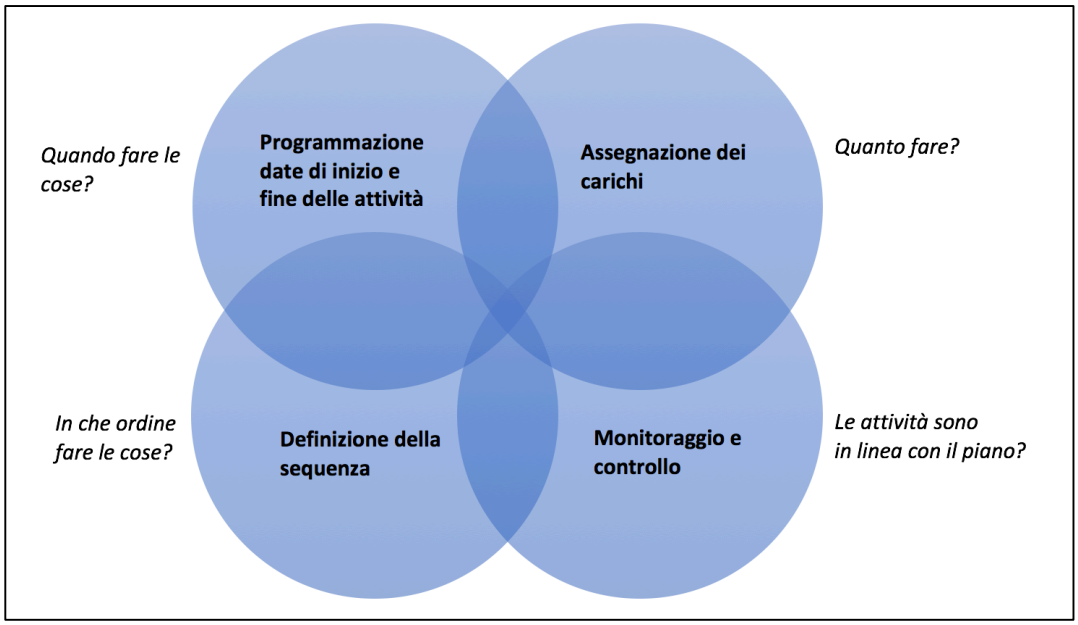


Fig. 1.2.1 I meccanismi critici della pianificazione e controllo (Slack, et al. 2013)

1.3 Il problema dello scheduling e notazione a tre campi

Le prime forme di schedulazione della produzione comparvero nel tardo 1800 e consistevano semplicemente nel listare quando un ordine doveva iniziare o per quando era da consegnare, senza nessuna informazione riguardo la durata delle singole operazioni. Con l'aumento della complessità delle aziende cominciarono a nascere i primi uffici per il controllo della produzione. Il problema fu affrontato per la prima volta in maniera formale da Henry Gantt, il quale creò un grafico innovativo per il controllo della produzione. Tale grafico permetteva di confrontare la performance attuale con la performance pianificata e quindi di capire se le attività erano in anticipo o in ritardo rispetto al programmato.
I diagrammi di Gantt sono ancora oggi largamente utilizzati come strumento di rappresentazione efficace per la pianificazione e il monitoraggio della produzione. Il diagramma è costituito da un asse temporale orizzontale e da un asse verticale con rappresentate tutte le attività da svolgere. Ogni operazione è rappresentata da un rettangolo con base pari al suo tempo di completamento. I diagrammi di Gantt vengono utilizzati per rappresentare la soluzione ad un problema di scheduling.

Con l'aumento della complessità dei processi produttivi e della varietà di prodotto e grazie alla diffusione dell'information technology, le decisioni di scheduling sono passate da essere affrontate manualmente ad essere affidate ai calcolatori, fino ad arrivare ai moderni sistemi di schedulazione che permettono di creare, valutare e modificare i piani di produzione (Jeffrey 2006).
Utilizzando tali sistemi è possibile ottenere attraverso più o meno sofisticate tecniche algoritmiche delle soluzioni di scheduling ottimali o subottimali in modo da massimizzare le performance del processo. Si definisce:
m: numero di macchinari
i: macchina i-esima
j: job o operazione j-esima
p_{ij}: tempo di processamento del job *j* sulla macchina *i*
C_j: tempo di completamento di *j*
w_j: peso attribuito al job j ossia il livello di priorità
d_j: due date o data di consegna del job *j*
s_j: tempo di inizio del job
L_j: $L_j = C_j - d_j$ ritardo del job *j* (Lateness); positiva se in ritardo, negativa se in anticipo
T_j: $T_j = \max(C_j - d_j;0)$ ritardo del job *j* (Tardiness); a differenza di L_j che può essere negativa, T_j è sempre positiva, poiché tiene conto dei ritardi rispetto ai tempi di completamento.
U_j: unità di penalità, 1 se $C_j > d_j$ altrimenti 0.
La teoria della schedulazione definisce un problema di scheduling tramite una notazione a tre campi $\alpha \mid \beta \mid \gamma$ (Bruno 2012) (Pinedo 2008).
α è un campo che contiene una sola sigla che descrive il tipo di sistema produttivo, che può essere:

- Macchina singola (1)
- Macchine identiche in parallelo (P_m)
- Macchine in parallelo con diverse velocità ma costante per ogni tipo di operazione (Q_m)

- Macchine in parallelo non correlate, ossia con velocità dipendente dall'operazione da realizzare (R_m)
- Flow shop o macchine in serie (F_m)
- Job Shop (J_m), dove le operazioni hanno un percorso predeterminato da seguire
- Open Shop (O_m) dove non ci sono restrizioni sul percorso che devono seguire le operazioni e ognuna può avere percorsi differenti.

Sono possibili anche configurazione miste come il Flexible Flow Shop (FF_c), ossia una generalizzazione di flow shop e macchine in parallelo, e Flexible Job Shop (FJ_c), ossia una generalizzazione di job shop e macchine in parallelo.

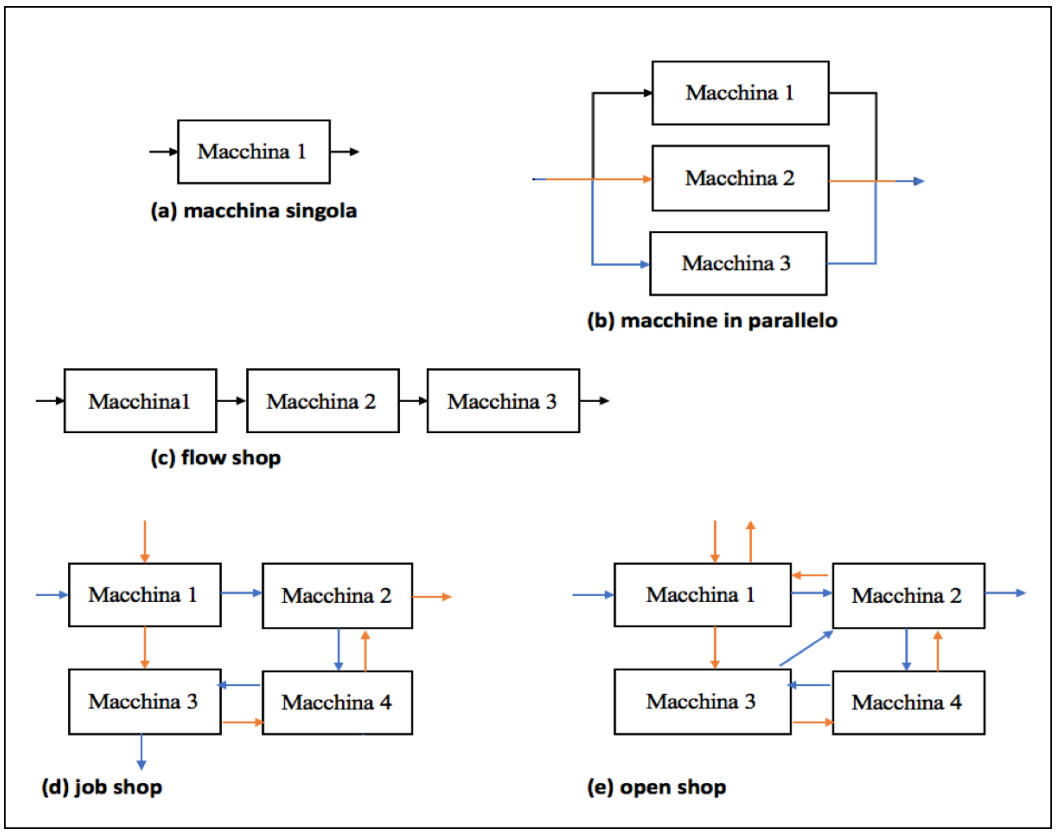


Fig. 1.3.1 Tipi di sistemi produttivi (elaborazione personale)

Il campo β identifica invece caratteristiche delle operazioni, restrizioni e vincoli del processo e può contenere più sigle. Alcuni esempi sono:

- Presenza di date di rilascio ossia è presente una data entro la quale l'operazione non può essere avviata (r_j)
- Possibilità di preemption ossia di interrompere arbitrariamente un'operazione e riprenderla in tempi successivi (prmp)
- Presenza di vincoli di precedenza tra operazioni (prec)
- Presenza di set up time dipendenti dalla sequenza delle operazioni (s_{jk})
- Processo di tipo batch, ossia ci sono un numero *b* di jobs processati simultaneamente (batch(*b*))
- Presenza di breakdowns della macchina (brkdwn)
- Restrizioni sull'eleggibilità di una macchina (M_j) può apparire in sistemi produttivi a macchine in parallelo per definire che non tutte le macchine possono eseguire l'operazione *j*
- Bloccaggio (block) può essere presente nei flow shop e indica che quando un buffer arriva al limite la macchina a monte non può più eseguire operazioni
- Nessun tempo di attesa fra le stazioni (nwt) che implica la presenza o meno di buffer fra le stazioni
- Ricircolo (rcrc) ossia un job può essere processato più volte sulla stessa macchina.

Infine, γ definisce l'obiettivo da minimizzare, ad esempio:

- Makespan, ossia il tempo di completamento dell'ultimo lavoro che lascia il sistema (C_{max} o $\max(C_j)$)
- Massimo ritardo di consegna (L_{max})
- Massima tardiness (T_{max})
- Somma pesata dei tempi di completamento ($\sum w_j C_j$)
- Somma pesata dei ritardi ($\sum w_j L_j$)
- Ritardo di consegna totale pesato o somma pesata delle tardiness ($\sum w_j T_j$)
- Numero totale di lavori in ritardo pesato ($\sum w_j U_j$)

Tali criteri di ottimizzazione possono essere ricondotti in obiettivi orientati all' efficienza, come il makespan, in obiettivi orientati ai tempi di risposta, come il

tempo di completamento totale pesato e in obiettivi orientati al rispetto delle scadenze, come il ritardo di consegna totale pesato ed il numero totale di lavori in ritardo pesato.

Spesso accade che tali obiettivi siano in conflitto tra di loro.

Un esempio rappresentativo può essere $1 \mid r_j, prmp \mid \sum w_j C_j$ che rappresenta una schedulazione di un sistema a singola macchina, con lavorazioni che entrano nel sistema ad una certa data, in cui la preemption è permessa e si deve minimizzare come funzione obiettivo il tempo totale pesato di completamento. Gli obiettivi sopraelencati sono dette funzioni decrescenti dei tempi di completamento e sono dette regolari. Ciò significa che al crescere dei tempi di completamento il valore della funzione obiettivo cresce.

Una schedulazione è detta attiva se, per anticipare il completamento di un'operazione qualsiasi, è necessario ritardare il completamento di almeno un'altra operazione. Una schedulazione è detta non ritardata se non si verifica mai che una macchina, pur potendo effettuare una operazione, resti inattiva.

A titolo di esempio si considerino due operazioni A e B con tempi di rilascio e di processamento $r_A=0$, $p_A=3$, $r_B=2$ e $p_B=2$ (Fig.1.3.2). Schedulando prima A e poi B ($s_A=0$, $C_A=3$, $s_B=2$ e $C_B=5$) si ottiene una soluzione attiva poiché anticipando B si ottiene il ritardo del completamento di A; tale soluzione è anche senza ritardo poiché la macchina non è mai inattiva. Schedulando prima B e poi A ($s_A=4$, $C_A=7$, $s_B=2$ e $C_B=4$) la soluzione rimane attiva poiché l'anticipazione di A provocherebbe il ritardo di B, come nell'esempio precedente $C_B=5$, ma è con ritardo poiché la macchina resta in attiva fino all'inizio dell'operazione B a $s_B=2$. Si può dimostrare che (Bruno 2012).:

- Una soluzione senza ritardo è certamente attiva e una soluzione non attiva è certamente con ritardo
- Una soluzione ottima di un problema con obiettivi regolari è attiva
- Una soluzione ottima con obiettivi regolari non necessariamente è senza ritardo.

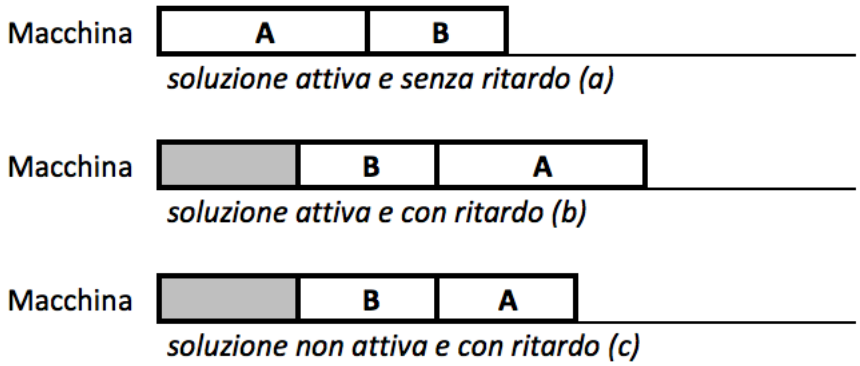


Fig. 1.3.2 Esempi di schedulazione (Bruno 2012)

Occorre inoltre identificare due casi: il caso statico, in cui i jobs presenti in un certo istante vengono processati senza che ne arrivino altri durante l'esecuzione, e il caso dinamico in cui i jobs arrivano continuamente e possono essere previsti solo su base statistica. A sua volta i due casi si possono dividere entrambi in due casi; il primo è il caso deterministico in cui i tempi di procesamento sono definiti e non variano mentre il secondo è il caso stocastico in cui i tempi di processamento sono soggetti a variazioni casuali.
Risolvere un problema di scheduling significa determinare la tempificazione delle operazioni su ogni macchina, rispettando i vincoli del sistema.
I problemi di schedulazione possono essere descritti attraverso modelli; i principali modelli verranno analizzati nel capitolo 2. Nel seguito si fornisce invece una classificazione delle principali tecniche risolutive (Bruno 2012).

1.4 Tecniche risolutive per i problemi di scheduling

Un problema di scheduling può essere ricondotto ad un problema di ottimizzazione che può essere formulato in maniera seguente:
$max(z = f(x_1, \dots, x_n))$ o $min(z = f(x_1, \dots, x_n))$
sottoposto a:
$h_i(x_1, \dots, x_n) \leq a_i$ i=1,...,s
$g_h(x_1, \dots, x_n) \leq b_h$ h=1,...,t

$(x1, \dots, xn) \in R^n$

Una descrizione formale di questo tipo prende il nome di modello.

Le relazioni h_i e g_h sono i vincoli fra le variabili, a_i e b_h sono i termini noti mentre z è la funzione obiettivo, ossia il criterio secondo il quale viene preferita una soluzione rispetto ad un'altra (Bruno 2012).

I metodi di risoluzione dei problemi di scheduling sono numerosi e si differenziano a seconda della complessità dei problemi che possono trattare, della tipologia del problema, come descritto nel paragrafo 1.3, della qualità della soluzione che essi forniscono e della difficoltà computazionale. A seconda di queste caratteristiche un problema di ottimizzazione può essere affrontato con tecniche euristiche, che forniscono una soluzione buona, approssimata, ma non ottima, piuttosto che tecniche esatte. Spesso per utilizzare tecniche esatte, specie per problemi complessi, è necessario un tempo computazionale troppo elevato rispetto al contesto operativo dove si vuole avere una soluzione in tempi più brevi. Ciò rende spesso necessario preferire tecniche euristiche che forniscono soluzioni buone, non ottime, ma in tempi in linea con il contesto produttivo. I metodi quindi possono essere raggruppati in due macro classi principali:

- i metodi euristici
- i metodi esatti.

Le tecniche euristiche, a loro volta, si dividono in tecniche costruttive e tecniche migliorative. Un algoritmo costruttivo è caratterizzato da 3 passi:

1- inizializzazione: si sceglie un elemento di partenza per la costruzione della soluzione parziale
2- selezione di un nuovo elemento da aggiungere alla soluzione parziale: si individua il criterio di selezione
3- criterio di arresto: criterio per cui l'algoritmo si ferma individuando la soluzione o si riparte dal passo 2.

Tali algoritmi sono detti greedy, o avidi, poiché ad ogni iterazione scelgono l'elemento che a quel passo risulta più conveniente, non effettuando considerazioni sulla composizione complessiva della soluzione. Una tecnica

costruttiva per i problemi di scheduling consiste nell'utilizzo di regole di priorità per ordinare le operazioni da eseguire. Le operazioni sono assegnate nell'ordine indicato man mano che si rendono disponibili le macchine. Le regole più utilizzate sono:

- Assegnazione casuale
- FCFS o first come first served ossia ordinamento secondo l'ordine di arrivo delle operazioni da eseguire
- WSPT (weighted shortest processing time) o WLPT (weighted longest processing time) ossia si ordinano le operazioni secondo il loro tempo di processamento pesato
- EDD o earliest due date ossia si ordinano le operazioni per tempo di scadenza crescente
- MST o minimum slack time ossia si ordinano le operazioni secondo il tempo di slack crescente ($sl_j=d_j-r_j-p_j$ con d_j tempo di consegna, r_j tempo di rilascio e p_j tempo di processamento
- CR o critical ratio che identifica un indice della criticità della lavorazione ($CRj=(d_j-r_j) / p_j$) e le operazioni andranno schedulate in ordine crescente di CR.

Alcune regole possono essere applicate anche dinamicamente, come la MST o la CR, sostituendo a r_j il valore dell'istante in cui l'operazione j si rende disponibile al più presto. La scelta della regola dipende dal tipo di problema e di funzione obiettivo (capitolo 2).

Le tecniche migliorative, permettono a partire da soluzioni ammissibili fornite dai metodi costruttivi, di migliorare la soluzione corrente. Essi prendono anche il nome di algoritmi di ricerca locale poiché ricercano il miglioramento analizzando l'intorno del punto della soluzione corrente. Sia Ω l'insieme delle soluzioni ammissibili, con $S\epsilon\Omega$ una di queste soluzioni e con $N(S) \subseteq \Omega$ un intorno di S. Con mossa si intende un'operazione che a partire da S consente di generare N(S) modificando una o più caratteristiche della soluzione corrente. All'interno di N(S) si sceglie una soluzione $S'\epsilon N(S)$ candidata a sostituire la soluzione corrente. S'

sarà accettata se in accordo con un criterio di accettazione e sostituirà S. I passi di un algoritmo migliorativo sono:

1- Inizializzazione: si individua una soluzione iniziale S e si valuta il valore di funzione obiettivo z(S)
2- individuazione dell'intorno: si definisce un intorno N(S) della soluzione S
3- scelta di una nuova soluzione e eventuale accettazione come nuova soluzione: di individua $S' \epsilon N(S)$ cui corrisponde il minimo della funzione obiettivo.
4- criterio di arresto: se Z(S') è migliore di Z(S) si sostituisce S con S' e si torna al passo 2. In caso contrario si arresta l'algoritmo.

Vi sono poi anche tecniche metaeuristiche come la simulated annealing, tabu search e gli algoritmi genetici che si sono dimostrate efficaci nella risoluzione di problemi di scheduling (Bruno 2012).

Per quanto riguarda i metodi esatti due approcci molto utilizzati sono la programmazione matematica (mathematical programming) e la programmazione a vincoli (constraint programming). Tali approcci sono tecniche simili di ottimizzazione che consistono nel trovare i valori delle variabili che minimizzano una funzione obiettivo soggetta a dei vincoli utilizzando la prima metodi matematici, la seconda metodi informatici derivanti dalla programmazione logica, teoria dei grafi e dall'intelligenza artificiale.

1.5 Programmazione matematica

Dato un problema di ottimizzazione, la programmazione matematica (MP) ha come obiettivo quello di minimizzare o massimizzare una funzione reale in cui le variabili sono soggette a dei vincoli. I metodi matematici utilizzati dipendono dalle caratteristiche del problema affrontato; in particolare:

- se *f*, *g* e *h* sono lineari il problema si dice di programmazione lineare
- se f è convessa e h e g sono lineari si parla di programmazione convessa

- se f o g o h sono non lineari il problema si dice di programmazione non lineare.

Questi tipi di modelli sono di programmazione continua ma in molti problemi reali, tra cui quelli di scheduling, le variabili sono vincolate ad essere numeri interi. Un problema di programmazione lineare intera è detto di ottimizzazione combinatoria. Vi sono inoltre anche problemi di programmazione lineare mista intera quando solo un sottoinsieme delle variabili è intero.
Si consideri un problema di ottimizzazione monodimensionale, ossia ad una variabile; formalmente:
$min(z=f(x))$
soggetto a:
$c \leq f(x) \leq d$
se $f(x)$ è lineare la soluzione si troverà su uno dei due vertici $f(a)$ o $f(b)$ come in figura 1.5.1, dove il minimo è $f(b)$.

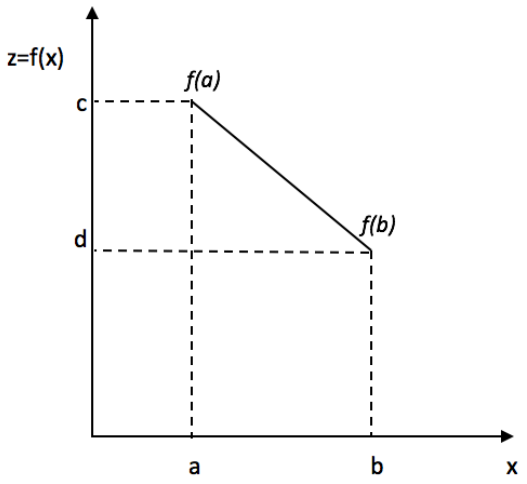


Fig. 1.5.1 Esempio ottimizzazione lineare monodimensionale (Bruno 2012)

Estendendo il caso a una funzione a n dimensioni, la soluzione ottima si troverà su un vertice appartenente all'insieme delle soluzioni ammissibili. Confrontando i valori di z nei vertici è possibile trovare la soluzione che ottimizza z. È necessario però esplorare i vertici in maniera intelligente altrimenti il confronto multiplo risulta inefficiente.

La programmazione lineare utilizza generalmente l'algoritmo del simplesso per esplorare i vertici del dominio di ammissibilità, il quale si è dimostrato mediamente molto efficiente per ottenere la soluzione ottima.
Molti problemi di scheduling possono essere ricondotti a problemi di programmazione lineare intera. Nel caso della programmazione lineare intera non valgono le considerazioni fatte per la programmazione lineare continua poiché i vincoli di interezza modificano la geometria del problema. Infatti essendo le variabili intere, il dominio di ammissibilità diventa discreto e il numero delle soluzioni ammissibili diventa finito. Si consideri il caso a due variabili in fig. 1.5.2. Le soluzioni ammissibili sono i punti della griglia interni alla regione di ammissibilità. A queste condizioni non valgono più le considerazioni sui vertici fatte per la programmazione lineare.

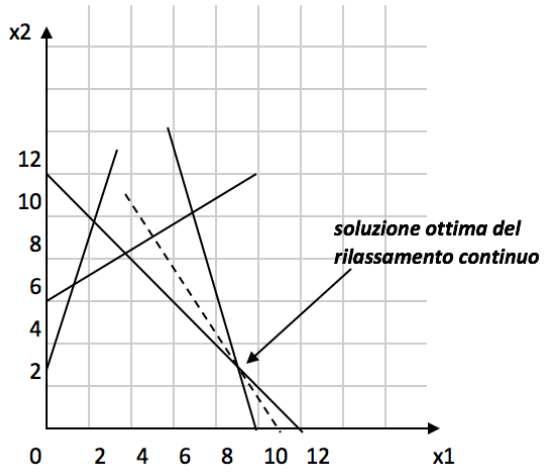


Fig. 1.5.2 Esempio (Bruno 2012)

Si definisce il rilassamento continuo del problema di programmazione lineare intera come il modello di programmazione lineare definito sostituendo i vincoli di interezza del problema di programmazione lineare intera con il vincolo di positività. Ottenuta la soluzione del problema rilassato, se essa è intera è anche la soluzione del problema non rilassato. Spesso però ciò non accade e l'arrotondamento non fornisce la soluzione esatta (Bruno 2012). In particolare, in un problema di minimizzazione la soluzione del problema rilassato fornisce un

limite inferiore dell'ottimo del problema originario, mentre per un problema di massimizzazione fornisce un limite superiore. L'arrotondamento permette solamente di avere una soluzione approssimata e quindi euristica del problema, non ottima. Per i problemi di programmazione lineare intera si utilizza spesso una combinazione di rilassamento e branch and bound.

1.6 Programmazione a vincoli

La programmazione a vincoli, invece, è una tecnica di ottimizzazione che deriva dall'informatica, dalla programmazione logica, dalla teoria dei grafi e dall'intelligenza artificiale. La programmazione a vincoli (CP) si è dimostrata molto efficiente nel risolvere problemi di schedulazione (IBM s.d.). La programmazione a vincoli è usata per risolvere problemi di soddisfacimento dei vincoli (constraint satisfaction problem o CSP). Un problema di soddisfacimento dei vincoli è costituito da un insieme di variabili $X=\{X_1,\ldots,X_n\}$, da un insieme di rispettivi domini delle variabili $D=\{D_1,\ldots,D_n\}$ e da un insieme di vincoli $C=\{C_1,\ldots,C_m\}$. Spesso i CSP richiedono inoltre una funzione obiettivo da minimizzare/massimizzare che non è però strettamente necessaria. Una soluzione di tale problema consiste nell'assegnare i valori alle variabili in modo che tutti i vincoli siano soddisfatti.

MP e CP utilizzano gli stessi concetti, ossia variabili decisionali, funzioni obiettivo e vincoli che costituiscono il modello matematico del problema da affrontare. Vi sono però delle differenze che rendono più efficace l'applicazione di una rispetto l'altra.

I modelli di CP utilizzano solo variabili decisionali discrete, intere o booleane, mentre i modelli di MP supportano sia variabili discrete che continue.

MP supporta solamente vincoli lineari, vincoli logici linearizzati e vincoli quadratici convessi (funzioni del tipo $ax^2+by^2+cz^2+dxy+exz+fyz$ convesse) mentre CP ha un range più ampio di espressioni come modulo (che fornisce il resto della

divisione intera), divisione intera, minimo o massimo. Qualsiasi tipo di vincolo può essere rappresentato nella programmazione a vincoli.
Sia CP, sia MP forniscono una prova di ottimalità della soluzione, CP esplorando tutto lo spazio di ricerca (in modo intelligente tramite tecniche che saranno esposte nel capitolo 3) e dimostrando che non ci sono soluzioni migliori di quella trovata, mentre MP tramite dimostrazione algebrica che la soluzione trovata rappresenta il limite inferiore, in caso di problema di minimizzazione o viceversa.
CP non necessita di assunzioni sulle proprietà matematiche dello spazio delle soluzioni (linearità, convessità etc.) mentre MP necessita di un modello che ricade in una ben definita categoria matematica, ad esempio mixed integer (IBM s.d.).
Le differenze possono essere riassunte nella seguente figura.

Constraint programming vs. mathematical programming

Feature	MP	CP
Relaxation	Yes	No
GAP measure	Yes	No
Optimality proof	Yes	Yes
Modeling limitations	Quadratic problems are limited to PSD (Positive Semi Definite) problems and Second Order Cone Programming (SOCP) problems	Discrete problems
Specialized constraints	No	Yes
Logical constraints	Yes	Yes
Theoretical grounds	Algebra	Graph theory and algorithmic
Modeler support	Yes	Yes
Model and run	Yes	Yes

Fig. 1.5.1 Mathematical vs Constraint programming. Tabella comparativa (IBM s.d.)

1.7 Utilizzo della programmazione a vincoli

La programmazione a vincoli è caratterizzata da flessibilità di modellazione e da una potente strategia di ricerca e per questo viene utilizzata spesso per risolvere problemi di ottimizzazione combinatoria come i problemi di scheduling, timetabling, sequencing e allocation. Questi problemi possono risultare difficili da

trattare con la programmazione matematica. CP permette un linguaggio di programmazione più flessibile, vicino al linguaggio naturale, che contiene all'interno funzioni primitive che facilitano la modellizzazione del problema, il quale non necessità che venga fatto rientrare in una categoria specifica. Come specificato nel paragrafo precedente, CP permette inoltre l'utilizzo di qualsiasi tipo di vincolo e espressione, non solamente lineare, con la possibilità o meno di inserire una funzione obiettivo.

La strategia di ricerca delle soluzioni della programmazione a vincoli, che verrà esposta nei capitoli successivi, permette di non dover esplorare l'intero spazio di ricerca che contiene tutte le combinazioni dei valori nei domini di ogni variabile decisionale, che altrimenti sarebbe troppo vasto, ma permette di rimuovere valori inconsistenti, accelerando il processo di ricerca (Feng 2015).

Capitolo 2 – Modelli per i problemi di schedulazione

Nel seguente capitolo si descrivono i modelli fondamentali di schedulazione e alcune possibili tecniche risolutive.

2.1 Parametri e variabili decisionali

Per modellizzare un problema di scheduling è necessario innanzitutto definire quali possono essere i parametri e le variabili decisionali.
I parametri più comuni dei job j, o ordini di produzione, utilizzati nella modellizzazione dei problemi di scheduling sono:

- p_j: tempo di esecuzione o processamento
- r_j: tempo di rilascio
- d_j: data di consegna

I parametri per le machine i possono essere:

- o_j: tempo di inizio disponibilità macchina i
- c_j: tempo di termine disponibilità macchina j
- v_{ij}: velocità di esecuzione del job j sulla macchina i

Il tempo effettivo di esecuzione è dato da p_j/v_{ij}; inoltre è possibile considerare all'interno del tempo effettivo anche l'efficienza μ della macchina che spesso non è 100%. Il tempo effettivo di esecuzione può dunque essere definito da $p_j/(\mu v_{ij})$.
Le variabili decisionali che compaiono nei modelli possono essere:

- x_{ij}: assegnamento job j alla macchina i
- C_j: tempo di completamento job j
- $s_j = C_i - p_i$: tempo di inizio esecuzione job j
- $W_j = s_j - r_j$: tempo di attesa
- $L_j = C_i - d_i$: tempo di ritardo
- $T_j = \max(0;C_j - d_j)$: ritardo
- U_i: numero di volte in cui si va in ritardo

2.2 Schedulazione a macchina singola

I modelli di schedulazione a macchina singola sono i più semplici ma permettono di porre le basi per problemi più complessi. In questo caso, una serie di job devono essere processati da una singola macchina. A seconda delle funzioni obiettivo da minimizzare si possono utilizzare diversi metodi. Per questo tipo di modello spesso l'utilizzo di regole di priorità fornisce la soluzione ottimale. Per il problema $1||C_{max}$, ossia macchina singola con ottimizzazione del makespan, la soluzione ottima si ottiene schedulando in ordine qualsiasi ottenendo $C_{max} = \sum p_j$.
Nel caso $1||\sum w_j C_j$, ossia dell'ottimizzazione del tempo totale pesato di completamento, è possibile dimostrare che la regola Weighted Shortest Processing Time first, o WSPT first la quale assegna prima i job con tempo di processamento pesato minore, fornisce la soluzione ottimale.
Dimostrazione: WSPT first è ottimale per $1||\sum w_j C_j$.
Si supponga per assurdo che esista una sequenza ottima S che non sia WSPT e sia la funzione obiettivo:

$$z(S) = \sum_j w_j C_j(S)$$

Per questo in S ci devono essere almeno due jobs *i* e *j* adiacenti per i quali:

$$\frac{p_i}{w_i} > \frac{p_j}{w_j}$$

Si assuma che *i* inizi al tempo *t*.
Si consideri la sequenza S' ottenuta da S scambiando *i* e *j*.

$$z(S') = z(S) - w_i(t + p_i) - w_j(t + p_i + p_j) + w_j(t + p_j) + w_i(t + p_j + p_i)$$

$$z(S') = z(S) + w_i(p_j) - w_j(p_i)$$

ed essendo $\frac{p_i}{w_i} > \frac{p_j}{w_j}$ $implica$ $z(S') < z(S)$ e quindi ciò contraddice l'ottimalità di z(S) e quindi la regola WSPT è ottimale per $1||\sum w_j C_j$.
Analogamente, tale regola di priorità è ottimale anche per $1||\sum w_j L_j$.
In caso di problemi di tipo $1||L_{max}$ o T_{max}, essi possono essere risolti in maniera ottimale con la regola Earliest Due Date, o EDD.

Il problema $1||\sum U_j$ può essere risolto tramite il seguente algoritmo:
Si definisce J insieme di tutte le operazioni, J_k la sequenza di operazioni provvisoriamente individuata fino all'iterazione k, J_k' l'insieme delle operazioni che fino all'iterazione k si è già stabilito che saranno in ritardo e J_k''=J-J_k-J_k' insieme delle restanti operazioni non ancora esaminate.
L'algoritmo procede con n iterazioni pari al numero dei jobs e a ogni iterazione prende in considerazioni i primi k jobs.
Step 1: Inizializzazione
k=0; J_0=Ø, J_0'=Ø, J_0''=J e t=0.
Step 2: Analisi di un nuovo elemento
Si individua l'operazione j* di J_k'' con il minimo tempo di scadenza (regola EDD) e la si inserisce in $J_{k+1} = J_k \cup \{j^*\}$.
Se $C_{j^*} = t + p_{j^*} > d_{j^*}$ si individua l'operazione $j^{**} \epsilon J'_{k+1}$ con tempo di processamento massimo; j** viene schedulata definitivamente in ritardo e passa da J_{k+1} a J'$_{k+1}$. Si pone quindi $J_{k+1} = J_{k+1} - \{j^{**}\}$ e $J'_{k+1} = J'_k \cup \{j^{**}\}$ e infine $J''_{k+1} = J''_k - \{j^*\}$, $t = \sum_{j \epsilon J_k} p_j$ e k=k+1.
Step 3: Criterio di arresto
Se J_k''=Ø l'algoritmo si arresta; si schedulano le operazioni di J_k secondo l'ordine individuato e le operazioni di J_k' secondo un ordine qualsiasi. Altrimenti si torna al passo 2. Il numero di operazioni in ritardo è dato dal numero di elementi in J'$_k$.
L'algoritmo aggiunge i jobs all'insieme dei jobs non in ritardo in ordine crescente di data di consegna. Se inserire il job k implica che k non sarà completato in tempo, allora il job con tempo più lungo di processamento viene scartato e inserito in J'$_k$.
È possibile dimostrare che tale algoritmo è ottimale per la schedulazione di problemi di tipo $1||\sum U_j$ (Bruno 2012).
Riassumendo quindi:

- $1||L_{max}$ o T_{max} si risolve con la regola EDD in maniera ottimale.
- $1||\sum w_j C_j$ e $1||\sum w_j L_j$ si risolvono con la regola WSPT first in maniera ottimale.

- $1||\sum w_j U_j$ si risolve con l'algoritmo descritto in maniera ottimale.

Semplificando, i migliori risultati su singola macchina si ottengono utilizzando la regola WSPT per minimizzare obiettivi di efficienza e tempi di risposta, mentre e utilizzando la regola EDD per minimizzare obiettivi orientati al rispetto delle scadenze.

Esempio Scheduling a singola macchina:

Elenco jobs				
Job_j	p_j	w_j	p_j/w_j	d_j
1	2	6	0,333	10
2	6	3	2	11
3	4	14	0,286	12
4	5	8	0,625	8

$Min(\sum w_j C_j)$		regola WSPT							
Job_j	p_j	w_j	p_j/w_j	d_j	s_j	C_j	L_j	T_j	$w_j C_j$
3	4	14	0,286	12	0	4	-8	0	56
1	2	6	0,333	10	4	6	-4	0	36
4	5	8	0,625	8	6	11	3	3	88
2	6	3	2	11	11	17	6	6	51
					Lmax	Tmax	Ltot	Ttot	$\sum w_j C_j$
					6	6	-3	9	231

Min(Tmax)		regola EDD							
Job_j	p_j	w_j	p_j/w_j	d_j	s_j	C_j	L_j	T_j	$w_j C_j$
4	5	8	0,625	8	0	5	-3	0	40
1	2	6	0,333	10	5	7	-3	0	42
2	6	3	2	11	7	13	2	2	39
3	4	14	0,286	12	13	17	5	5	238
					Lmax	Tmax	Ltot	Ttot	$\sum w_j C_j$
					5	5	1	7	359

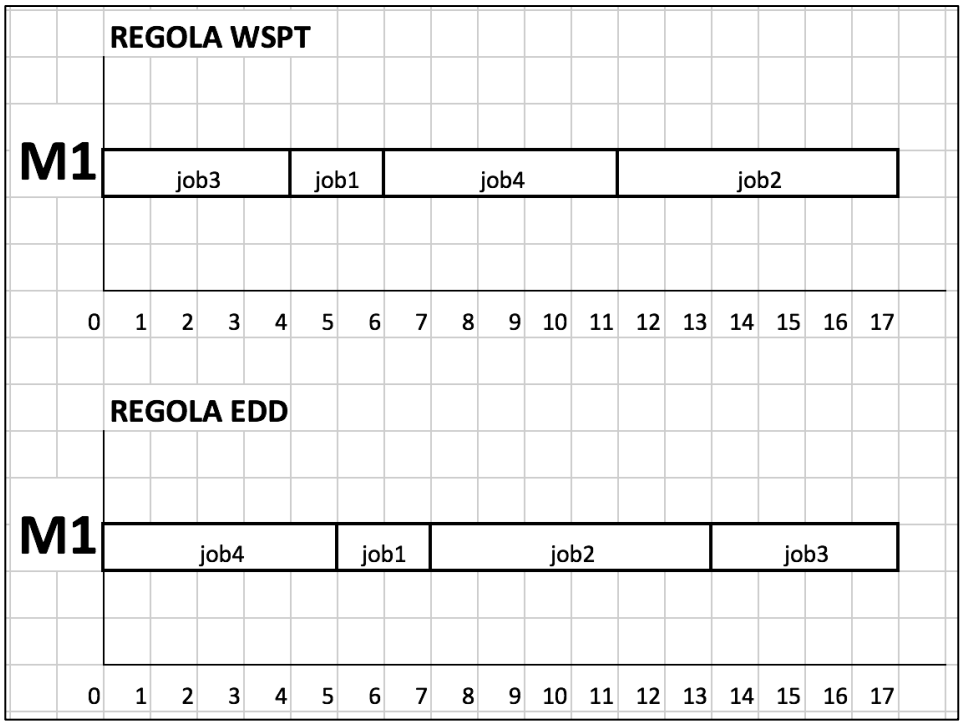


Fig. 2.1.1 Gantt esempio schedulazione a macchina singola (elaborazione personale)

Si considera ora la presenza di legami di precedenza fra i jobs, in particolare il problema $1|prec|\sum w_j C_j$. Per problemi con relazioni di precedenza formate da semplici catene, la soluzione può essere trovata con il metodo della catena (Chain method) che consiste in:

1- Per ogni insieme di jobs nel diagramma delle precedenze, formare per ogni catena l'insieme dei job non ancora schedulati.

2- Trovare, per ogni catena, il valore minimo di $\rho = \frac{\sum p_j}{\sum w_j}$, detto $\rho - factor$.

3- Selezionare i jobs dalla catena che presenta complessivamente valore minimo di $\rho - factor$

4- Inserire questi jobs nello scheduling parziale e cancellarli dal diagramma delle precedenze

5- Ripetere fino a che non sono stati schedulati tutti i jobs.

Esempio:

1 →2→3→4

5→6→7

Jobs	1	2	3	4	5	6	7
w_j	6	18	12	8	8	17	18
P_j	3	6	6	5	4	8	10

Per la prima catena:

Jobs	1	1→2	1→2→3	1→2→3→4
$\rho - factor$	3/6=0,5	(3+6)/(6+18)=0,375	0,416	0,455

Minimo per 1→2

Per la seconda catena:

Jobs	5	5→6	5→6→7
$\rho - factor$	0,51	0,480	0,512

Minimo per 5→6

Complessivamente il valore minimo si ha per 1→2 e quindi si andrà a schedulare prima 1→2 per poi ripetere l'algoritmo per le catene 3→4 e 5→6→7 (Pinedo 2008).

Per il problema 1|prec|L_{max} in cui si va a minimizzare L_{max} si utilizza la regola LRS (last remaining slack). RS o remaining slack è calcolabile da $RS_j = d_j - p_j - t$ con t tempo dello scheduling.

La regola prevede:

1- A tempo zero, t=0.

2- Dal diagramma delle precedenze, formare l'insieme di jobs schedulabili

3- Calcolare RS per ognuno di questi jobs
4- Schedulare il job j con minore RS
5- Rimuovere il job schedulato dall'insieme dei job schedulabili
6- Se tutti i job sono schedulati STOP. Altrimenti aggiornare l'insieme dei job schedulabili dal diagramma delle precedenze e il valore di t. Andare allo step 3 (Pinedo 2008).

2.3 Schedulazione a macchine parallele

Lo scheduling a macchine parallele prevede che una serie di jobs o operazioni debbano essere processati da una macchina delle m macchine in parallelo, con tempi di processamento che possono essere identici su ogni macchina, maggiori o minori su una macchina rispetto alle altre, oppure non correlati.
Nel caso di macchine parallele con minimizzazione del makespan, il problema $P_m||C_{max}$ può essere modellizzato nel seguente modo:

$\min (z = Cmax)$ (1)

sottoposto a:

$\sum_{j=1}^{n} p_j x_{ij} \leq Cmax \qquad i = 1, \dots, m$ (2)

$\sum_{i=1}^{m} p_j x_{ij} \leq Cmax \qquad j = 1, \dots, n$ (3)

$\sum_{i=1}^{m} x_{ij} = 1 \quad i = 1, \dots, m$ (4)

$x_{ij} \geq 0$ *se preemption* è *ammessa* (5)

$x_{ij} = [1; 0]$ *se preemption non* è *ammessa.* (6)

(1) consiste nella funzione obiettivo da minimizzare, in questo caso il makespan.
(2) impone che il tempo totale di processamento su ogni macchina i sia minore del makespan
(3) il tempo totale di processamento di ogni operazione j su tutte le macchine non superi il makespan
(4) ogni operazione deve essere assegnata completamente.

(5) se vi è la possibilità di preemption, indica la porzione di job j completato sulla macchina m

(6) se non vi è la possibilità di preemption, indica se il job j è o non è assegnato alla macchina m.

Questa formulazione fornisce una soluzione ma non da una tempificazione delle attività, ossia una vera programmazione, ma solamente fornisce quanto tempo il job j spende sulla macchina i. Tuttavia con queste informazioni una schedulazione può essere costruita facilmente. Un euristico che è in grado di trovare una buona soluzione per il problema $Pm||C_{max}$ (di minimizzazione del makespan) è la regola LPT, ossia longest processing time. La regola LPT assegna a t=0 il job più lungo alle macchine m. Dopodiché, ogni volta che una macchina è liberata, il job più lungo tra quelli non ancora processati viene assegnato alla macchina. In questo modo vengono lasciati per ultimi i job più brevi; in tal modo possono essere usati per bilanciare i carichi sulle macchine. In caso di preemption ammessa, la regola equivalente alla LPT è la regola LRPT, ossia longest remaining processing time first, il quale però risulta difficilmente applicabile in pratica poiché il numero di preemption può risultare infinito.

In problemi con minimizzazione di L_{max} o T_{max} la regola EDD può fornire soluzioni subottimali.

In problemi $Pm||\sum C_j$ è possibile dimostrare che la regola SPT, shortes processing time, fornisce una soluzione ottima. Tuttavia, la regola più generale, ossia WSPT (weighted shortest processing time), utilizzata nel caso a singola macchina per ottenere una soluzione ottima per il problema $1||\sum w_j C_j$, non può essere generalizzata alle macchine parallele, ma rimane tuttavia un buon euristico (Pinedo 2008) (Bruno 2012).

Esempio di schedulazione $P_m||\sum C_j$: utilizzo della regola SPT (Bruno 2012).

3 macchine M1, M2, M3;

jobs	A	B	C	D	E	F	G
p_j	6	8	5	7	10	7	4

Regola SPT

ORDINE DI SCHEDULAZIONE:

G	C	A	D	F	B	E

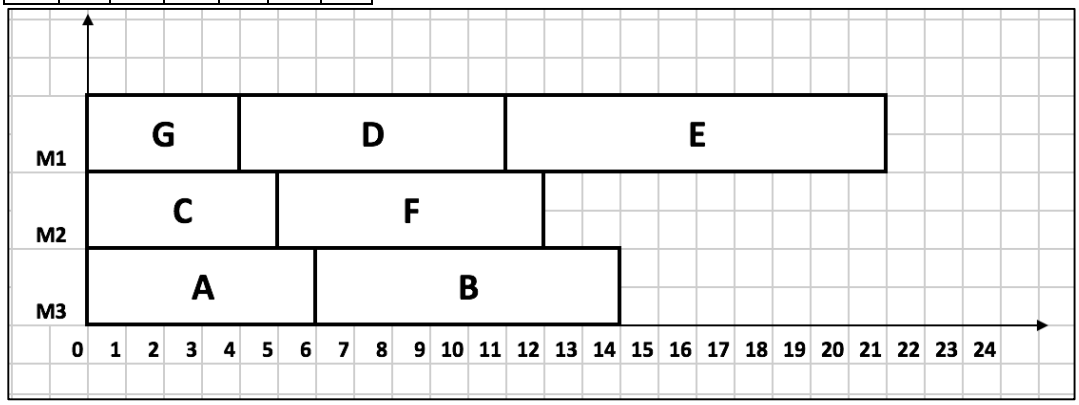


Fig. 2.3.1 Esempio schedulazione con regola SPT (Bruno 2012)

2.4 Schedulazione di un Flow shop

La schedulazione di problemi di flow shop è un problema multifase che è caratterizzato da elevata complessità computazionale. In un problema multifase ogni job è individuato da una sequenza di operazioni, tali che due operazioni consecutive sono eseguite su macchine diverse. Si utilizza la terna (i, j, k) per indicare l'operazione j relativa al job i da eseguirsi sulla macchina k.

In un flow shop ogni job è costituito dalla medesima successioni di operazioni tipica di impianti a produzione continua con elevata standardizzazione. In un flow shop tutti i jobs, o gli ordini di produzione, presentano la stessa successione di operazioni. Se ogni operazione è eseguite da una macchina diversa si può utilizzare solamente due indici dato che l'operazione j e la macchina k sono correlate (j=k quindi si può utilizzare solo gli indici i e j o i e k del job e della macchina/operazione).

Per i problemi F2||C_{max} di minimizzazione del makespan con due macchine in serie (flow shop a due macchine) è possibile trovare una soluzione esatta tramite la regola di Johnson. Sia J un insieme di jobs ciascuno formato da due operazioni e si

supponga che la prima macchina sia la 1 e la seconda macchina sia la 2. p_{jk} è il tempo di processamento dell'operazione j sulla macchina k. Si suddividono i jobs in due insiemi:

- $L' = \{j \in J : p_{i1} \leq p_{i2}\}$ ossia i job con tempo di processamento sulla prima macchina inferiori o uguali a quelli sulla seconda;
- $L'' = J - L'$ ossia i job con tempo di processamento sulla prima macchina maggiori a quelli sulla seconda.

Definiti i due insiemi la regola di Johnson indica di effettuare una schedulazione con permutazione, ossia una schedulazione in cui la successione delle operazioni su ogni macchina si ripete, secondo l'odine $L' \rightarrow L''$. In particolare:

- le operazioni L' seguiranno la regola SPT, shortest processing time first, sui tempi di processamento sulla prima macchina;
- le operazioni L'' seguiranno la regola LPT, longest processing time first, sui tempi di processamento sulla seconda macchina.

Esempio:

JOBS	1	2	3	4	5	6	7	8
p_{i1}	2	3	3	4	7	3	3	3
p_{i2}	3	1	5	6	5	2	5	4

Utilizzo della regola di Johnson:

L'= {1,3,4,7,8}

L''= {2,5,6}

SPT per L' quindi l'ordine sarà {1, 3, 7, 8, 4};

LPT per L'' quindi l'ordine sarà {5, 6, 2}

L'ordine complessivo sarà quindi {1, 3, 7, 8, 4, 5, 6, 2}. La soluzione è mostrata in figura 2.4.1 dove ogni coppia (i,k) rappresenta il job i sulla macchina k. Ne risulta un makespan di 32.

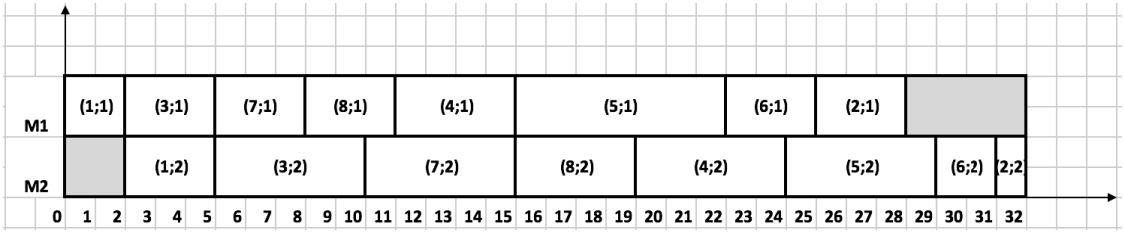


Fig. 2.4.1 Scheduling flow shop. Esempio. (elaborazione Personale)

Per m=2 la regola di Johnson permette di fornire la soluzione ottima ma non è però applicabile a problemi più generali. Perciò nei casi con m>2 si utilizzano delle regole che permettono di ottenere una soluzione approssimata. Le regole di priorità più usate sono:

- regola dei rapporti: ordinamento per p_{i2}/p_{i1} decrescente
- regola delle differenze o regola di Parker: ordinamento secondo la priorità $w_i = -(m-1)p_{i1} - (m-3)p_{i2} + \cdots + (m-3)p_{i,m-1} + (m-1)p_{im}$
- regola del tempo totale di processamento o di Gupta: si attribuisce a ogni job j la priorità:

  $w_i = \frac{e_i}{\min_{k=1,m-1}(p_{ik}+p_{i,k+1})}$ $con\ e_i = 1\ se\ p_{i1} < p_{im}\ e\ e_i = -1\ altrimenti.$

Un ulteriore approccio è l'approccio bottleneck in cui si individua una macchina "collo di bottiglia" e si sequenziano gli ordini in base a delle considerazioni su tale macchina. Sia m* la macchina collo di bottiglia, scelta ad esempio in base alla somma dei tempi di processamento maggiore, e supponiamo di conoscere un limite superiore C'_{max} del C_{max} individuato con un qualsiasi algoritmo. Fissato arbitrariamente un valore $C''_{max} < C_{max}$, l'algoritmo si articola in:

- passo 1: calcolo dei parametri

  Per ogni job j si valutano i seguenti parametri:

  h_j: somma dei tempi di processamento delle operazioni del job j che precedono m*

  t_j: somma dei tempi di processamento delle operazioni del job j che seguono m*

d_j: C''_{max}-t_j: tempo massimo di completamento che l'operazione del job j sulla macchina m* deve necessariamente presentare per evitare che sia abbia $C_{max}>C''_{max}$

- passo 2: risoluzione del problema $1|r_j|L_{max}$
  si risolve tale problema sulla macchina m* assumendo $p_j=p_{jm*}$, $r_j=h_j$ e $d_j=C''_{max}-t_j$. Se $L_{max}>0$ la permutazione individuata presenta certamente un $C_{max}>C''_{max}$. Al contrario se $C_{max}\leq C''_{max}$ si deve verificare se la sequenza replicata sulle m macchine facendo rispettare i vincoli di precedenza delle operazioni dello stesso job migliora.

C''_{max} può essere posto pari ad un valore desiderato o si può fissare $C''_{max} = C'_{max} - \Delta$ (Bruno 2012).

Esempio tratto da (Bruno 2012):

F3||C_{max}

				Regola Gupta		Regola Parker	
JOB	pi1	pi2	pi3	wi	ordine	wi	ordine
1	4	5	2	-1/7	4	-4	5
2	2	2	1	-1/3	5	-2	4
3	1	7	3	1/8	2	+4	1
4	5	4	4	-1/8	3	-2	3
5	3	4	3	1/7	1	0	2
TOT	15	22	13				

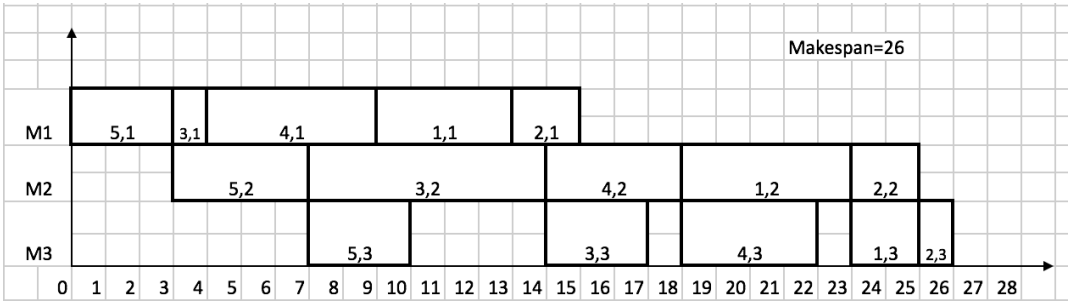


Fig. 2.4.2 Schedulazione flow shop con regola di Gupta (Bruno 2012)

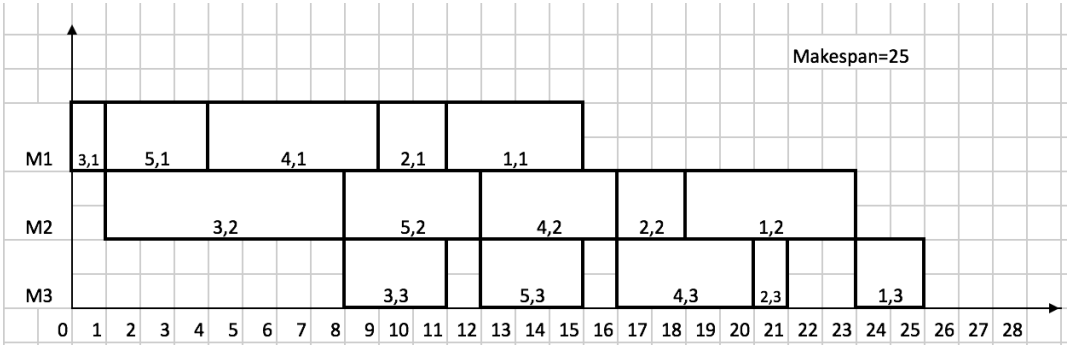


Fig. 2.4.3 Schedulazione flow shop con regola di Parker (Bruno 2012)

Nell'approccio bottleneck: m*=macchina 2 essendo $\sum p_{i2} = 22$ la massima somma dei tempi di processamento sulla macchina e C'_{max}=25 limite superiore di C_{max}. Si schedula la macchina 2 risolvendo il problema $1|r_j|L_{max.}$ Per tale problema di utilizza la regola EDD considerando però le date di rilascio r_j.

Passo 1:

JOB	pi2	hj	tj	dj=C'max-tj
1	5	4	2	25-tj=23
2	2	2	1	24
3	7	1	3	22
4	4	5	4	21
5	4	3	3	22

Passo 2:

EDD CON r_j:

	Ot	Assegno	Ot insieme job assegnabili a t
t=0	0	-	
t=1	3	3	
t=8	1,2,4,5	4	Avendo minore dj
t=12	1,2,5	5	
t=16	1,2	1	
t=21	2	2	

Makespan=24.

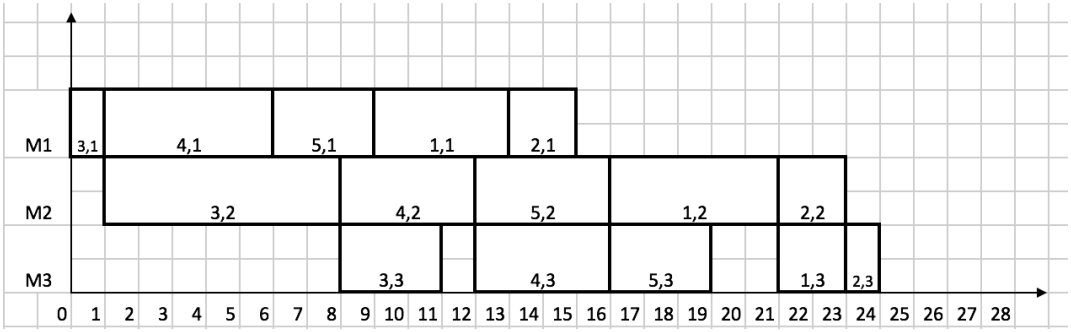


Fig. 2.3.4 Schedulazione flow shop approccio bottleneck (Bruno 2012)

Un ulteriore euristico per il problema di flow shop con m>2 con minimizzazione del makespan è quello proposto da Campbell, Dudek e Smith (Vollmann, Berry e Whybark 1997). L'algoritmo si articola nei seguenti passi:

1- risolvere il primo problema considerando solo la macchina 1 e la m, ossia definire la sequenza ignorando le altre m-2 macchine (si risolve un problema $F2||C_{max}$)
2- risolvere il secondo problema mettendo insieme le macchine 1 e 2 e m-1 e m per formare due macchine fittizie. I tempi di processamento alla macchina fittizia 1 sono uguali alla somma di quelli della macchina 1 e 2

così come quelli della macchina fittizia 2 sono uguali alla somma di quelli di m-1 e m.

3- Continuare così finché sono risolti m-1 problemi. Nell'ultimo problema la macchina fittizia 1 contiene le macchine da 1 a m-1 mentre la 2 le macchina da 2 a m.
4- Calcolare il makespan per ogni problema e scegliere la sequenza migliore.

Un problema di minimizzazione del makespan con un arbitrario numero di macchine può essere formulato come un problema di programmazione intera lineare. Una possibile formulazione proposta da Pinedo, 2008 è la seguente. Si noti che in questo caso k non rappresenta la macchina k-esima ma serve a identificare la posizione del job nella sequenza. È necessario innanzitutto definire una serie di variabili:

- le variabili decisionali x_{jk} uguali a 1 se il job j è k-esimo nella sequenza dei job, 0 altrimenti.
- La variabile ausiliaria $I_{i(k)}$ che denota il tempo di ozio della macchina i fra l'esecuzione del job in k-esima posizione e l'esecuzione del job in k-esima+1 posizione.
- La variabile ausiliaria $W_{i(k)}$ che denota il tempo di attesa del job k-esimo nella sequenza fra la macchina i e la macchina i+1.

Esiste una relazione fra W e I che rappresenta un vincolo nella formulazione MIP del problema. Tale relazione è rappresentata in figura 2.3.5 ed è la seguente:

$$\Delta_{ik} = I_{ik} + p_{i(k+1)} + W_{i,k+1} = W_{ik} + p_{i+1(k)} + I_{i+1,k}$$

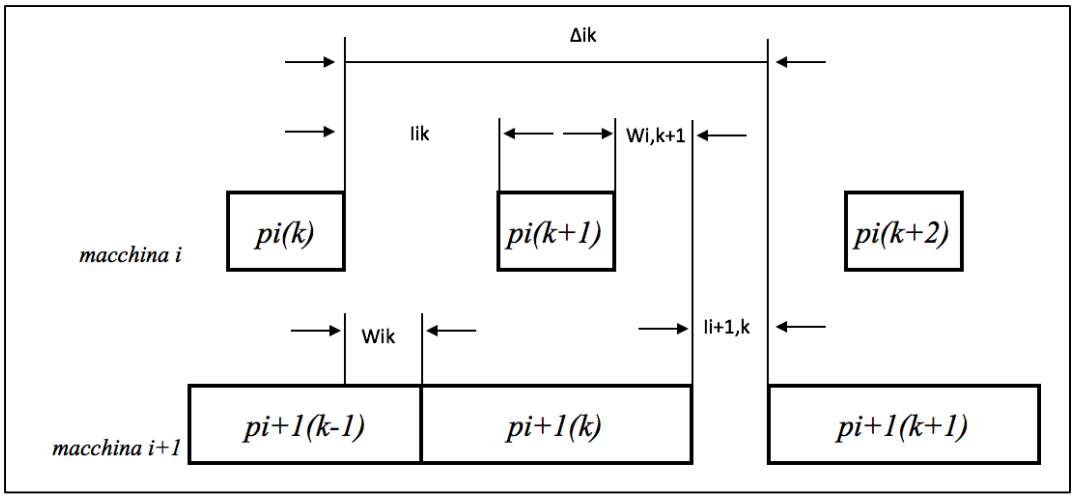


Fig. 2.3.5 Formulazione MIP per la schedulazione di un flow shop adattato da (Pinedo 2008)

In particolare, minimizzare il makespan è equivalente a minimizzare il tempo di ozio totale sull'ultima macchina m. Il tempo di ozio totale sulla macchina m è dato da: $\sum_{i=1}^{m-1} p_{i(1)} + \sum_{j=1}^{n-1} I_{mj}$.

Il primo termine consiste nel tempo che la macchina m aspetta prima che il job in prima posizione della sequenza arrivi alla macchina m; il secondo termine è la somma dei tempi di ozio sulla macchina m per tutti i jobs.

Vale inoltre la relazione: $p_{i(k)} = \sum_{j=1}^{n} x_{jk} p_{ij}$ che permette che le variabili siano x_{jk}.

La formulazione MIP è la seguente:

$$\min \left(\sum_{i=1}^{m-1} \sum_{j=1}^{n} x_{j1} p_{ij} + \sum_{j=1}^{n-1} I_{mj} \right)$$

soggetta a:

$$\sum_{j=1}^{n} x_{jk} = 1 \; con \; k = 1, \dots, n.$$

$$\sum_{k=1}^{n} x_{jk} = 1 \; con \; j = 1, \dots, n.$$

$$I_{ik} + \sum_{j=1}^{n} x_{j,k+1} p_{ij} + W_{i,k+1} - W_{ik} - \sum_{j=1}^{n} x_{jk} p_{i+1,j} - I_{i+1,k} = 0$$

$$con\ k = 1, \dots, n-1; i = 1, \dots, m-1$$

$$W_{i1} = 0\ con\ i = 1, \dots, m-1$$

$$I_{1k} = 0\ con\ k = 1, \dots, n-1$$

$$x_{ij} intero$$

$$W_{ik}\ e\ I_{ik} \geq 0$$

Il primo vincolo stabilisce che esattamente un job deve essere assegnato alla posizione k per ogni k.

Il secondo vincolo specifica che il job j deve essere assegnato esattamente a una posizione.

Il terzo vincolo permette di mettere in relazione le variabili x_{jk} ai vincoli fisici rappresentati dal tempo di ozio e di attesa (Pinedo 2008).

2.5 Schedulazione di un Job Shop

Un job shop è costituita da n jobs e m macchine ognuna delle quali può realizzare tipi di operazioni diverse ma può processare soltanto un'operazione alla volta.

Ogni job i è composto da q_i operazioni in serie in modo tale che ciascuna coppia di operazioni consecutive venga effettuata su macchine distinte.

Un job shop è definito da:

- Una matrice dei tempi di processamento P=$\{p_{ij}\}$, dove p_{ij} rappresenta il tempo di processamento dell'operazione j del job i.
- Una matrice di routing R=$\{r_{ij}\}$ dove l'elemento generico r_{ij} rappresenta la macchina su cui l'operazione j del job i deve essere eseguita.

Entrambe le matrici sono costituite da n righe rappresentanti il numero di jobs e un numero di elementi per ogni riga pari a q_i, ossia il numero di operazioni per ogni job i. Un esempio di matrice P con 7 jobs con le relative operazioni è:

Operazioni (j=1, 2, 3, 4)				
Job (i=1,…,7)	1	2	3	4
1	5	1	3	6
2	4	5	6	
3	8	2	2	
4	4	2	1	2
5	1	7		
6	2	4	4	
7	1	5	3	
Matrice dei tempi di processamento				

Un esempio di matrice R, dove ogni elemento della matrice rappresenta il numero della macchina su cui l'operazione j del job i deve essere eseguita, è la seguente:

Operazioni (j=1, 2, 3, 4)				
Job (i=1,…,7)	1	2	3	4
1	1	3	2	4
2	2	3	1	
3	1	2	3	
4	1	4	1	2
5	3	1		
6	2	4	1	
7	4	3	1	
Matrice di routing				

Dalla matrice R di routing si evince che il numero di macchina è 4 (m=4).
Con numero elevato di macchine i problemi di job shop risultano particolarmente complessi e fanno parte dei problemi cosiddetti NP-hard, nondeterministic

polynomial-time hard problem ossia problemi difficili non risolvibili in tempi polinomiali. Per il problema $J_m||C_{max}$ un euristico utilizzabile è il metodo delle liste di priorità già utilizzato in precedenza e estendibile ai job shop. Il metodo consiste nel definire, dopo che una macchina k ha terminato l'operazione, un insieme O_k delle operazione assegnabili e scegliere a seconda della regola di priorità desiderata. Un altro metodo è quello di generazione di soluzioni con particolari proprietà, come ad esempio soluzioni attive o senza ritardo (paragrafo 1.3). Il metodo di generazione di soluzioni attive permette di ottenere un numero arbitrario di soluzioni attive. Ad ogni iterazione si individuano le operazioni che devono essere schedulate per evitare che la soluzione non sia attiva. Se ne esistono diverse, si sceglie a caso o in base a una regola di priorità.

Si definisce r_{ijk} e C_{ijk} rispettivamente il tempo di rilascio e il tempo di completamento dell'operazione (i,j,k). Sia S_t la schedulazione già definita relativa alle t-1 operazioni e O_t l'insieme delle operazioni ammissibili all'iterazione t per il fatto che tutte le operazioni precedenti sono in S_t. Il tempo di rilascio r_{ijk} di un operazione $(i,j,k \in Ot)$ è dato dal massimo tra il tempo di completamento dell'operazione (i.j-1,k') già eseguita e il tempo di completamento dell'ultima operazione effettuata sulla macchina k. L'algoritmo costruttivo consiste in:

1: inizializzazione

Si pone t=1, S1={}, Ot costituito dall'insieme delle prime operazioni di ogni job $Ot = \{(i,1,k): i \in J\}$.

2: selezione di un nuovo elemento da aggiungere alla soluzione parziale

Si calcola $C(Ot) = \min_{(i,j,k)\in Ot}\{r_{ijk} + p_{ijk}\}$ ossia il minimo dei tempi di completamento associati ad ogni operazione ammissibile e si indica con k' la macchina corrispondente a C(Ot) e con $Ot' = \{(i,1,k'): (i,j,k') \in Ot, r_{ijk} < C(Ot)\}$. Si schedula al più presto un operazione $(i',j',k') \in Ot'$ a caso o secondo un criterio prestabilito e si pone $S_{t+1} = S_t \cup \{(i',j',k')\}$, $O_{t+1} = O_t - \{(i',j',k')\} \cup \{i',j'+1,k'\}$.

3: criterio di arresto

Si arresta la schedulazione quando Ot={} altrimenti si passa al punto 2.

Se al passo 2 esistono più macchine k' associate a C(Ot), l'insieme di Ot' sarà caratterizzato da tutte le operazioni di Ot associate a k' e verrà scelta l'operazione (i',j',k') secondo una qualsiasi regola di priorità.

Un approccio simile è il metodo per la generazione di soluzioni senza ritardo che permette di generare soluzioni in cui nessuna macchina viene lasciata inattiva se può processare delle operazioni. L'algoritmo è simile al precedente ma con una modifica del passo 2.

1: inizializzazione

Si pone t=1, S1={}, Ot costituito dall'insieme delle prime operazioni di ogni job $Ot = \{(i, 1, k): i \in J\}$.

2: selezione di un nuovo elemento da aggiungere alla soluzione parziale

Si calcola $r_{(i',j',k')\in Ot} = \min_{(i,j,k')\in Ot}\{r_{ijk}\}$ e si pone $S_{t+1} = S_t \cup \{(i', j', k')\}$ e $O_{t+1} = O_t - \{(i', j', k')\} \cup \{i', j' + 1, k'\}$ e t=t+1.

3: criterio di arresto

Si arresta la schedulazione quando Ot={} altrimenti si passa al punto 2.

Se più operazioni hanno lo stesso valore di $r_{(i',j',k')}$ se ne sceglie una a caso o secondo una qualsiasi regola di priorità.

Esempio:

Matrice dei tempi di processamento

Job	1	2	3
1	3	4	2
2	3	4	3
3	2	2	4
4	3	2	1

Matrice di routing

Job	1	2	3
1	1	3	2
2	3	2	1
3	2	1	3
4	2	1	3

a) Generazione di schedulazione attiva:

It.	Ot				Ci,j-1,k				Ck				rijk				pijk				Cijk				C(Ot)	k'	Regola MOPNR Ot'	i''j'k'	ri',j',k'	Ci''j'k'	
t	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4							
1	111	213	312	412	0	0	0	0	0	0	0	0	0	0	0	0	3	3	2	3	3	3	2	3	2	2	312,412	312	0	2	
2	111	213	321	412	0	0	2	0	0	2	0	0	0	0	2	2	3	3	2	3	3	3	4	5	3	1,3	111,213,321	213	0	3	
3	111	222	321	412	0	3	2	0	0	2	3	0	3	3	2	2	3	4	2	3	3	7	4	5	3	1	111,321	111	0	3	
4	123	222	321	412	3	3	2	0	3	2	3	0	3	3	3	2	4	4	2	3	7	7	5	5	5	1,2	222,321,412	222	3	7	
5	123	231	321	412	3	7	2	0	3	7	3	3	4	3	3	7	4	3	2	3	7	10	5	10	5	1	321	321	3	5	
6	123	231	333	412	3	7	5	0	5	7	3	3	3	7	5	7	4	3	4	3	7	10	9	10	7	3	123,333	123	3	7	
7	132	231	333	412	7	7	5	0	5	7	7	7	7	7	7	7	2	3	4	3	9	10	11	10	9	2	132,412	412	7	10	
8	132	231	333	421	7	7	5	10	5	10	7	10	7	7	7	10	2	3	4	1	12	10	11	11	10	1	231	231	7	10	
9	132		333	421	7	10	5	10	10	10	7	10	10		7	10	2		4	2	12		11	12	11	3	333	333	7	11	
10	132			421	7	10	11	10	10	10	11	10				10	2			2	12			12	12	1,2	132,421	421	10	12	
11	132			433	7	10	11	12	12	10	11	10				12	2			1	12			13	12	2	132	132	10	12	
12				433	7	12	11	12	12	12	11					12				1				13	13	1	433	433	12	13	

b) Generazione di schedulazione senza ritardo:

Regola MOPNR

It.	Ot				Ci,j-1,k				Ck				rijk				pijk				Cijk				C(Ot)	k'	Ot'	i'j'k'	ri',j',k'	Ci'j'k'	
t	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4							
1	111	213	312	412	0	0	0	0	0	0	0	0	0	0	0	0	3	3	2	3	3	3	2	3	2	2	312,412	312	0	2	
2	111	213	321	412	0	0	2	0	0	2	0	0	0	0	2	2	3	3	2	3	3	3	4	5	3	1,3	111,213,321	213	0	3	
3	111	222	321	412	0	3	2	0	0	2	3	0	0	3	2	2	3	4	2	3	3	7	4	5	3	1	111,321	111	0	3	
4	123	222	321	412	3	3	2	0	3	2	3	0	3	3	3	2	4	4	2	3	7	7	5	5	5	1,2	222,321,412	222	3	7	
5	123	231	321	412	3	7	2	0	3	7	3	0	3	7	3	7	4	3	2	3	7	10	5	10	5	1	321	321	3	5	
6	123	231	333	412	3	7	5	0	5	7	3	0	3	7	5	7	4	3	4	3	7	10	9	10	7	3	123,333	123	3	7	
7	132	231	333	412	7	7	5	0	5	7	7	0	7	7	7	7	2	3	4	3	9	10	11	10	9	2	132,412	412	7	10	
8	132	231	333	421	7	7	5	10	5	10	7	7	10	7	7	10	2	3	4	1	12	10	11	11	10	1	231	231	7	10	
9	132		333	421	7	10	5	10	10	10	7	10	10		7	10	2		4	2	12		11	12	11	3	333	333	7	11	
10	132			421	7	10	11	10	10	10	11	10	10			10	2			2	12			12	12	1,2	132,421	421	10	12	
11	132			433	7	10	11	12	12	10	11	10	10			12	2			1	12			13	12	2	132	132	10	12	
12				433	7	12	11	12	12	12	11					12				1				13	13	1	433	433	12	13	

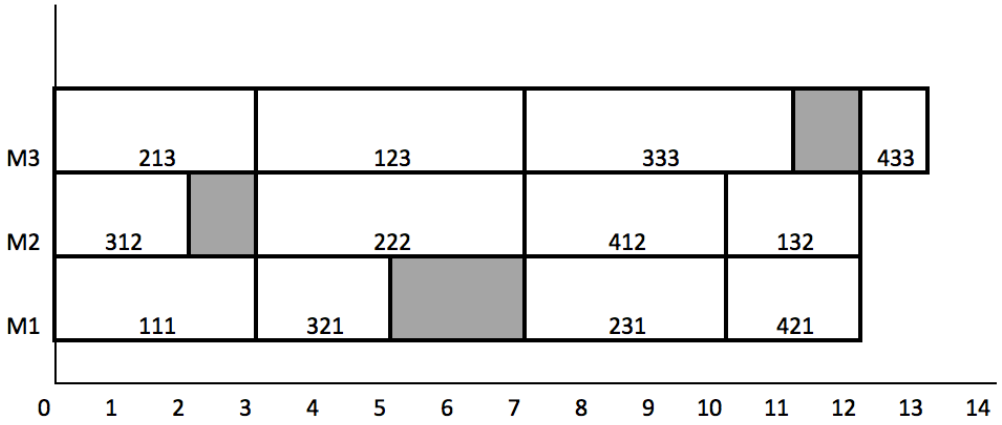


Fig. 2.5.1 Esempio schedulazione con algoritmo di generazione di soluzioni attive (elaborazione personale)

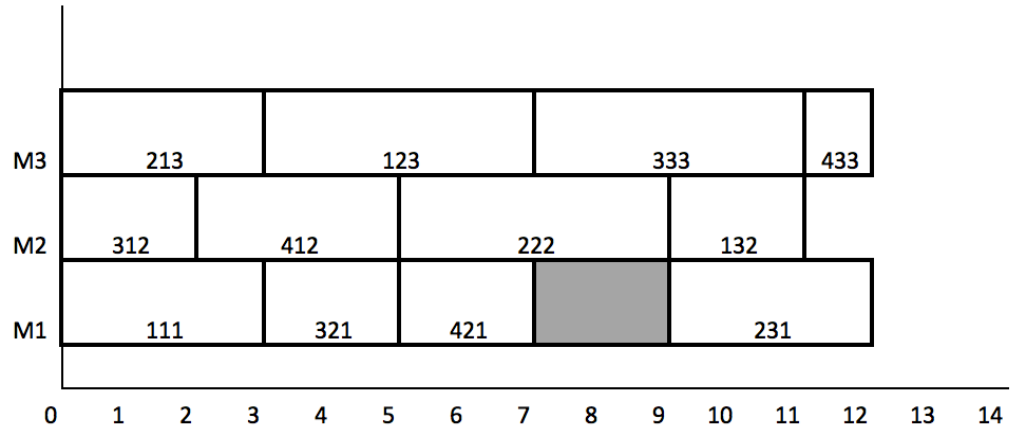


Fig. 2.5.2 Esempio schedulazione con algoritmo di generazione di soluzioni senza ritardo (elaborazione personale)

Una schedulazione di un job shop può essere inoltre rappresentata tramite un grafo orientato G = (V, A, D), in cui ogni operazione (i, j, k) corrisponde ad un nodo (i, j, k) del grafo. Possono essere presenti due nodi fittizi di inizio e fine schedulazione. V rappresenta l'insieme dei vertici del grafo, gli archi dell'insieme A, detti congiuntivi, rappresentano le relazioni di precedenza tra operazioni dello stesso job, mentre l'insieme degli archi D, detti disgiuntivi, uniscono, in entrambe

le direzioni, due operazioni da effettuare sulla stessa macchina. Ad ogni arco degli insiemi A e D uscente da (i, j, k) è associato un costo pari al tempo di processamento dell'operazione (i, j, k). Una schedulazione consiste nel assegnare un verso agli archi dell'insieme D in maniera che non vi siano cicli, poiché, assegnare un verso implica assegnare una sequenza a due operazioni eseguite sulla stessa macchina e la presenza di un ciclo implicherebbe una soluzione non ammissibile. Definito un orientamento, il percorso di lunghezza massimo dal nodo I di inizio al nodo F di fine rappresenta il makespan C_{max}. Tale percorso è detto percorso critico e può essere individuato con tecniche di tipo PERT tipiche della schedulazione di progetti. Il problema Jm||C_{max} quindi si riduce alla definizione dell'orientamento da attribuire ad un insieme di archi, l'insieme D, per minimizzare la lunghezza del percorso massimo tra I, nodo di inizio, e F, nodo di fine (Bruno 2012).

Matrice tempi di processamento				
	operazioni			
jobs	1	2	3	4
1	5	2	4	3
2	4	3		
3	6	4	4	5

Matrice di routing				
	operazioni			
jobs	1	2	3	4
1	3	1	2	4
2	3	2		
3	1	3	4	2

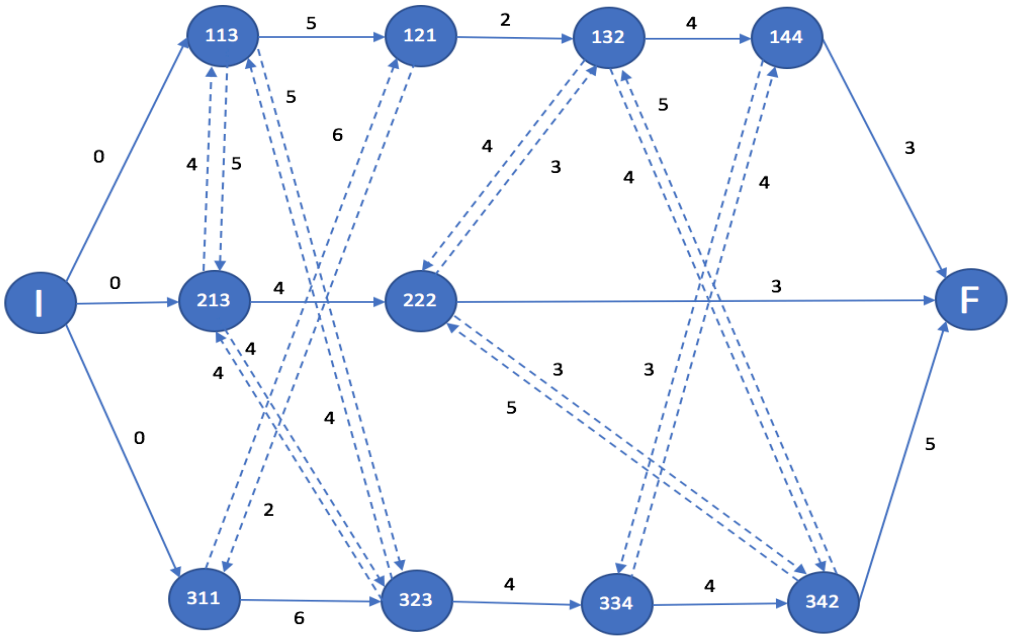


Fig. 2.5.3 Grafo di un problema di schedulazione job shop (Bruno 2012)

Una possibile formulazione di programmazione matematica connessa a tale rappresentazione è la seguente:

$$\min(Cmax)$$

soggetto a:

$y_{kj} - y_{ij} \geq p_{ij} \qquad \forall\, (i,j) \rightarrow (k,j) \epsilon A$

$Cmax - y_{ij} \geq p_{ij} \qquad \forall\, (i,j) \epsilon V$

$y_{ij} - y_{il} \geq p_{il}$ oppure $y_{il} - y_{ij} \geq p_{ij} \;\; \forall\, (i,l)\; e\; (i,j), \quad i = 1, \dots, m$

$y_{ij} \geq 0 \;\; \forall\, (i,j) \epsilon V$

Tale formulazione è detta di programmazione matematica disgiuntiva a causa del terzo vincolo di tipo disgiuntivo; per la soluzione vengono utilizzate le stesse tecniche della programmazione matematica intera.

Nella formulazione, la prima equazione rappresenta il vincolo di precedenza, ossia che lo starting time di (k,j) deve essere maggiore dello starting time di (i, j), sommato al tempo di processamento dell'operazione (i, j), se esiste una relazione di precedenza $(i,j) \rightarrow (k,j)$. La seconda stabilisce che il makespan è maggiore o

uguale della somma dello starting time e del tempo di processamento per ogni operazione/vertice.
La terza equazione assicura che esista un ordine tra le operazioni di job diversi che devono essere processate sulla stessa macchina. Questa equazione rappresenta il vincolo sull'orientamento degli archi appartenenti all'insieme D del grafo G = (V, A, D).
Uno degli euristici più usati per il problema $J_m||C_{max}$ è il Shifting Bottleneck Heuristic, o Euristico del collo di bottiglia mobile. L'algoritmo consiste nei seguenti passi:
Passo 1: Condizioni iniziali
$M_0=Ø$, dove M è l'insieme delle macchine m e M0 è un sottoinsieme di M costituito dalle macchine già sequenziate.
G, grafo con sono gli archi congiuntivi e non quelli disgiuntivi.
$C_{max}(M_0)$ è uguale al percorso più lungo in G.

Passo 2: Analisi delle macchine ancora da schedulare
Per ogni macchina i in M-M_0 generare un'istanza di $1|r_j|L_{max}$ con r_j dell'operazione (i, j) determinata dal percorso più lungo nel grafo G dal nodo di origine I al nodo (i, j) e con data di consegna di (i, j) determinata da $C_{max}(M_0)$ meno il percorso più lungo nel grafo G dal nodo (i, j) a nodo di fine F, sommato p_{ij}.
Per minimizzare L_{max} in ogni sottoproblema di singola macchina, si può utilizzare la regola EDD come visto nei paragrafi 2.2 e 2.4.
Sia $L_{max}(i)$ il minimo L_{max} nel sottoproblema corrispondente alla macchina i, minimizzato ad esempio grazie alla regola di priorità EDD.

Passo 3: Scelta del collo di bottiglia e sequenziamento
Sia:

$$L_{max}(k) = \max_{i \in \{M-M_0\}} (L_{max}(i))$$

sequenziare la macchina k secondo la sequenza ottenuta al passo 2 per quella macchina.

Inserire tutti i corrispondenti archi disgiuntivi nel grafo G. Inserire la macchina k in M_0.

Passo 4: Risequenziare tutte le macchine schedulate precedentemente
Al fine di ridurre ulteriormente C_{max}, per ogni macchina $i \epsilon \{M_0 - k\}$:
cancellare da G gli archi disgiuntivi corrispondenti alla macchina i;
formulare un problema di singola macchina con date di rilascio e date di consegna determinate dai percorsi più lunghi nel grafo G.
Trovare la sequenza che minimizza $L_{max}(i)$ e inserire gli archi disgiuntivi corrispondenti nel grafo G.

Passo 5: Criterio di stop
Se M_0=M STOP, altrimenti andare al passo 2.
(Pinedo 2008).
In seguito si mostra un esempio di applicazione dell'algoritmo (adattato da Pinedo, 2008).

Matrice tempi di processamento				
	operazioni			
jobs	1	2	3	4
1	10	8	4	
2	8	3	5	6
3	4	7	3	

Matrice di routing				
	operazioni			
jobs	1	2	3	4
1	1	2	3	
2	2	1	4	3
3	1	2	4	

Iterazione 1:
M_0=Ø
G senza archi disgiuntivi:

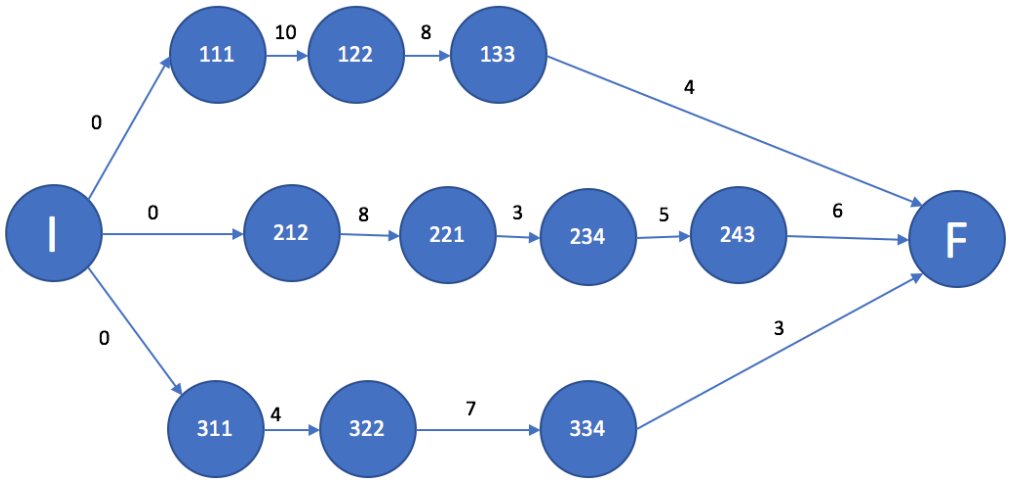


Fig. 2.5.4 Shifting bottleneck heuristic (a) (adattato da Pinedo, 2008)

Cmax=22 (massimo dei tempi di processamento fra i 3 jobs)

problema 1|rj|Lmax su m=1

Dati per 1|rj|Lmax su m=1

jobs	1	2	3
p1j	10	3	4
r1j	0	8	0
d1j	10	11	12

Sequencing di 1|rj|Lmax

t	Ot	Assegnoi	C(i)	L(i)
0	1,3	1	10	0
10	2,3	2	13	2
13	3	3	17	5
17				

problema 1|rj|Lmax su m=2

Dati per 1|rj|Lmax su m=2

jobs	1	2	3
p2j	8	8	7
r2j	10	0	4
d2j	18	8	19

Sequencing di 1|rj|Lmax

t	Ot	Assegnoi	C(i)	L(i)
0	2	2	8	0
8	3	3	15	-4
15	1	1	23	5
23				

Lmax(1)=5	Lmax(2)=5

problema 1|rj|Lmax su m=3

Dati per 1|rj|Lmax su m=3

jobs	1	2
p3j	4	6
r3j	18	16
d3j	22	22

Sequencing di 1|rj|Lmax

t	Ot	Assegno i	C(i)	L(i)
0				
16	2	2	22	2
22	1	1	26	4
26				
Lmax(3)=4				

problema 1|rj|Lmax su m=4

Dati per 1|rj|Lmax su m=4

jobs	2	3
p4j	5	3
r4j	11	11
d4j	16	22

Sequencing di 1|rj|Lmax

t	Ot	Assegno i	C(i)	L(i)
0				
11	2,3	2	16	0
16	3	3	19	-3
19				
Lmax(4)=0				

Max(Lmax(i))=5 su macchina 1 o 2.

Il bottleneck è la macchina 1 o 2. Viene scelta arbitrariamente la macchina 1.

$M_0=\{1\}$.

Si aggiungono i corrispondenti archi disgiuntivi a G con la sequenza individuata.

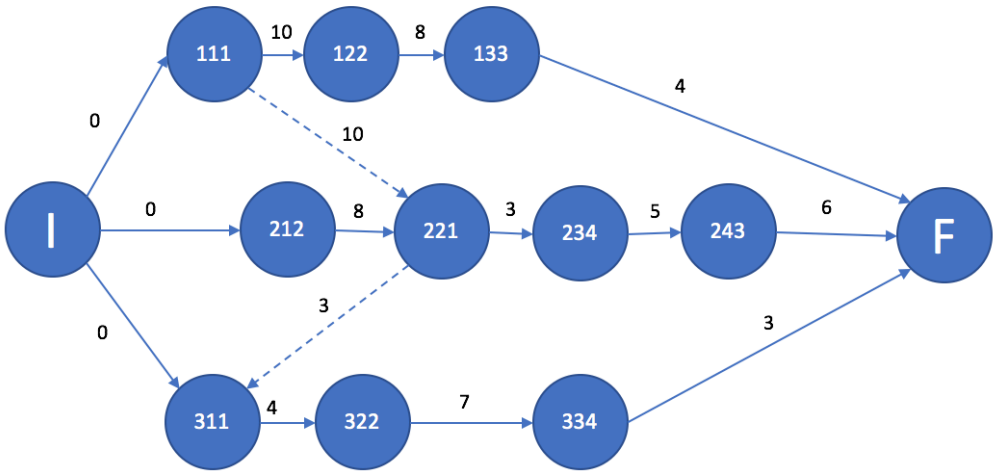


Fig. 2.5.5 Shifting bottleneck heuristic (b) (adattato da Pinedo, 2008)

$$Cmax(M_0 = \{1\}) = Cmax(\{\}) + Lmax(1) = 22 + 5 = 27$$

Iterazione 2:

M_0={1}

Dati per 1|rj|Lmax su m=2

jobs	1	2	3
p2j	8	8	7
r2j	10	0	17
d2j	23	10	24

Dati per 1|rj|Lmax su m=3

jobs	1	2
p3j	4	6
r3j	18	18
d3j	27	27

Dati per 1|rj|Lmax su m=4

jobs	2	3
p4j	5	3
r4j	13	24
d4j	21	27

Sequencing di 1|rj|Lmax

t	Ot	Assegno	C(i)	L(i)
0	2	2	8	-2
8				
10	1	1	18	-5
18	3	3	25	1
25				
Lmax(2)=1				

Sequencing di 1|rj|Lmax

t	Ot	Assegno i	C(i)	L(i)
0				
18	1,2	1	22	-5
22	2	2	28	1
28				
Lmax(3)=1				

Sequencing di 1|rj|Lmax

t	Ot	Assegno i	C(i)	L(i)
0				
13	2	2	18	-3
18				
24	3	3	27	0
27				
Lmax(4)=0				

Il bottleneck è la macchina 2 o 3. Si sceglie arbitrariamente la macchina 2.

M_0={1,2} Cmax(M_0)=27+1=28
Si aggiungono gli archi disgiuntivi di 2 con la sequenza individuata e si ottiene:

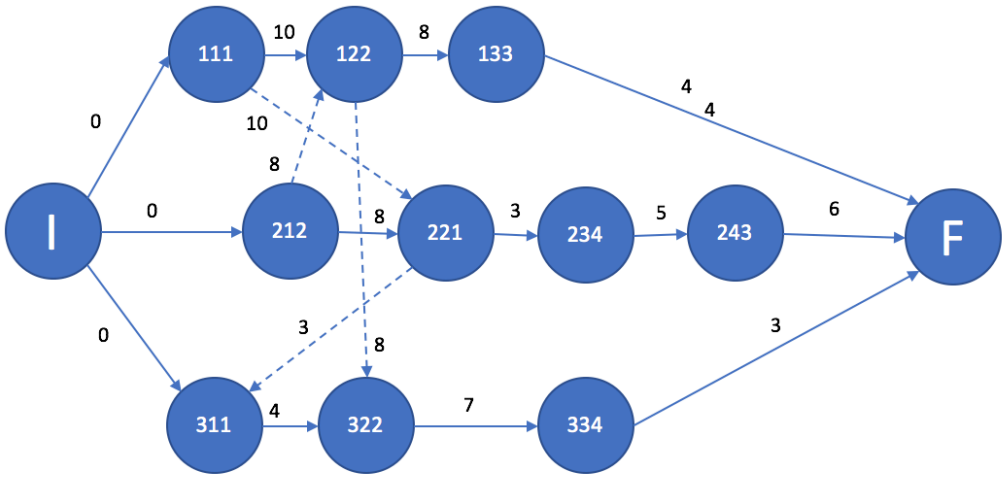


Fig. 2.5.6 Shifting bottleneck heuristic (c) (adattato da Pinedo, 2008)

Nel tentativo di diminuire C_{max} si va a risequenziare la macchina 1 secondo il passo 4 eliminando gli archi disgiuntivi e risequenziando con i nuovi archi inseriti nell'iterazione 2. In questo caso, ciò non va a migliorare il valore di C_{max}.

Iterazione 3:

M_0={1,2} Cmax=28

Dati per 1|rj|Lmax su m=3

jobs	1	2
p3j	4	6
r3j	18	18
d3j	28	28

Dati per 1|rj|Lmax su m=4

jobs	2	3
p4j	5	3
r4j	13	25
d4J	22	28

Sequencing di 1|rj|Lmax

T	Ot	Assegno i	C(i)	L(i)
0				
18	1,2	2	24	-4
24	1	1	28	0
28				
Lmax(3)=0				

Sequencing di 1|rj|Lmax

t	Ot	Assegno i	C(i)	L(i)
0				
13	2	2	18	-4
18				
25	3	3	28	0
28				
Lmax(4)=0				

Nessuna dei due rappresenta un collo di bottiglia. Posso andare a sequenziarla entrambe. Quindi M_0={1,2,3,4} Cmax=28.

Sequenza finale:

m	Ordine		
1	1	2	3
2	2	1	3
3	2	1	
4	2	3	

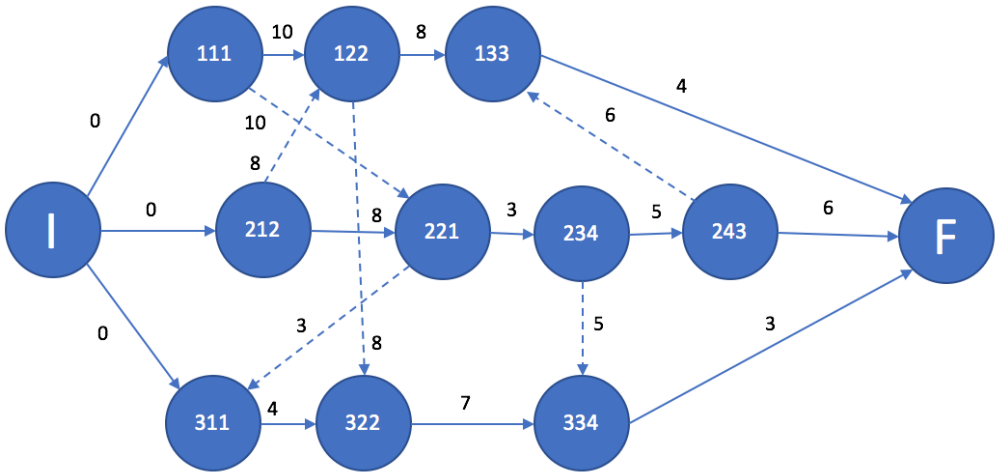


Fig. 2.5.6 Shifting bottleneck heuristic (d) (adattato da Pinedo, 2008)

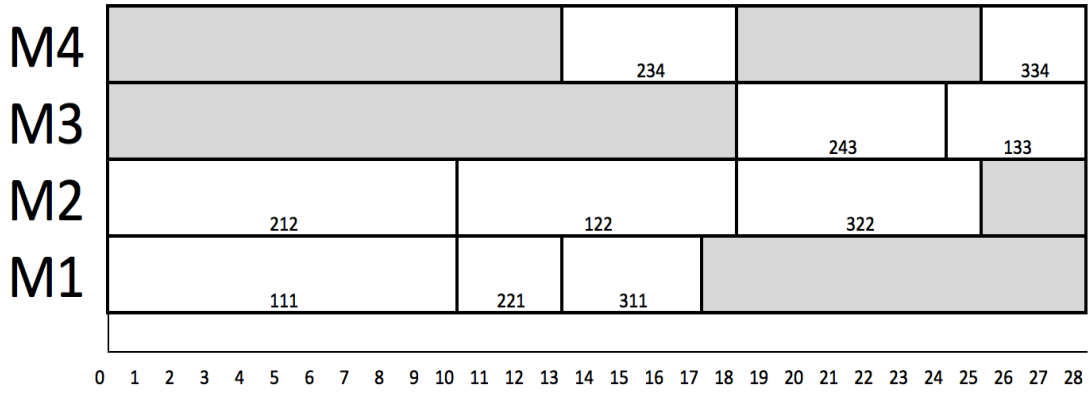


Fig. 2.5.7 Shifting bottleneck heuristic, diagramma di Gantt (e) (elaborazione personale)

Nel caso la funzione obiettivo da ottimizzare fosse la weighted total tardiness, ossia il ritardo totale pesato $\sum w_j T_j$, è possibile utilizzare una variante dello shifting bottleneck heuristic (SBH), già utilizzato per minimizzare il makespan. La prima differenza è che nel problema Jm||C_{max} solo il tempo di completamento dell'ultimo pezzo che lascia il sistema è importante, per questo esiste solamente un nodo di fine, mentre nel problema Jm||$\sum w_j T_j$ il tempo di completamento di ogni

job è importante e contribuisce al valore della funzione obiettivo, per questo non esiste un unico nodo di fine ma esistono n nodi pari al numero di jobs. La lunghezza del percorso più lungo dal nodo di inizio I al nodo V_k rappresenta il tempo di completamento del job k. La differenza principale con lo SBH classico consiste nella maniera in cui vengono definiti i sottoproblemi di scheduling di singola macchina e nella maniera in cui viene misurata la criticità di una macchina quando si va per scegliere quale macchina è da schedulare (passo2). I sottoproblemi di singola macchina che prima erano del tipo $1|r_j|L_{max}$, ora vengono definiti come $1||\sum w_j T_j$, e affrontati con la regola di priorità Apparent Tardiness Cost, o regola ATC. La regola ATC stabilisce un indice di priorità per ogni operazione (i, j), con i=macchine e j=operazione, calcolato nella maniera seguente:

$$I_{ij}(t) = \sum_{k=1}^{n} \frac{w_k}{p_{ij}} \exp\left(-\frac{(d_{ij}^k - p_{ij} + \max(0; r_{ij} - t))}{K\overline{p}}\right)$$

dove t è il tempo più recente nel quale la macchina i può essere usata, K è un parametro di scala, $\overline{p}$ la parte intera della media fra i tempi di processamento sulla macchina i, w_k è il peso del job k e d_{ij}^k è la data di consegna locale ossia il percorso più lungo dall'operazione (i, j) al nodo V_k. Se non è presente nessun percorso da (i, j) a V_k d_{ij}^k è infinito. Le operazioni sulla macchina i vengono sequenziate a seconda dell'indice $I_{ij}(t)$, scegliendo prima quella con indice maggiore. Una volta analizzati i problemi di singola macchina con la regola ATC, e quindi assegnata una sequenza delle operazioni per ogni macchina analizzata e trattata singolarmente, si deve scegliere una regola per misurare la criticità di tali macchine. Un modo semplice può essere quello di misurare la criticità della macchina con il valore corrispondente della funzione obiettivo e scegliere quella più critica. Un ulteriore modo proposto da Pinedo, 2008 per determinare la criticità di una macchina è usare questa funzione:

$$\sum_{k=1}^{n} w_k(C''_k - C'_k)\exp\left(-\frac{\max(0; (d_k - C''_k)}{K}\right)$$

dove C'_k è il tempo di completamento prima di aggiungere la macchina i per cui si sta calcolando l'indice mentre C''_k è il tempo di completamento inserendo la macchina i. Viene scelta e inclusa in M_0 la macchina con più alto valore di criticità (Pinedo 2008).

L'algoritmo SBH per il problema Jm||$\sum w_j T_j$, può essere riassunto nella maniera seguente:

Passo 1: Condizioni iniziali

M_0=Ø, dove M è l'insieme delle macchine m e M_0 è un sottoinsieme di M costituito dalle macchine già sequenziate.

G, grafo con sono gli archi congiuntivi e non quelli disgiuntivi.

$C_{max}(M_0)$ è uguale al percorso più lungo in G.

Passo 2: Analisi delle macchine ancora da schedulare

Per ogni macchina i in M-M_0 generare un'istanza di 1| |$\sum w_j T_j$.

Per minimizzare $\sum w_j T_j$ in ogni sottoporoblema di singola macchina, si utilizza la regola ATC.

Passo 3: Scelta del collo di bottiglia e sequenziamento

Scegliere come collo di bottiglia la macchina più critica determinata secondo il valore della funzione obiettivo o secondo la funzione $\sum_{k=1}^{n} w_k(C''_k - C'_k)\exp\left(-\frac{\max(0;(d_k - C''_k)}{K}\right)$.

Inserire tutti i corrispondenti archi disgiuntivi nel grafo G. Inserire la macchina k più critica in M_0.

Passo 4: Risequenziare tutte le macchine schedulate precedentemente

Al fine di ridurre ulteriormente C_{max}, per ogni macchina $i \epsilon \{M_0 - k\}$:

cancellare da G gli archi disgiuntivi corrispondenti alla macchina i;

formulare un problema di singola macchina con date di rilascio e date di consegna determinate dai percorsi più lunghi nel grafo G.

Trovare la sequenza ottima con la regola ATC e inserire gli archi disgiuntivi corrispondenti nel grafo G.

Passo 5: Criterio di stop
Se M_0=M STOP, altrimenti andare al passo 2.

2.6 Schedulazione di un Open Shop

La caratteristica principale dell'open shop è che non ci sono restrizioni sul percorso che devono seguire le operazioni e ognuna può avere percorsi differenti. A differenza del job shop in cui ogni operazione a un percorso definito e fisso, detto route, nell'open shop il percorso che le operazioni devono seguire è a discrezione dello scheduler.
Si consideri il caso di open shop più semplice, ossia quello a 2 macchina $O2||C_{max}$. Il job j può essere processato indifferentemente prima sulla macchina 1 e poi sulla 2 o viceversa. In questi tipi di problemi con solamente due macchine e minimizzazione del makespan la regola Longest Alternate Processing Time first, o LAPT, che prevede che quando una macchina è libera si vada a processare il job che ha il tempo di processamento più lungo sulla macchina opposta, fornisce una soluzione ottima. Se un job è già stato processato sulla macchina opposta, questo ha la minima priorità sulla macchina appena liberata, ossia zero. La regola LAPT è un caso speciale che può essere applicata ai casi a 2 macchine della regola più generale Longest Total Remaining Processing on Other Machines first (LTRPOM), applicabile ai casi con m>2 ma che non fornisce una soluzione ottima a differenza di m=2. Per m>2, infatti, i problemi di open shop sono NP-hard, esclusi alcuni casi particolari come $O_m|r_j, p_{ij}=1|L_{max}$ in cui i tempi di processamento uguali a uno rendono il problema più semplice.

Esempio scheduling open shop a 3 macchine con 5 jobs:

	macchina		
jobs	1	2	3
1	4	3	4
2	4	3	2
3	3	3	3
4	4	2	5
5	6	8	2

Tabella con regola LTRPOM per l'assegnazione; x=già schedulato vuoto=non assegnabile perché momentaneamente assegnato ad un'altra macchina.

		LTRPOM(job j)					
t	macchina liberata	1	2	3	4	5	Assegno
0	1	7	5	6	7	10	5
0	2	8	6	6	9		4
0	3	7	7	6			1
2	2		6	6	x		2
4	3	x		6	4		3
5	2	4	x	3	x	2	1
6	1		2		5	x	4
7	3	x	4	x		8	5
8	2	x	x	3	x		3
9	3	x	4	x	0	x	2
10	1	3			x	x	1
11	2	x	x	x	x	0	5
11	3	x	x	x	0	x	4

14	1	x	0	0	x	x	2
16	3	x	x	x	x	x	fine
18	1	x	x	0	x	x	3
19	2	x	x	x	x	x	fine
21	1	x	x	x	x	x	fine

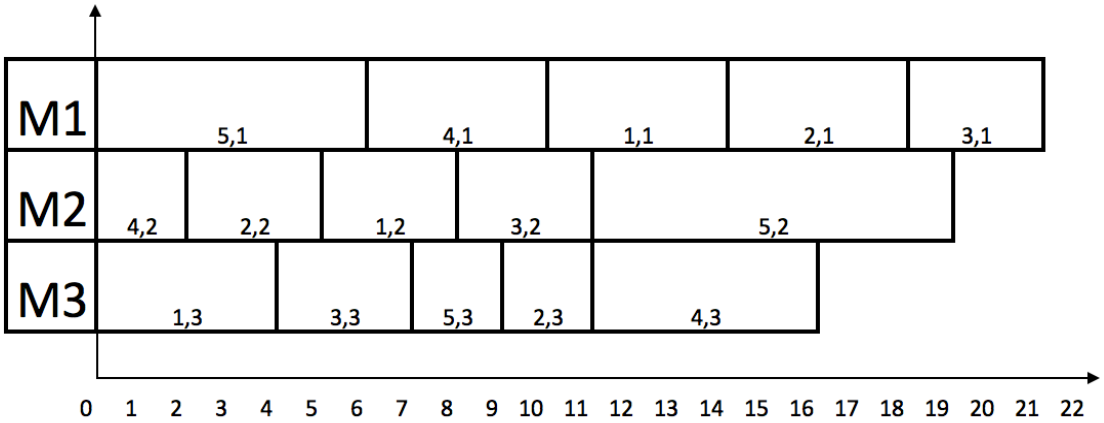


Fig. 2.6.1 Esempio scheduling open shop con regola LTRPOM (elaborazione personale)

Capitolo 3 - Constraint satisfaction problems e programmazione a vincoli

Affinché un problema di schedulazione della produzione possa essere ricondotto a un constraint satisfaction problem (CSP) è necessario definire il problema e quali sono i metodi di risoluzione utilizzati dalla programmazione a vincoli. Vengono presentati i principali algoritmi di ricerca e i metodi per migliorarne l'efficienza.

3.1 Definizione e rappresentazione del constraint satisfaction problem.

Come brevemente descritto nel paragrafo 1.6, un CSP è definito come un insieme di variabili $\{X_1,\ldots,X_n\}$, di vincoli $\{C_1,\ldots,C_m\}$ e di domini delle singole variabili $\{D_1,\ldots,D_n\}$ ossia i valori ammissibili per quelle variabili.

Uno stato del problema è definito come l'assegnazione di valori alle singole variabili, ad esempio X_i=vi, Xj=v_j e così via. Un'assegnazione può essere:

- consistente o legale, se non viola nessun vincolo
- completo se tutte le variabili hanno valori assegnati
- parziale se solo alcune variabili hanno valori assegnati
- soluzione se è sia completo che consistente.

Alcuni CSP necessitano inoltre di una funzione obiettivo da massimizzare o minimizzare.

Un esempio di CSP è il map coloring problem, che implica di colorare gli stati presenti su una mappa in modo che gli stati confinanti non abbiano lo stesso colore.

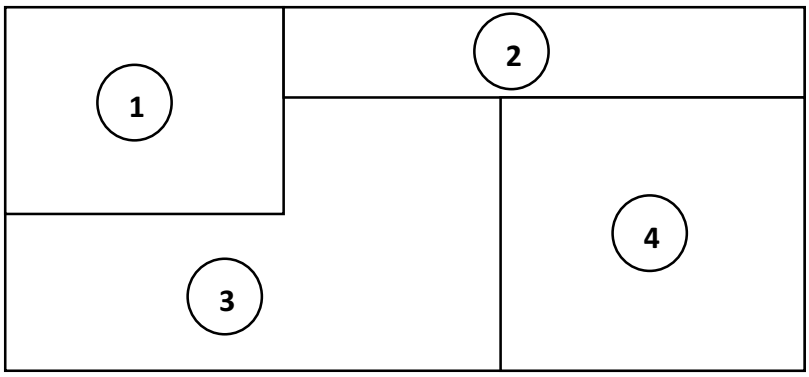


Fig. 3.1.1 Map coloring problem (elaborazione personale)

Le variabili sono rappresentate dalle regioni, ossia 1, 2, 3 e 4, i domini dai colori permessi, ossia rosso, verde e blu mentre i vincoli sono che le regioni adiacenti devono avere colori diversi.
È utile rappresentare un CSP con un grafo dove i nodi rappresentano le variabili e gli archi i vincoli. Per il map coloring problem precedente il grafo è mostrato in figura 2.1.2.

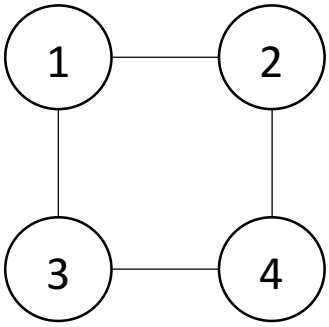


Fig. 3.1.2 Grafo del map coloring problem (elaborazione personale)

I CSP possono essere di diverso tipo, ad esempio con variabili booleane, discrete o continue, domini finiti o infiniti, vincoli lineari o non lineari etc.
Come esposto nel paragrafo 1.6 e 1.7, gli algoritmi di ricerca della programmazione a vincoli sono efficaci nel trattare alcuni tipi di CSP, come quelli con variabili discrete e domini finiti, tra cui i problemi di scheduling (Russel e

Norvig 2009). In seguito vengono delineate le tecniche e alcuni algoritmi utilizzati nell'affrontare i CSP.

3.2 Generate and test e Backtracking

Il generate and test è un algoritmo di ricerca generale e semplice per risolvere un CSP. Esso consiste essenzialmente nel generare ogni combinazione di valori da assegnare alle variabili e poi testare se ciò soddisfa i vincoli. La prima combinazione che soddisfa tutti i vincoli è la soluzione. Il numero di combinazioni considerate è dato dal prodotto cartesiano delle dimensioni dei domini delle variabili e quindi in presenza di numerose variabili e domini ampi il metodo risulta essere inefficiente, con complessità esponenziale (Kumar 1992). Si utilizza il seguente problema a titolo di esempio:

X, Y, Z sono le variabili con dominio {1,2}.

X=Y, X≠Z, Y>Z sono i vincoli.

In figura 2.2.1 viene rappresentato lo spazio di ricerca utilizzando un albero decisionale con all'interno del nodo il valore assunto dalla variabile. Il numero delle possibili combinazioni è dato dal prodotto cartesiano delle dimensioni dei domini di ogni variabile e quindi 2x2x2=8 combinazioni che corrispondono a 8 rami dell'albero.

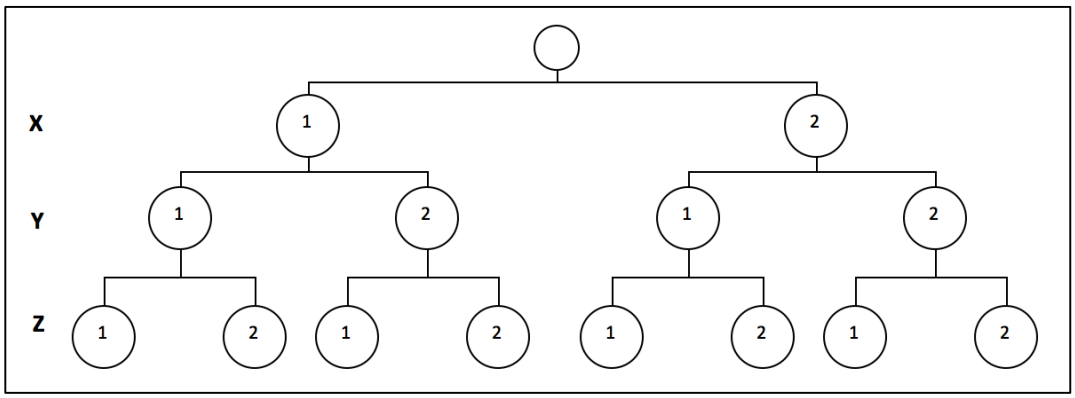


Fig. 3.2.1 Albero di ricerca (elaborazione personale)

Il generate and test per questo problema funziona come di seguito:

X	Y	Z	test
1	1	1	fail
1	1	2	fail
1	2	1	fail
1	2	2	fail
2	1	1	fail
2	1	2	fail
2	2	1	passed

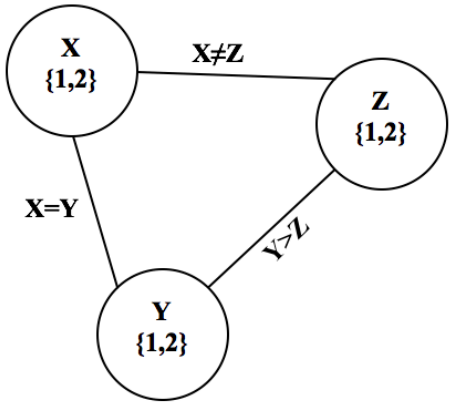
Vengono generati i valori delle variabili e viene poi testato il soddisfacimento dei vincoli a ogni iterazione.
La soluzione è dunque X=2, Y=2, Z=1.
Un algoritmo più efficiente per la ricerca è il backtracking. In questo caso le variabili sono istanziante sequenzialmente, ossia viene assegnato valore ad una variabile alla volta e si torna indietro quando una variabile viola i vincoli o quando non ci sono più valori assegnabili che non violano i vincoli. Quindi il backtracking prevede di tornare indietro quando l'assegnazione parziale non è consistente, ossia non soddisfa i vincoli.
Il backtracking applicato all'esempio precedente funziona come di seguito:

X	Y	Z	test
1	1	1	fail
		2	fail
	2		fail
2	1		fail
	2	1	passed

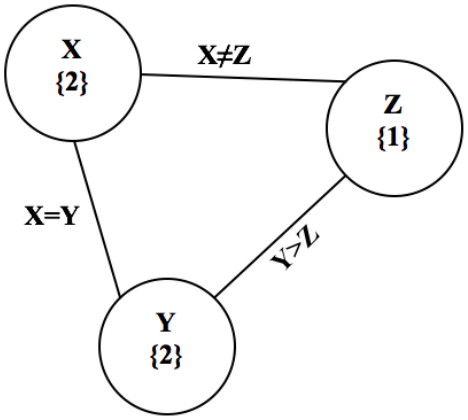
Viene trovata la medesima soluzione del generate and test con meno iterazioni. Sebbene il backtracking sia migliore del generate and test, la complessità è ancora esponenziale. Una delle cause prende il nome di thrashing ossia il fatto che le ricerche in diverse zone dello spazio continuano a fallire per le stesse ragioni. Il thrashing è dovuto a due cause: inconsistenza dei nodi e inconsistenza degli archi. L'inconsistenza dei nodi è data dalla presenza dei vincoli unari, ossia vincoli che influenzano un'unica variabile. Se un dominio D_i di una variabile X_i contiene il valore *a* e tale valore non soddisfa il vincolo unario su X_i allora l'assegnazione del valore *a* ad X_i risulterà sempre in una violazione dei vincoli (failure). Supponiamo ora che le variabili siano istanziate nell'ordine X_1, X_2,…, X_i,…X_j,... X_n e che ci sia un vincolo binario fra X_i e X_j tale che quando X_i=*a* non è ammesso nessun valore di X_j. Ogni volta che la variabile X_j viene istanziata ad *a* la ricerca fallirà sempre al momento di istanziare la variabile X_j perché nessuna valore è accettabile. Il thrashing dovuto all'inconsistenza dei nodi può essere rimosso semplicemente rimuovendo dal dominio di ogni variabile i valori che non soddisfano i vincoli unari. Il thrashing dovuto all'inconsistenza degli archi ha bisogno che ogni arco sia reso consistente prima di cominciare la ricerca. Esistono quindi delle tecniche che permettono di andare a ridurre lo spazio di ricerca prima di cominciare la ricerca o durante la ricerca stessa. Una di queste la la propagazione dei vincoli (Kumar 1992) (Russel e Norvig 2009).

3.3 La propagazione dei vincoli

Un arco è detto consistente se per ogni valore *a* del dominio di X_i esiste un valore *b* nel dominio di X_j tale che X_i=*a* e X_j=*b* sono permessi dal vincolo binario fra X_i e X_j. Tale consistenza è direzionale, ossia se l'arco (X_i;X_i) è consistente, ciò non implica che l'arco (X_i;X_i) sia consistente. Utilizzando l'esempio del paragrafo precedente si ottiene il seguente grafo con indicati i domini delle variabili e i vincoli.



Se si prende l'arco (Y,Z) si nota che per il valore Y=2 esiste il valore 1 che può essere assegnato a Z, soddisfando il vincolo. Per il valore Y=1 non esiste nessun valore di Z assegnabile e quindi si ha un'inconsistenza. Rimuovere tale inconsistenza implica la rimozione dal dominio di Y del valore, in questo caso 1. Se si prende ora l'arco (Z;Y), si nota che per il valore di Z=1 esiste il valore Y=2 che soddisfa il vincolo. Per il valore di Z=2 non esiste invece nessun valore di Y tale che Y>2 e quindi si ha un'inconsistenza che può essere rimossa rimuovendo dal dominio di Z il valore 2. Analogamente nel dominio di X viene rimosso il valore 1 per il vincolo con Z o con Y. Il grafo che ne risulta propagando quindi i vincoli è il seguente:



Se il dominio di una variabile rimane vuoto non ci sono soluzioni; se il dominio di tutte le variabili ha cardinalità uguale a 1, esiste un'unica soluzione. In questo caso esiste un'unica soluzione (2,2,1).
Quindi, un arco può essere reso consistente semplicemente eliminando dal dominio della variabile i valori che non soddisfano la condizione di consistenza. Tale eliminazione non elimina soluzioni del CSP originale ma ne riduce la complessità. La consistenza degli archi può essere applicata all'inizio del processo di ricerca o dopo ogni assegnazione durante la ricerca. Tuttavia essa non permette di identificare tutte le inconsistenze. I principali algoritmi utilizzati dalla programmazione a vincoli (ad esempio generate and test, simple backtracking, forward checking, partial lookahead, full lookahead, really full lookahead) differiscono per il grado di consistenza degli archi attuata ai nodi dell'albero di ricerca. Tali algoritmi risultano essere quindi una combinazione di pura ricerca e di consistenza.
Il generate and test (GT) non utilizza nessuna tecnica di consistenza ma solamente la ricerca pura. Nel backtracking (BT) si può notare un grado molto basso di consistenza degli archi. Ogni volta che una nuova variabile è considerata per essere istanziata, ogni valore del dominio inconsistente (che viola i vincoli) con ogni istanza precedente causa il fallimento della ricerca. Quindi il dominio della variabile da istanziare è filtrato per contenere solo i valori consistenti con le istanze "più alte" nell'albero di ricerca, ossia quelle precedente. Il forward checking (FC) utilizza un grado superiore di AC (consistenza degli archi) poiché prevede che ogni volta che una nuova variabile viene istanziata, i domini delle variabili future, ossia non ancora istanziate, siano filtrati in modo da contenere solo i valori che sono consistenti con questa istanza. Se il dominio di una di queste future variabili ancora da istanziare diventa vuoto, allora viene riconosciuto un fallimento e si attua quindi il backtracking. PL, FL e RFL sono versioni potenziare del FC poiché attuano l'AC anche fra variabili non ancora istanziate.
Una propagazione più forte prende il nome di k-consistenza. Un CSP è k-consistente se per ogni insieme di k-1 variabili e per ogni assegnazione a queste

variabili, un valore consistente può essere assegnato a ogni k-esima variabile. 1-consistente coincide con la consistenza del nodo ossia nella verifica dei vincoli unari. 2-consistente coincide con la consistenza di arco ossia nella verifica dei vincoli binari. Per k>2 si parla di k-consistenza, o consistenza di cammino, poiché vengono analizzati i vincoli lungo un cammino da un nodo di partenza ad un nodo di destinazione. Un CSP è k-consistente se per ogni insieme di k-1 variabili e per ogni assegnamento consistente su tali k-1 variabili, può essere assegnato un valore consistente ad ogni k-esima variabile.
Esempio con consistenza di cammino con k=3:
A:{0,1}, B:{1}, C{0,1,2}.
$A \neq B, B \neq C$
scegliamo l'insieme di k-1 (2) variabili A e C e vediamone gli assegnamenti possibili.

A	C
0	**0**
0	**1**
0	**2**
1	**0**
1	**1**
1	**2**

Sono tutti consistenti poiché non esiste nessun vincolo tra A e C. Si analizza la consistenza di cammino A-C tenendo conto dei vincoli con B.

A	C	B
0	**0**	**1**
0	**1**	**{}**
0	**2**	**1**

1	**0**	{}
1	**1**	{}
1	**2**	{}

Alcune combinazioni non sono 3-consistenti poiché il dominio di B diventa vuoto. È possibile eliminare i valori non consistenti dal dominio delle variabili A e C cosi le elaborazioni successive si concentrino su combinazioni consistenti.
Dall'insieme A si rimuove 1 poiché non esiste un'assegnazione di C tale che B sia non vuoto quando A=1; dall'insieme C si rimuove 1 poiché non esiste un'assegnazione di A tale che B sia non vuoto quando C=1.
I domini ridotti diventano A{0}, B{1}, C{0,2}. Il CSP trattato nell'esempio è dunque 3-consistente poiché ciò vale per ogni insieme di k-1 variabili e per ogni assegnazione delle k-1 variabili.
Un CSP è fortemente k-consistente se è k-consistente ed è anche k-1, k-2,…,1-consistente. Un grafo fortemente n-consistente permette di trovare una soluzione senza l'utilizzo del backtracking poiché è possibile scegliere un valore consistente per X1, poi per X2 essendo 2-consistente, poi per X3 essendo 3-consistente e cosi via fino a Xn. (Russel e Norvig 2009). Tale processo di n-consistenza è di solito più costoso del semplice backtracking. Nella pratica si utilizza dunque una combinazione delle due, con un appropriato livello di consistenza e il backtracking (Russel e Norvig 2009). Molti studi di ricerca infatti indicano che si hanno prestazioni migliori utilizzando la propagazione dei vincoli in maniera limitata (Kumar 1992).

3.4 Backtracking intelligente

Un'ulteriore causa che influenza l'efficienza del backtracking, oltre che al thrashing, è la presenza di lavoro ridondante. Il backtracking intelligente, o backjumping, permette di ridurre tale lavoro saltando direttamente all'assegnazione della variabile che causa il conflitto. Il backtracking una volta fallita la ricerca in un ramo, ossia quando il dominio dei valori assegnabili alla variabile diventa vuoto, torna alla variabile precedente e prova un valore differente. Per questo motivo questo tipo di backtracking è detto cronologico. Il backjumping, invece, una volta fallita la ricerca in un ramo perché il dominio rimane vuoto, esso ritorna non alla variabile precedente, ma alla variabile più recente presente nel conflict set. Il conflict set della variabile X è l'insieme delle variabili precedentemente assegnate che sono connesse a X da vincoli. Ciò permette di ridurre lo spazio di ricerca non andando ad esplorare rami che sono già destinati a fallire per le stesse ragioni. Infatti se la variabile X_{k-1} precedente non influenza il fallimento della variabile X_k, allora tornare indietro alla variabile X_{k-1} risulterà comunque in un fallimento, per questo il conflict set permette di identificare la variabile più recente che causa l'inconsistenza e saltare indietro a quella.

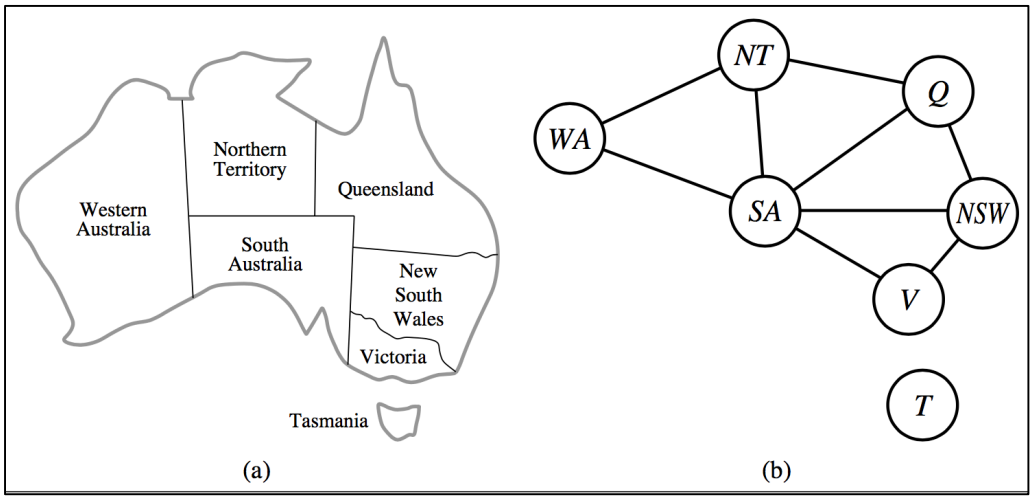


Fig. 3.4.1 Esempio map coloring problem (Russel e Norvig 2009)

Si consideri il problema in figura 2.4.1. Si tratta di colorare le diverse zone geografiche con tre colori, rosso, blue e verde, senza che due regioni adiacente abbiano lo stesso colore.
Si assuma il seguente ordine di assegnazione fisso Q, NSW, V, T, SA, WA, NT e il seguente assegnamento parziale {Q=rosso, NSW=verde, V=blu, T=rosso}.

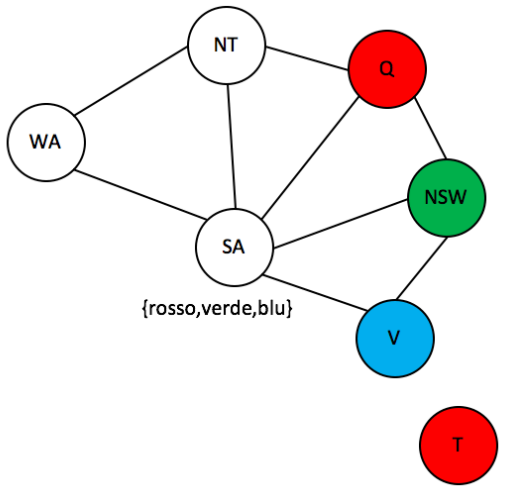
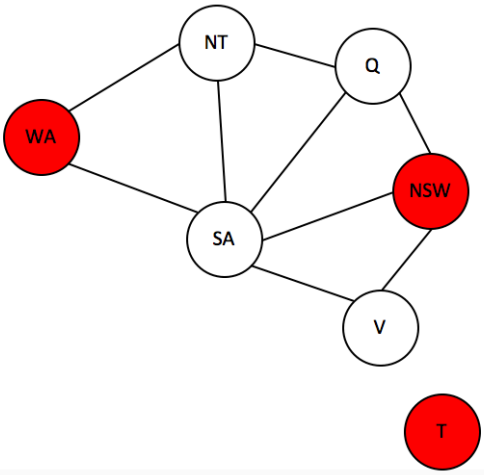


Fig. 3.4.2 Map coloring problem con backjumping (elaborazione personale)

Quando si va ad assegnare la variabile SA essa non può essere né rossa né verde né blu a causa di {Q, NSW, V} che rappresentano il suo conflict set. Si è arrivati quindi ad un nodo foglia morto, ossia senza possibilità di proseguire, non avendo SA nessuna soluzione parziale ammissibile. Il backtracking tornerebbe al nodo T, cambiando T=blu e riprovando ad esplorare SA inutilmente per poi riprovare T=verde e riesplorare SA ancora una volta inutilmente poiché non è T ad influenzare l'ammissibilità di SA. Il backjumping arrivando al nodo foglia morto SA che ha dominio vuoto non torna a T ma alla variabile più recente, ossia inizializzata per ultima, presente nel conflict set. Ecco quindi che l'assegnazione salterà T per andare a riassegnare V. Il risultato è che non si va ad esplorare inutilmente rami destinati a fallire.

Un ulteriore miglioramento del backjumping è il conflict-directed backjumping. Il backjumping è, infatti, in grado di saltare ad una variabile precedente solamente a partire da nodi foglia morti, ossia nei quali tutti i valori provati sono inconsistenti mentre il conflict-directed backjumping permette di saltare indietro anche a partire da nodi interni.
Si ipotizzi che le variabili siano inizializzate nell'ordine WA-NSW-T-NT-Q-V-SA e che sia stato assegnato momentaneamente alle variabili i seguenti valori:
WA=rosso, NSW=rosso, T=rosso



.

Fig. 3.4.3 Map coloring problem con conflict-directed backjumping (elaborazione personale)

NT non può essere rosso perché WA è rosso. Se NT è blu Q non può essere né blu né rosso poiché NT è blu e NSW è rosso. Se Q è verde SA non può essere né rosso, né verde, né blu e quindi, con un semplice backtracking Q non può essere verde e NT non può essere blu. Analogamente NT non può essere verde.
Quindi con la combinazione WA=rosso e NSW=rosso non esiste nessuna combinazione possibile per NT-Q-V-SA
NT quindi, se si utilizzasse il backtraking o il backjumping classico, andrebbe a T che però non influenza il fallimento di NT, Q, V e SA e quindi risulterebbero un fallimento tutti i rami sia con T=blu sia con T=rosso. Il backjumping come

precedentemente definito non funzionerebbe poiché NT non è un nodo foglia morto ma è un nodo interno che ha valori consistenti con la precedente assegnazione parziale ma le quattro variabili NT, Q, V e SA prese insieme non presentano valori consistenti. Il conflict set di NT come precedentemete definito non è completo poiché WA, NSW e T non causano il fallimento di NT. Il conflict-directed backjumping risolve il problema in questo modo: una volta che un ramo fallisce perché il dominio della variabile diventa vuoto, ossia si arriva ad un nodo foglia morto, esso passa il suo conflict set alla variabile a cui salta indietro. Tale conflict set viene assorbito dalla variabile a cui si è tornati, togliendo sé stesso dal conflict set. Nell'esempio il conflict set di SA, nodo foglia morto, è {WA, NT, Q}. Essendo fallito si torna a Q e quindi il conflict set di Q che è {NT, NSW} assorbe quello di SA togliendo però sé stesso dall'insieme; il conflict set di Q diventa {WA, NT, NSW}. La ricerca in Q fallisce e si torna quindi alla variabile più recente nel conflict set di Q che è NT. Fino ad ora non si è fatto passi in meno rispetto al backjumping o backtracking ma a differenza del backjumping NT ha un conflict set completo che deriva da Q, il quale è {WA, NSW}. Ora essendo il dominio di NT diventato vuoto, l'algoritmo torna a NSW e non a T come farebbero backjumping e backtracking, risparmiandosi andare a cercare soluzioni in rami già destinati a fallire essendo T non influente sul fallimento delle variabili NT, Q, V e SA (Russel e Norvig 2009).

3.5 Ordinamento delle variabili e dei valori

Ordinare le variabili permette di rendere più efficiente la ricerca. L'ordinamento viene fatto attraverso dei metodi euristici. Uno di questi è l'euristico MRV o minimum remaining values, in cui viene scelta la prossima variabile da assegnare in modo crescente di valori presenti nel dominio. L'assegnazione è quindi dinamica e non statica, ossia con un ordine prefissato. Esso è anche detto most constrained variable perché viene scelta la variabile che è più probabile che causi il fallimento della ricerca presto, dovendo tornare indietro nell'albero. L'euristico

MRV non permette di scegliere quando i domini hanno lo stesso numero di valori. Per questo è possibile utilizzare il degree heuristic che sceglie per prima la variabile che presenta il più alto numero di vincoli con le altre variabili non assegnate. Nell'esempio 3.4.1 SA ha grado 5 e viene scelta per prima.
Una volta scelta la variabile si deve decidere l'ordine di esplorazione dei valori. Un euristico per scegliere tale ordine è il least-constraining-value che preferisce i valori che escludono il minor numero di valori per le variabili vicine nel grafo. Ordinando le variabili si è visto che le performance del backtracking migliorano da 3 a 3000 volte, a seconda del problema. (Russel e Norvig 2009)

3.6 Inserimento di una funzione obiettivo

Con gli algoritmi precedentemente esposti è possibili risolvere qualsiasi CSP, con un costo in tempo di elaborazione che dipende dalla complessità del problema. L'esplorazione dello spazio di ricerca può essere fatto tramite generate and test, per problemi semplici, o tramite backtracking. Le performance del backtracking possono essere migliorate tramite l'ordinamento delle variabili e dei valori che riduce il branching factor, ossia il numero medio di successori di un nodo in un albero, e quindi la complessità del problema spostando la soluzione nella parte a sinistra dell'albero, velocizzando il backtracking. Il backtracking può essere ancora migliorato tramite due tipi di metodi: i metodi look back e i metodi look ahead. I metodi look back, come i backtracking intelligenti, prevedono di tornare indietro in un punto della ricerca evitando che vengano esplorati rami che falliranno sistematicamente per le stesse cause. I metodi look ahead, come le tecniche di propagazione dei vincoli, cercano di prevedere gli effetti del percorrere alcuni rami, in particolari quali termineranno in un fallimento della ricerca. La differenza tra propagazione dei vincoli e backtracking intelligente è che la prima trasforma il CSP in un CSP diverso ma con spazio di ricerca ridotto identificando i punti di fallimento della ricerca, mentre il backtracking intelligente (backjumping e conflict-directed backjumping) non modifica il CSP ma lo esplora in maniera più

accurata, memorizzando informazioni sui fallimenti e utilizzandole per non esplorare punti che falliranno per gli stessi motivi. Applicare ognuna di queste tecniche “all’estremo” porta ad eliminare thrashing ma aumenta il tempo computazionale, per questo le tecniche vengono usate in maniera combinata e/o semplificata per ottimizzare il tempo di ricerca (Kumar 1992).
Le tecniche esposte mostrano come risolvere un CSP quando non è presente una funzione da ottimizzare. Nel caso sia presente una funzione obiettivo da ottimizzare il problema prende il nome di contraint optimization problem, o COP. Un COP è descrivibile quindi come un CSP in cui lo scopo non è solo trovare una soluzione ma trovare quella migliore secondo il criterio di valutazione racchiuso nella funzione obiettivo. Un qualsiasi algoritmo di CSP può essere utilizzato per risolvere un COP, infatti, una volta descritto il problema, è sufficiente aggiungere una variabile che rappresenta la funzione obiettivo e ogni volta che si trova una soluzione al CSP viene aggiunto un nuovo vincolo che garantisce che la soluzione trovata sarà migliore della soluzione trovata precedentemente. Quando non si troveranno più soluzioni l’ultima soluzione trovata sarà quella ottima.

Capitolo 4 – Schedulazione della produzione: MBM management System e applicazione al caso Danieli

Nel seguente capitolo viene presentata un esempio applicativo della programmazione a vincoli nella schedulazione della produzione. Viene presentata l'azienda MBM che produce software gestionali per aziende industriali tra cui ESS (Enterprise Scheduling System), un software per la schedulazione della produzione che utilizza il motore di ottimizzazione IBM ILOG CP Optimizer, basato sulle tecniche di programmazione a vincoli e viene mostrata un'applicazione di tale software al caso Danieli, multinazionale che produce macchinari per la lavorazione dell'acciaio.

4.1 MBM management systems

MBM nasce nel 1980 con l'obiettivo di offrire assistenza tecnica ed applicativa nella progettazione e nello sviluppo di sistemi informativi per aziende industriali. MBM realizza applicazioni software per la gestione aziendale, in particolare sistemi informativi per il controllo dei flussi logistici aziendali sia interni che esterni. MBM è leader nel settore e le sue soluzioni sono state inserite da IBM tra le referenze Smarter Planet. Smarter Planet è un'iniziativa promossa da IBM a partire dal 2008 per promuovere la crescita di un mondo più intelligente; essa si pone di mettere in luce gli strumenti informatici più innovativi ed evoluti, che attraverso la loro eccellenza sono in grado di ridefinire i sistemi, i processi e le infrastrutture tecnologiche del pianeta.

In tale ambito, MBM è stata premiata per le sue soluzioni relative alla pianificazione della produzione, ed in particolar modo per i moduli che incorporano i motori di ottimizzazione matematica IBM ILOG. Con queste applicazioni, utilizzabili nei più svariati ambiti aziendali, le imprese possono aumentare l'efficienza attraverso la ricerca della miglior soluzione possibile.

La realizzazione di un progetto di un sistema informativo avviene nelle seguenti fasi:

- studio di fattibilità per individuare le problematiche e definire un piano d'azione
- definizione economica preventiva
- stima dei tempi di realizzazione
- acquisizione del cliente del know-how applicativo per la gestione autonoma.

Un progetto di sistema informativo deve essere in grado di integrarsi con moduli già esistenti in azienda e di adattarsi alle specifiche esigenze del cliente. I principali prodotti offerti da MBM sono APACHE V4, ERP per aziende di produzione, SCM, supply chain management system, PCM 64b, controllo costi e analisi varianze, RETRACKER, sistema per la gestione dei flussi logistici inbound e outbound e GPS 64b, sistema di pianificazione della produzione.
In particolare GPS 64b è il sistema di pianificazione dei materiali e delle risorse a capacità finita che utilizza, per alcuni moduli, il motore di ottimizzazione IBM ILOG.
In figura 4.1.1 viene presentato lo schema dell'architettura applicativa di GPS. Il software estrae direttamente i dati dall'ERP aziendale per l'elaborazione, per poi presentarli all'utente tramite browser. Una volta estratti i dati dall'ERP è possibile eseguire simulazioni what if con ripianificazione selettiva.
GPS è costituito dai seguenti moduli:

- DPF: Demand Planning & Forecast
- MPS: Master Production Schedule
- MRP: Material Requirements Planning
- PRM: Pegging Resolution Management
- CRP: Capacity Requirement Planning
- RCP: Raw Capacity Planning
- FCP: Finite Capacity Planning
- ESS: Enterprise Scheduling System

DPF si occupa della previsione della domanda.

MPS, MRP e PRM si occupano della pianificazione dei materiali.

CRP, RCP, FCP ed ESS si occupano invece dell'analisi delle capacità produttive.

Alcuni moduli software fanno utilizzo dei motori di ottimizzazione IBM ILOG, in particolare:

- CPLEX per la risoluzione di problemi di ottimizzazione lineare
- CP Optimizer per i problemi di ottimizzazione risolti con la tecnica della programmazione a vincoli, utilizzato in particolare nel modulo ESS per la schedulazione della produzione.

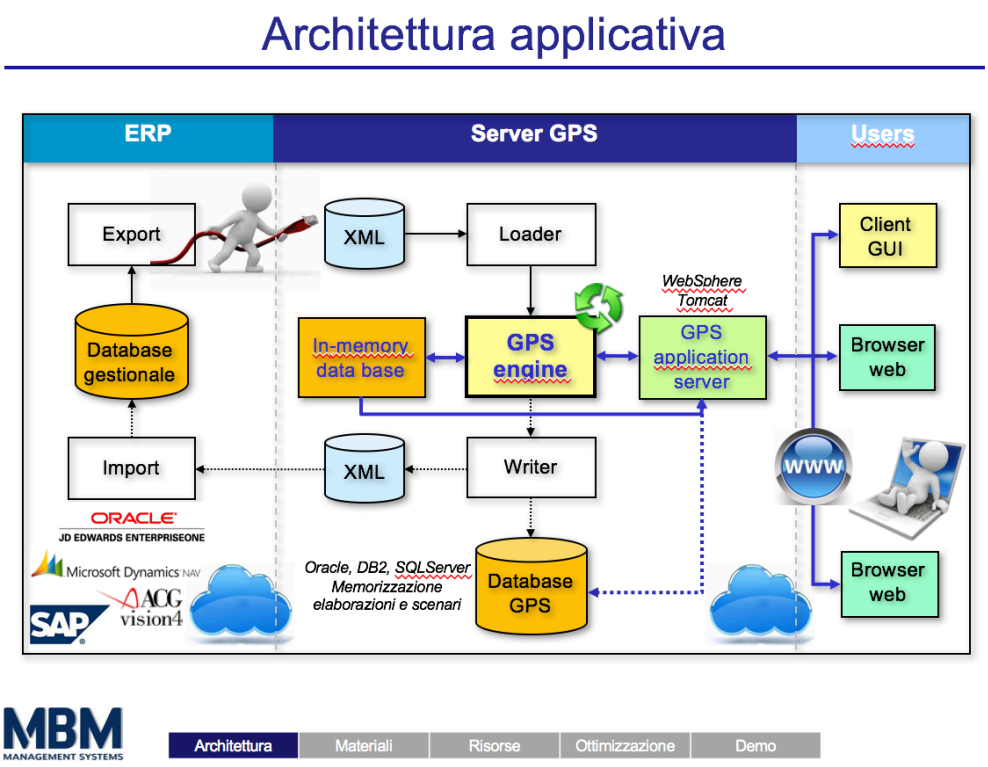


Fig. 4.1.1 Architettura applicativa GPS 64b (MBM Italia S.r.l s.d.)

Sia nel caso dell'utilizzo di CPLEX che di CP Optimizer la tecnica risolutiva si basa sulla modellazione del problema con varabili, vincoli e funzione obiettivo. I vantaggi dell'utilizzo dell'ottimizzazione in questo tipo di problemi sono la qualità

del risultato, la possibilità, attraverso la modellazione, di descrivere scenari anche molto complessi e la facilità nella personalizzazione dei modelli attraverso il macro-linguaggio di modellazione ILOG.

4.2 Il modulo ESS

ESS, o enterprise scheduling system, è il modulo di GPS 64b per la schedulazione della produzione, che, a differenza di altri software disponibili sul mercato, non utilizza tecniche algoritmiche o euristiche, ma arriva alla soluzione sfruttando le potenzialità del motore di ottimizzazione CP Optimizer di IBM ILOG.
IBM ILOG CP Optimizer è un software che fornisce un motore di ottimizzazione basato sulla programmazione a vincoli finalizzato alla risoluzione di CSP e COP (contraint satisfaction problem e contraint optimization problem). Con questo motore è possibile modellare contesti operativi complessi (esempio centri FMS, impianti di verniciatura, forni per trattamenti termici) che difficilmente possono essere schedulati con metodi tradizionali. Per trovare la soluzione ottima il motore di ottimizzazione analizza tutti i possibili scenari di combinazioni, fino a determinare la migliore soluzione. CP Optimizer utilizza le tecniche di programmazione a vincoli esposte nel capitolo 3, come il backtracking e la propagazione dinamica dei vincoli.
La dinamica elaborativa di CP Optimizer può essere schematizzata nei seguenti punti:

1- Una volta definito il modello matematico (costruito dinamicamente dal motore di pianificazione in base ai dati in input forniti), CP Optimizer assegna un valore arbitrario, ma comunque all'interno del proprio dominio di esistenza, ad una delle n variabili (ad esempio alla variabile 1): così facendo il dominio di esistenza di tutte le altre variabili viene ridotto (propagazione dei vincoli)
2- Se lo scenario creatosi a seguito di questa decisione prevede ancora una soluzione ammissibile, il motore prosegue l'elaborazione attribuendo un

valore arbitrario ad un'altra variabile (ad esempio alla variabile 2): in caso contrario, l'azione effettuata viene annullata e viene dato un nuovo valore alla variabile 1 (back-tracking). La selezione delle variabili da valorizzare e dei valori da assegnare è pilotata da precisi criteri, volti a rendere l'esplorazione più efficiente e minimizzare i tempi di elaborazione (capitolo 3). L'esplorazione di alcuni rami viene interrotta quando si rivela non migliorativa della soluzione sub-ottima già raggiunta

3- La procedura itera fino a quando a tutte le variabili del modello viene attribuito un valore ammissibile: in questo modo il sistema giunge ad una prima soluzione del problema
4- CP Optimizer comunica il risultato raggiunto (in azzurro nel log in figura 4.2.2) e ripete la dinamica appena descritta nella ricerca di nuove soluzioni, creando scenari alternativi
5- Se il motore trova una soluzione ammissibile migliore della precedente la segnala, altrimenti non ne dà evidenza.

Esempi di come opera CP Optimizer sono riportati nel Capitolo 3 e nell'appendice A con il problema delle 6 regine.

In contesti complessi, dove il numero di vincoli e di variabili sono elevati, i tempi di elaborazione possono risultare molto lunghi; infatti, la complessità dei problemi di scheduling è spesso esponenziale, ciò significa che il tempo di elaborazione per trovare una soluzione ottima aumenta in maniera esponenziale all'aumentare del numero di variabili e di vincoli.

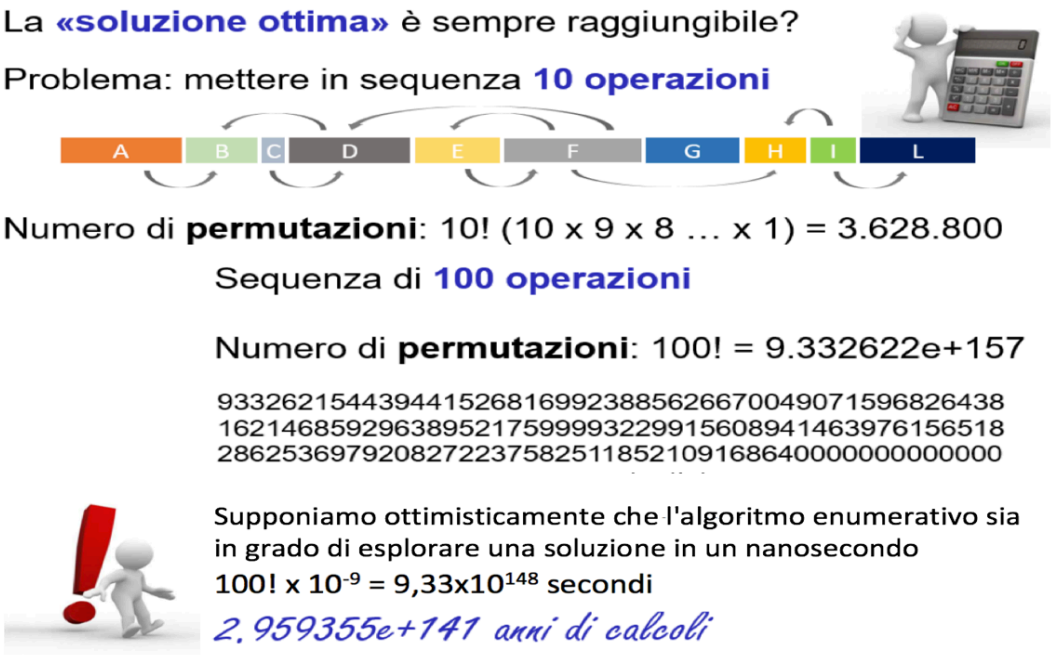


Fig. 4.2.1 Complessità di calcolo della soluzione ottima (elaborazione personale)

Lunghi tempi di elaborazione sono spesso incompatibili con i contesti produttivi, che richiedono una soluzione migliore possibile in tempi sufficientemente ridotti. Accade inoltre che, superato un certo lasso di tempo, i benefici derivanti dal miglioramento continuo del risultato non compensino l'attesa richiesta per raggiungere l'ottimo. In altre parole dopo un certo intervallo temporale si avrà un miglioramento della funzione obiettivo minimo a fronte di un intervallo temporale necessario per trovare tale soluzione elevato. L'andamento asintotico delle soluzioni è mostrato nella figura seguente.

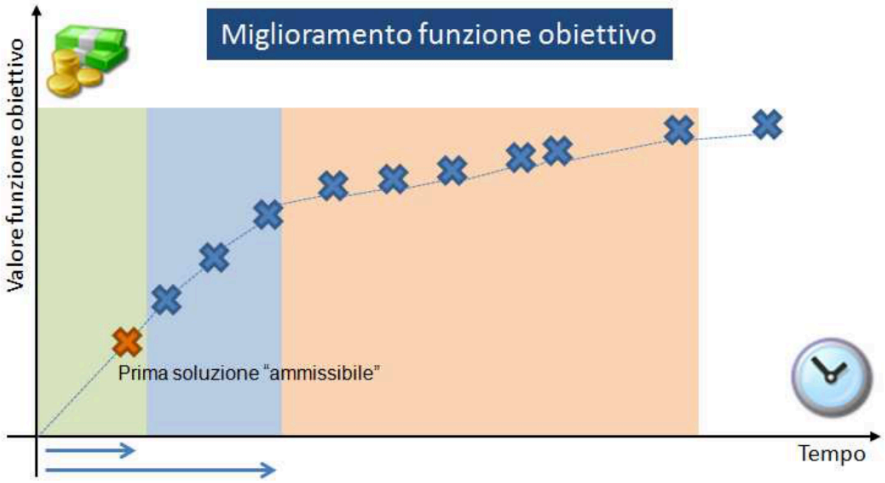


Fig. 4.2.2 Andamento asintotico delle soluzioni trovate

Per tutte queste ragioni è spesso conveniente accontentarsi di una soluzione "sub-ottima", ossia la soluzione migliore che il motore riesce a trovare in quel determinato intervallo di tempo che gli è stato fornito. A differenza dell'ottimizzazione matematica e quindi possibile interrompere l'elaborazione e utilizzare la soluzione ammissibile sub ottima trovata in quel tempo.
La figura 4.2.3 fornisce un esempio di elaborazione di CP Optimizer a cui si è dato un tempo massimo di elaborazione di 120 secondi su un problema caratterizzato da 6006 variabili e 25881 vincoli per cui sono state trovare 3 soluzioni ammissibili tra cui viene data la migliore nel tempo fornito.

```
! ------------------------------------------------------------------------
! Minimization problem - 6006 variables, 25881 constraints
! Presolve       : 3321 extractables eliminated
! TimeLimit            = 120
! Initial process time : 0.71s (0.60s extraction + 0.10s propagation)
!  . Log search space  : 97648.5 (before), 97648.5 (after)
!  . Memory usage      : 37.8 MB (before), 43.7 MB (after)
! Using sequential search.
! ------------------------------------------------------------------------
!           Best Branches  Non-fixed             Branch decision
                     1000       4618             on 030043700A_120_0010  _setUp_1
                     2000       4696        F    on 0300617_948_0010   _setUp_1
                     3000       3865             on 0300617_24897_0010 0 _setUp_1
                     4000       2000             on 0300617_24926_ZZZZ  _oop
                     5000         15             on 0300617_8823_0010  _oop
*  255357431472      5357 53.44s                      -
   255357431472      6000       3824             on _itv3673
   255357431472      7000       1850             on 0300617_24857_ZZZZ  _oop
   255357431472      8000          2        F    on 0300617_24826_ZZZZ  _oop
*  241814052492      8267 64.21s                      -
   241814052492      9000       3211             on 0300617_24914_0010 1 _setUp_0
   241814052492     10000       1653             on 0300617_11071_0010  _setUp_3
   241814052492     11000       1653             on 0300617_24816_0010 1 _oop
   241814052492     12000       2398             on 0300617_24893_0010 0 _setUp_1
   241814052492     13000        143             on 0300617_20955_0010  _setUp_2
*  233127818946     13328 87.70s                      -
   233127818946     14000       4987        F    on 0300617_24849_0010 1 _setUp_1
   233127818946     15000       4408        F    on 0300617_24897_0010 1 _setUp_5
   233127818946     16000       4624             on 0300617_24897_0010_1_span
   233127818946     17000       2972             on 0300617_24889_0010 1 _setUp_4
! Time = 118.49s, Average fail depth = 708, Memory usage = 161.0 MB
!           Best Branches  Non-fixed             Branch decision
! ------------------------------------------------------------------------
! Search terminated by limit, 3 solutions found.
! Best objective         : 233127818946
! Number of branches     : 17933
! Total memory usage     : 168.3 MB (159.4 MB CP Optimizer + 8.9 MB Concert)
! Time spent in solve    : 120.26s (119.65s engine + 0.60s extraction)
! Search speed (br. / s) : 149.9
! ------------------------------------------------------------------------
```

Fig. 4.2.3 Elaborazione ILOG CP Optimizer

ESS quindi utilizza CP Optimizer per risolvere il problema di eseguire una schedulazione dei reparti produttivi che ottimizzi l'impiego di risorse e l'utilizzo dei materiali. ESS permette l'allocazione simultanea delle risorse principali, secondarie, manodopera, attrezzature, risorse in serie, ossia che possono essere assegnate a un'unica operazione, e in parallelo, ossia che ammettono più operazioni alla volta. ESS permette inoltre la selezione automatica dei sottocicli alternativi (ad esempio scelta fra due macchinari alternativi ma con prestazioni

diverse), il rispetto dei vincoli di materiali stabiliti da MRP/PRM, la possibilità di personalizzazione del modello (che va adattato al processo produttivo che si intende ottimizzare) e la possibilità di gestire i calendari delle risorse.
La procedura di schedulazione, descritta in figura 4.2.4, si basa sulle seguenti fasi:

- Descrizione del problema
- Creazione dinamica del modello, ossia il modello viene creato a partire dai dati in input dall'ERP e dalle modifiche apportate dal pianificatore, il quale può agire tramite interfaccia per modificare vincoli e parametri
- Risoluzione del problema da parte di ILOG CP Optimizer
- Analisi della soluzione e possibilità di simulazione what if.

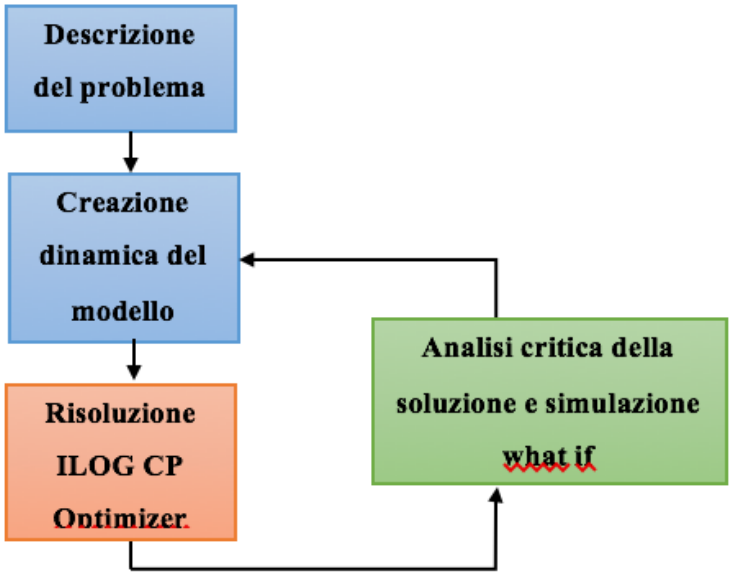


Fig. 4.2.4 ESS schema applicativo (elaborazione personale)

Molto importante è l'analisi what-if che consente di simulare scenari alternativi e di arrivare a una soluzione ottimale che meglio rispecchia le esigenze aziendali con procedimento interattivo. Il software è integrato con l'ERP ed è in grado di generare un piano di lavoro globale, ossia coordinato per tutti i reparti produttivi e non a compartimenti stagni. L'intervento umano diventa quindi di primaria

importanza nella fase di analisi critica dell'elaborazione attuata dal motore, per rilevare situazioni da ritarare manualmente, esempio rotture utensili, macchinari o ritardi fornitori.

4.3 Caso Studio Danieli & C. Officine Meccaniche.

Scenario e introduzione al problema

Il gruppo Danieli è una multinazionale italiana con sede a Buttrio (Udine) leader mondiale nella produzione di differenti tipologie di impianti siderurgici. Essa ha filiali operative in Germania, Austria, Svezia, Cina, Stati Uniti, Russia, India e Thailandia, con oltre 10.000 dipendenti.

Per mantenere la propria leadership Danieli punta non solo su qualità del prodotto e customer satisfaction ma anche su ricerca e sviluppo per la quale investe circa 140 milioni euro l'anno.

Danieli fornisce impianti progettati e forniti chiavi in mano, realizzando la quasi totalità dei macchinari nei propri stabilimenti (come laminatoi, forni, impianti per la produzione di ghisa e acciaio ecc.). Dovendosi far carico non solo della progettazione ma anche della manifattura, allestimento e consegna dei macchinari, è necessario attuare una programmazione della produzione che risulta complessa. Per questo il gruppo Danieli ha deciso di affidarsi e collaborare ormai da molti anni con MBM Italia con l'obiettivo di adottare nuove metodologie di pianificazione e schedulazione della produzione basate sui motori di ottimizzazione matematica, sostituendo il vecchio sistema che non era in grado di gestire le complessità dell'azienda.

Ogni commessa in arrivo a Danieli viene valutata e affidata a uno o più degli stabilimenti in base a politiche aziendali, saturazione delle fabbriche e alle tipologie di impianti da realizzare. Ogni anno si contano tra i 40.000 e i 50.000 nuovi codici articolo e ciò mostra come i prodotti sviluppati da Danieli non siano realizzati in serie ma siano spesso fabbricati su progetto. Ogni commessa ha caratteristiche uniche come urgenza, costi, tempi di consegna ecc.

Una volta incaricato uno stabilimento, esso ne programma la realizzazione in base alle commesse già presenti, alle risorse disponibili, attuando la pianificazione a capacità finita su un arco temporale mediamente di 2 anni, dall'ingegnerizzazione alla consegna chiavi in mano. Sulla base di tale piano lo stabilimento procede alla schedulazione su un orizzonte temporale più limitato. Tale schedulazione copre circa 45 giorni ed ha l'obiettivo di individuare la sequenza delle singole lavorazioni in grado di ottimizzare tempi di set up, ritardi di consegna, o altri obiettivi aziendali rilevanti; in particolare l'obiettivo ricercato da Danieli nel progetto del nuovo sistema di schedulazione era quello di ridurre i ritardi di consegna.

La complessità della schedulazione di uno stabilimento Danieli è dovuta a quattro fattori: (i dati a titolo di esempio sono relativi al sito produttivo di Buttrio)

- orizzonte temporale (copertura di 45 giorni)
- pianificazione di circa 10.000 operazioni
- presenza di oltre 90.000 vincoli
- presenza di oltre 32.000 variabili.

Questi quattro fattori contribuiscono quindi a rendere il modello matematico complesso e time-consuming dal punto di vista dell'elaborazione.

La schedulazione, inoltre, deve essere in grado di gestire i calendari della macchina (disponibilità della macchina), i turni degli addetti in base alle competenze professionali, la quantità di materiali necessaria per le lavorazioni (per evitare eccedenze o fermi macchina per mancanze) e i trasferimenti dei materiali.

La schedulazione deve inoltre operare su un arco temporale mobile, ossia slittare in avanti giorno dopo giorno senza dover aspettare il termine di un piano per pianificare il giorno successivo dell'orizzonte temporale; inoltre deve essere possibile variare la programmazione in tempo reale in modo da garantire flessibilità. (Russo 2015).

4.4 Caso Studio Danieli & C. Officine Meccaniche.

Le fasi del processo produttivo

Ogni commessa in Danieli ha caratteristiche uniche e la dimensione dei beni prodotti è spesso rilevante (con pezzi che possono arrivare alla lunghezza di 20 metri) e dunque anche l'area adibita alle specifiche lavorazioni è estremamente ampia.

Il processo produttivo di Danieli è costituito da quattro reparti:

- trattamenti termici
- lavorazioni meccaniche
- carpenteria
- montaggio.

Il primo reparto è dunque costituita dai trattamenti termici, dove i componenti che andranno a costituire le macchine subiscono processi di riscaldamento e raffreddamento per modificarne le caratteristiche fisiche.

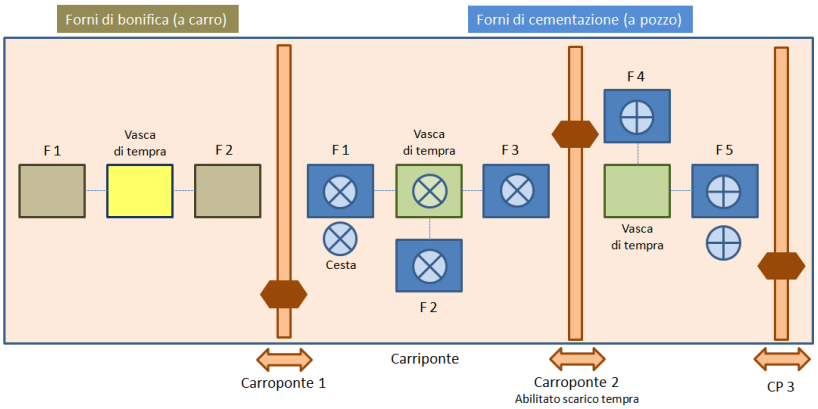


Fig. 4.4.1 Schema semplificativo del reparto trattamenti termici

I componenti vengono trasportati all'interno di ceste sollevate e movimentate da carriponte, i quali permettono di posizionare le ceste all'interno dei forni per il riscaldamento, per poi essere tolte dal forno e immerse nelle vasche di tempra.

I prodotti che escono dal reparto trattamenti termici vengono poi lavorati nel reparto lavorazioni meccaniche. Parallelamente al reparto lavorazioni meccaniche opera la carpenteria, che si occupa di lavorare prodotti siderurgici come lamiera, quadri, tondi o tubi per realizzare componenti che andranno poi a costituire i macchinari. I macchinari presenti sono soprattutto taglio laser, piegatura, saldatura e verniciatura.
Il reparto lavorazioni meccaniche contiene circa 100-150 macchine. Le macchine più complesse presenti nel reparto sono un centro di lavoro FPT e un centro di lavoro Burkhardt Weber, i quali sono dei centri FMS, alesatrici Pama e Waldrich, torni verticali e paralleli.
Infine l'ultimo reparto del processo produttivo di Danieli è costituito dal montaggio, che si occupa di montare i moduli che verranno poi spediti nelle diverse parti del mondo dove verrà poi fatto in loco l'assemblaggio finale e il collaudo.
Il software di schedulazione ESS attualmente gestisce solo il reparto lavorazioni meccaniche, mentre è in corso di implementazione per i reparti carpenteria e montaggio.
Tale reparto è classificabile come Open shop (paragrafo 1.3) in cui le operazioni da eseguire non hanno un percorso determinato da seguire. Il motivo per cui si è deciso di partire con le lavorazioni meccaniche è che esso rappresenta il reparto più critico, in particolare per la presenza dei due centri FMS Burkhardt Weber e FPT.
Un centro FMS, o flexible manufacturing system, è un sistema di produzione flessibile costituito da un gruppo di stazioni di lavoro collegate tra di loro da un sistema di trasporto automatico che agiscono controllate da un sistema computerizzato di controllo. Ogni stazione deve essere dotata di un sistema automatico di carico e scarico, di cambio automatico dell'utensile e di un magazzino a bordo macchina di utensili. Il sistema deve poter accettare in ingresso pezzi differenti con arrivi casuali e deve essere in grado di ottimizzare il flusso dei materiali al suo interno. Un centro di lavorazione di questo tipo è quindi in grado

di effettuare lavorazioni diverse, con tempi di set up ridottissimi. Questa capacità di riconfigurazione è ciò che rende il sistema "flessibile" pur conservando i vantaggi della produzione di serie.

I vantaggi dei centri FMS sono:

- capacità di lavorare un elevato numero di articoli
- elevata saturazione: essendo una macchina flessibile, può essere impiegata per molte operazioni diverse e ciò porta ad una elevata utilizzazione
- minor spazio occupato rispetto a macchine dedicate
- minor costi di manodopera
- lead time inferiori: essendo flessibili, questi sistemi possono infatti riconfigurarsi velocemente
- ridondanza: qualora una macchina sia temporaneamente fuori uso, la versatilità delle macchine stesse fa sì che le stesse operazioni possano, almeno in gran parte, essere ridistribuite tra le altre macchine
- modularità: è possibile introdurre le macchine in tempi successivi
- maggiore qualità del prodotto.

Il motivo per cui schedulare i centri FMS rappresentano una criticità per questo reparto è dovuto al fatto che schedulare gruppi di macchine singole risulta meno complesso che schedulare un centro FMS a causa di:

- necessità di coordinamento fra le macchine
- disponibilità dei pallet porta pezzo
- disponibilità e coordinamento degli operatori per il carico/scarico
- gestione delle attrezzature

Schedulare un centro FMS significa determinare l'ordine con cui è più conveniente introdurre gli ordini di produzione nel sistema e l'ordine con cui lavorare i pezzi su ciascuna macchina. Per la presenza di numerosi vincoli e variabili tale problema di schedulazione può essere affrontato efficacemente solo attraverso la modellazione.

Un centro FMS ai fini della modellazione è composto dai seguenti elementi principali:

1- una o più macchine a controllo numerico
2- un certo numero di pallet o tavole, su cui vengono fissati i pezzi per la movimentazione e la lavorazione
3- delle attrezzature per fissare i pezzi sul pallet in modo da presentare i pezzi alle macchine in modo preciso con le tolleranze previste
4- una o più baie di carico/scarico
5- uno o più operatori addetti al carico/scarico dei pallet

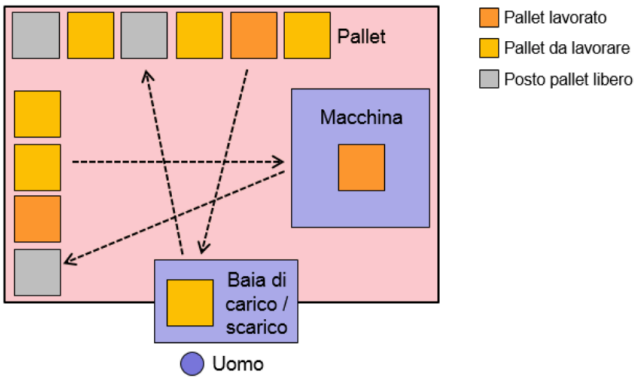


Fig. 4.4.2 Centro FMS (MBM Italia S.r.l s.d.)

Macchina, pallet, attrezzature e manodopera sono tutte risorse a capacità finita, ognuna con un proprio calendario di disponibilità: tipicamente macchina e attrezzature sono disponibili h24, mentre la disponibilità della manodopera è regolata da turni di lavoro. La macchina è una risorsa di tipo seriale, in grado cioè di lavorare un pezzo alla volta: quando il pallet entra in macchina, questa lavora uno dopo l'altro tutti i pezzi montati sul pallet.
I vincoli presenti sono quindi vincoli di capacità di macchine, disponibilità di pallet, attrezzature e utensili, vincoli di precedenza delle operazioni e vincoli di disponibilità dei materiali e componenti.

Un ordine di produzione su un centro FMS può essere di quantità superiore al numero di pezzi collocabili sul singolo pallet: lo schedulatore deve frazionare automaticamente la lavorazione in più "split", in base al numero di attrezzature disponibili e alla capienza dei pallet.

Ogni split di operazione è composto da tre azioni:

1- assemblaggio o carico dei pezzi da parte dell'operatore sul pallet
2- lavorazione effettuata dalla macchina FMS
3- di assemblaggio o scarico dei pezzi da parte dell'operatore

Se, per esempio, l'ordine di produzione è di 10 pezzi e ogni pallet può ospitare un solo pezzo per volta, l'operazione viene automaticamente suddivisa in 10 split, ognuno dei quali viene schedulato autonomamente: normalmente i vari split di un'operazione verranno schedulati sulla macchina alternati a split di altre operazioni, in modo da massimizzare l'utilizzo della macchina. Operare non su split ma sulla quantità complessiva dell'operazione non rappresenta quindi in modo reale la dinamica operativa del centro.

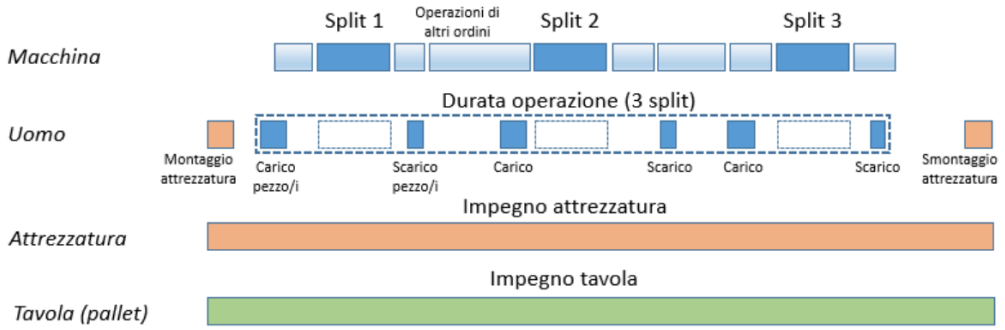


Fig. 4.4.3 Schedulazione centro FMS (MBM Italia S.r.l s.d.)

4.5 Caso Studio Danieli & C. Officine Meccaniche.

La soluzione adottata e il modello matematico

Come esposto nel paragrafo 4.3, il modello matematico della schedulazione della produzione di uno stabilimento Danieli risulta complesso.
La soluzione adottata da Danieli utilizza il software ESS di MBM basato sulla programmazione a vincoli esposto nel paragrafo 4.2.
Il modello prevede diversi tipi di vincoli, sia generali che specifici dell'ambiente Danieli tra cui:

- Vincoli di pegging o di distinta base: i vincoli di pegging stabiliscono i legami fra le divere operazioni, ad esempio l'impossibilità di eseguire una certa operazione che richiede un componente se non si è fatto o ricevuto prima il componente stesso;
- Vincoli di precedenza tra operazioni;
- Vincoli di simultaneità o vincoli esterni: implica l'esecuzione di ordini diversi che devono arrivare insieme alla lavorazione successiva;
- Vincoli di calendario delle risorse.

Tali vincoli vengono espressi non in forma matematica ma attraverso il macrolinguaggio di programmazione ILOG. Alcuni esempi di tale linguaggio sono rappresentati nella figura 4.5.1.

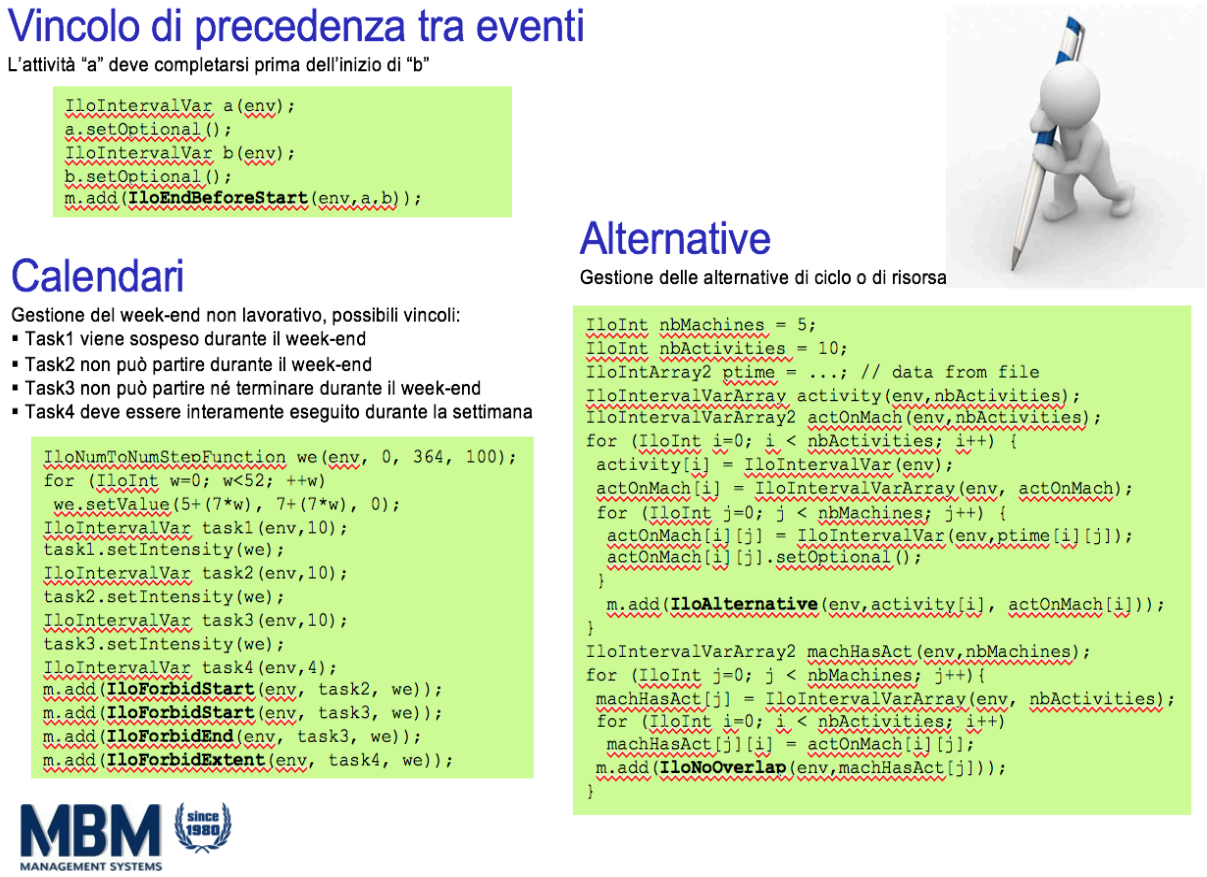


Fig. 4.5.1 Esempi di modellazione

Ogni operazione prevede inoltre tempi di lavorazione, di coda, di attesa, di trasporto e di set up. Ogni risorsa può avere un'efficienza che influenza il tempo di esecuzione delle lavorazioni.

La criticità principale del modello e del sistema consiste nei tempi di elaborazione. Infatti un modello cosi complesso passato direttamente al motore CP Optimizer impiegherebbe tempi troppo elevati rispetto al contesto operativo di Danieli. Per questo MBM ha deciso di utilizzare un approccio di elaborazione a stadi per fornire una soluzione quanto più vicina all'ottimo, sfruttando la possibilità di usufruire della funzionalità di starting point in CP Optimizer. Tale funzionalità in particolare permette a CP Optimizer in alcuni casi di ottenere una soluzione

migliore e più velocemente se viene fornito all'ottimizzatore uno starting point, ossia una istanza della soluzione del problema. In questo modo il motore non deve specificare ogni singolo valore per ogni variabile decisionale. Le informazioni contenute nello starting point forniscono una guida per la ricerca della soluzione, sebbene non ci sia garanzia che la soluzione ottima sia vicina ad esso (IBM s.d.).
La procedura di elaborazione a stadi prevede che nel weekend, periodo con meno impegni, venga fatta una elaborazione più lunga che si interrompe al raggiungimento di una soluzione sub ottima soddisfacente, evitando ulteriori dispendi di energia sproporzionati rispetto ai benefici che si otterrebbero dal miglioramento. Successivamente, partendo da questo starting point del weekend, ogni giorno vengono elaborati i piani operativi con le modifiche causate dai nuovi input (come ad esempio gli avanzamenti di produzione effettuati, le modifiche manuali dei pianificatori, criticità e priorità presenti), evidenziando in tempo reale accavallamenti, incongruità e problemi e determinando l'elemento che li ha generati. In questo modo viene implementata la logica rolling della programmazione, ossia l'avanzare di giorno in giorno della pianificazione.
Il sistema in questo modo fornisce la massima flessibilità e tempestività. Per evitare eventuali problemi di organizzazione interna è previsto inoltre un orizzonte temporale di 3-5 giorni in cui i piani operativi sono congelati, ad esclusione di modifiche rilevanti.
Nel modello e nell'elaborazione la funzione obiettivo riveste un ruolo di particolare importanza. Essa non è solo il criterio secondo il quale una soluzione è preferibile ad un'altra, ma una diversa formulazione di essa può snellire molto la procedura di calcolo del motore.
Si considerino ad esempio 3 ordini A, B e C. Sia R(A) il ritardo dell'ordine A e PR_A la priorità dell'ordine A. La funzione obiettivo più semplice possibile che esprime la somma dei ritardi pesati è la seguente:

$$f(x) = [PR_A R(A) + PR_B R(B) + PR_C R(C)].$$

Per semplicità, e perché in Danieli l'interesse è posto particolarmente sulla riduzione dei ritardi di consegna, viene considerato solo il ritardo totale pesato;

tuttavia niente impedisce di inserire nella funzione obiettivo ulteriori fattori come la somma dei tempi di set up o fattori di costo. In questo caso si stabilisce una funzione obiettivo in questo modo:

$$f(z) = w_x f(x) + w_y f(y)$$

w_x e w_y sono coefficienti che permettono di rendere le funzioni dello stesso ordine di grandezza. È possibili utilizzare anche una funzione composta del tipo

$$f(z) = f(x)(1 + f(y))$$

quando vi è un fattore principale (esempio ritardi di consegna) da minimizzare ma contemporaneamente si vuole agire su un secondo fattore (esempio somma dei tempi di set up). Infatti una funzione obiettivo di questo tipo fa in modo che l'ottimizzatore sia portato a minimizzare f(y), ma nel contempo a non aumentare f(x).

La funzione obiettivo fornita è quella più semplice adottabile.

In realtà si utilizza piuttosto non R(A) ma bensì $e^{R(A)}$, nella forma seguente:

$$f(x) = [PR_A e^{R(A)} + PR_B e^{R(B)} + PR_C e^{R(C)}]$$

L'utilizzo dell'esponenziale del ritardo e non del ritardo è motivato dal fatto che la funzione e^x decresce più velocemente di x al diminuire del ritardo e questo velocizza il processo di ottimizzazione poiché la soluzione converge prima verso il minimo. I coefficienti di priorità invece vengono lasciati non all'esponente per evitare problemi di overflow.

La metodologia utilizzata in Danieli però, differisce ulteriormente da quella sopra esposta. Si consideri A, B e C tre operazioni, non più ordini.

Per ogni risorsa a capacità finita sia:

R(A): ritardo dell'operazione A

PR_A: priorità dell'operazione A

m_A: la distanza dell'inizio dell'operazione da oggi (makespan)

C(A): contributo di A alla funzione obiettivo

d_W: peso del ritardo dato dalla seguente tabella (esempio):

# giorni di ritardo	peso
5	1.0
20	1.000
30	100.000

In questo modo viene pesato di più il ritardo relativo a giorni più alti, e meno quello relativo a giorni più bassi.

$$C(A) = d_w PR_A m_A e^{R(A)}$$

$$F.O. = f(x) = \sum_{su\ tutte\ le\ risorse} C(A) + C(B) + C(C)$$

e generalizzando a n operazioni e m risorse:

$$F.O. = f(x) = \sum_m \sum_n d_w PR_A m_A e^{R(i)}$$

Il makespan è stato inserito al fine di incentivare l'ottimizzatore a concentrarsi sulle operazioni in ritardo più vicine all'oggi, ossia quelle nel primo periodo di schedulazione (esempio fino ad un mese dall'oggi), dando quindi meno peso ad ordini in ritardo ma che verranno schedulati successivamente.
Per implementare ciò, il makespan viene misurato con unità di misura differenti a seconda del numero di giorni che intercorrono tra l'inizio schedulato dell'operazione e l'oggi di pianificazione. L'unità di misura utilizzata è definita da una tabella come la seguente:

Valore makespan in giorni	Unità di misura usata nella formula
5	Secondi
30	Ore
60	Giorni

In questo modo si cerca di dare più peso agli ordini in ritardo vicini all'oggi di pianificazione poiché il valore numerico di m_A sarà maggiore per operazioni in ritardo vicine (poiché misurato in secondi) e minore per operazioni in ritardo lontane (poiché misurato in giorni). Il motivo per cui si è adottato questa soluzione è perché nel caso in oggetto non si voleva che operazioni con date di consegna lontane nel tempo e quindi gestibili successivamente, andassero a influire troppo sulla funzione obiettivo rispetto a quelle con scadenza vicine all'oggi.
Un ulteriore tecnica, non utilizzata in Danieli ma comunque possibile, è quella detta multicriterion, nella quale viene definita una priorità di obiettivi in cui un obiettivo 1 è prioritario rispetto a un obiettivo 2.
Ob1 = f(x) = priorità 1 (esempio riardo di consegna)
Ob2 = f(y) = priorità 2 (esempio tempi di set up).
Definita questa priorità di funzioni obiettivo si impone che f(y) può migliorare solamente se f(x) non peggiora.

4.6 Caso Studio Danieli & C. Officine Meccaniche. Implementazione, vantaggi e risultati ottenuti

La fase di avviamento del nuovo sistema è cominciata implementando il nuovo sistema solamente per le operazioni relativi al reparto lavorazioni meccaniche, nello stabilimento in Russia, più piccolo di quello italiano da gestire (15 macchine FMS). I risultati positivi raggiunti hanno portato a introdurre la procedura anche nello stabilimento italiano e successivamente anche in Cina, Thailandia e India. Il team di schedulazione ha verificato l'impatto positivo del sistema in termini di tempestività di risposta alle modifiche, affidabilità, la possibilità di adottare simulazioni what if, riduzione dei tempi di set up e dei ritardi di consegna soprattutto nel settore ricambi dove il lead time rappresenta una leva competitiva chiave dell'azienda. I tempi di set up, pur non essendo inseriti nella funzione obiettivo (nel caso specifico di Danieli), si sono ridotti poiché la minimizzazione

dei ritardi di consegna induce il sistema a scegliere sequenze convenienti anche dal punto di vista dei setup.

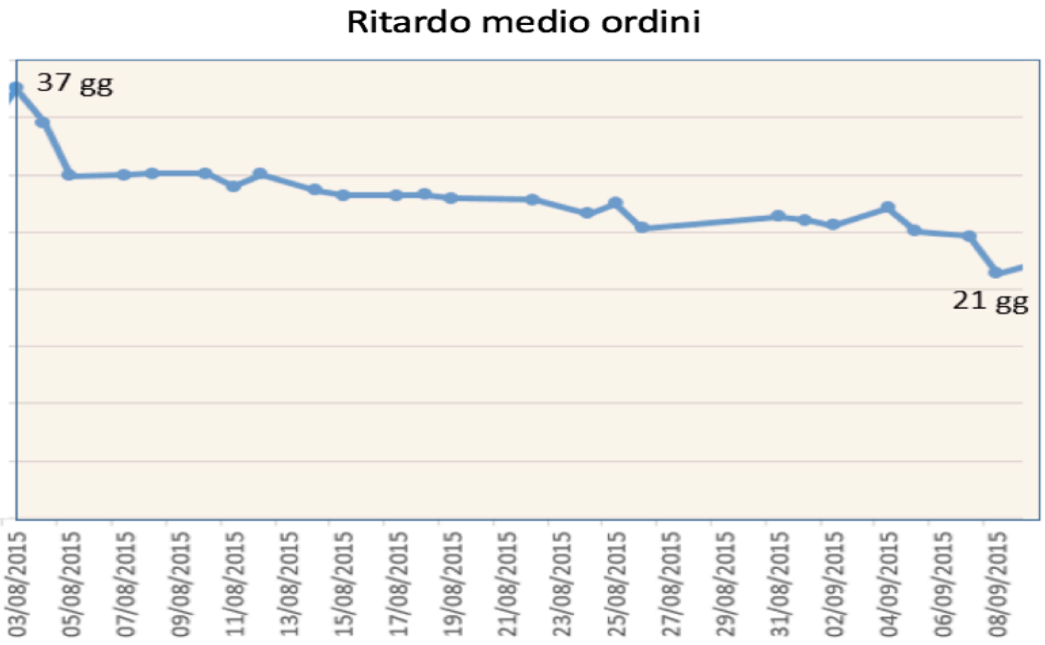


Fig. 4.6.1 Riduzione del ritardo medio degli ordini in Danieli

La figura 4.5.1 mostra come, attraverso il monitoraggio successivo all'implementazione del sistema, il ritardo medio degli ordini si sia ridotto da 37 giorni a 21 giorni (-41%).
Visti i risultati positivi ottenuti dall'implementazione di queste procedure di schedulazione della produzione basate sulla programmazione a vincoli, l'obiettivo è quello di allargare il sistema anche agli altri reparti.

APPENDICE A

Il Problema delle sei regine

Il problema delle 6 regine è uno dei problemi che può essere risolto con la programmazione a vincoli. Il problema consiste nel posizionare, su una scacchiera 6x6, 6 regine in modo da evitare un attacco reciproco.

Il problema può essere formalizzato come segue:

Le 6 regine sono rappresentate dalle variabili X_1, X_2, X_3, X_4, X_5, X_6, il pedice *i* esprime la colonna occupata dalla regina.

Il valore della variabile va da 1 a 6 ed esprime la riga sulla quale si trova la regina.

I vincoli del problema sono:

Dominio delle variabili:

$1 \leq X_i \leq 6$ per $1 \leq i \leq 6$

Due regine non si devono trovare sulla stessa riga

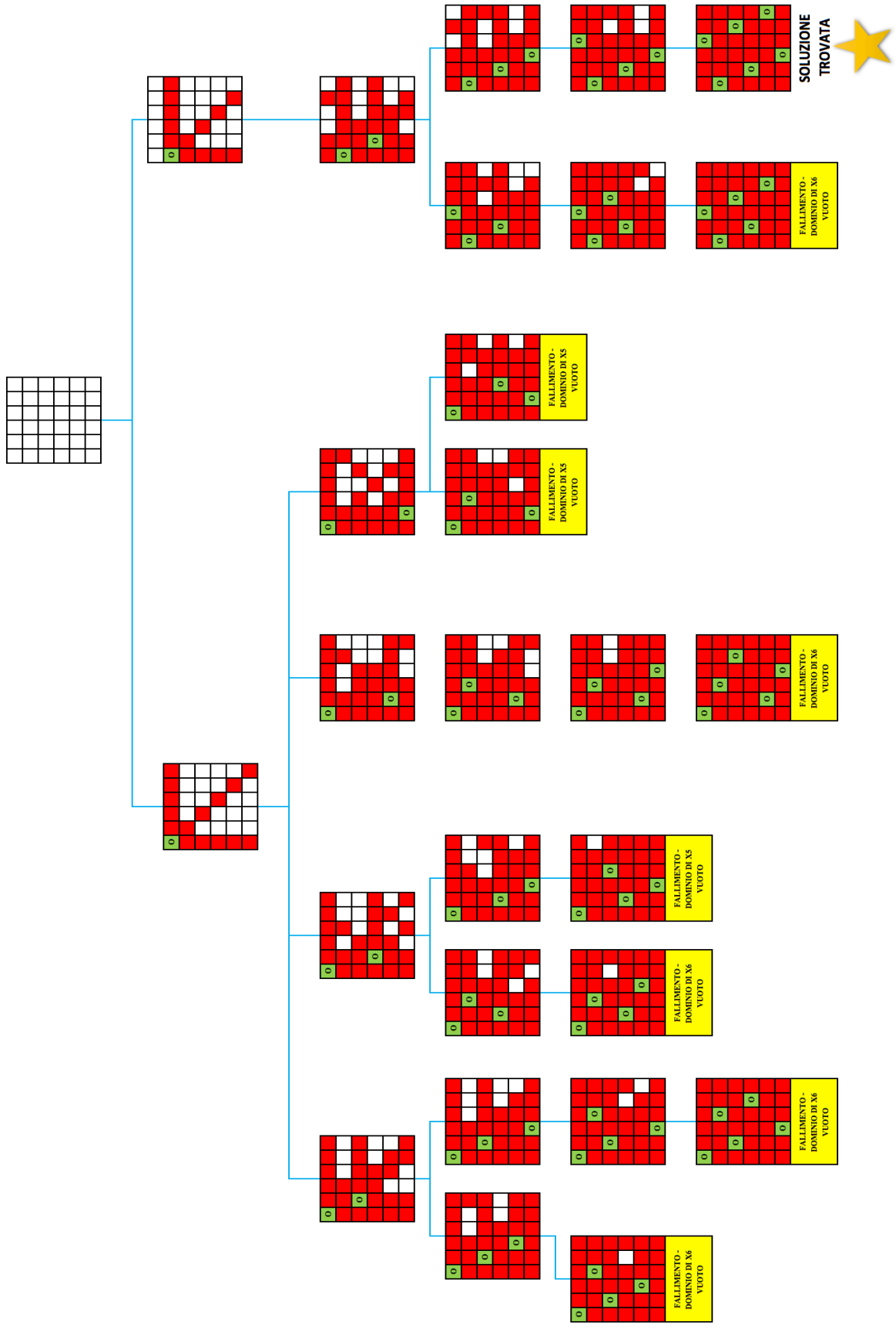
$X_i \neq X_j$ per $1 \leq i < j \leq 6$

Due regine non si devono trovare sulla stessa diagonale

$X_i \neq X_j + (j-i)$ per $1 \leq i < j \leq 6$

$X_i \neq X_j - (j-i)$ per $1 \leq i < j \leq 6$

La figura seguente mostra come utilizzando il backtracking e la propagazione dinamica dei vincoli si arriva a trovare una soluzione ammissibile per il problema, andando a sequenziare le variabili nell'ordine X_1, X_2, X_3, X_4, X_5, X_6. I quadrati verdi rappresentano dove è posizionata la regina mentre quelli rossi i valori che, data l'assegnazione parziale, non possono essere assegnati alle variabili successive. La soluzione ammissibile ottenuta è $X_1=2$, $X_2=4$, $X_3=8$, $X_4=1$, $X_5=3$, $X_6=7$.


SOLUZIONE TROVATA
FALLIMENTO - DOMINIO DI X6 VUOTO
FALLIMENTO - DOMINIO DI X5 VUOTO
FALLIMENTO - DOMINIO DI X5 VUOTO
FALLIMENTO - DOMINIO DI X6 VUOTO
FALLIMENTO - DOMINIO DI X5 VUOTO
FALLIMENTO - DOMINIO DI X6 VUOTO
FALLIMENTO - DOMINIO DI X6 VUOTO
FALLIMENTO - DOMINIO DI X6 VUOTO

Bibliografia

Brailsford, Sally C., Chris N. Potts, and Barbara M. Smith. 1998. "Constraint satisfaction problems: Algorithms and applications." *European Journal of Operational Research* (Elsevier).

Bruno, Giuseppe. 2012. *Operations Management Modelli e metodi per la logistica.* terza edizione. Vol. Capitolo 6. Edizioni Scientifiche Italiane. Accessed Dicembre 16, 2016. https://www.docenti.unina.it/supportoAlleLezioni/VisualizzaContenutoCartellePub.do?codInse=&percorso=/MASTER_PER_LA_GESTIONE_DELLA_SICUREZZA&idDocente=474955534550504542525 54e4f42524e47505036334232314135303 94c&cognomeDocente=BRUNO&nomeDocente=GIUSEPPE.

Feng, Lei. 2015. "Benefits of Constraint Programming." *Lei Feng Tech Blog.* Luglio 1. Accessed Dicembre 21, 2016. https://leifengtechblog.wordpress.com/2015/07/01/benefits-of-constraint-programming/.

IBM. n.d. *IBM ILOG OPL Language User's Manual V6.3.*

—. n.d. *IBM Knowledge Center - ILOG CPLEX Optimization Studio.* IBM. Accessed Marzo 12, 2017. https://www.ibm.com/support/knowledgecenter/en/SSSA5P_12.7.0/ilog.odms.cplex.help/refcppcplex/html/startpoint.html.

Jeffrey, Herrmann. 2006. "A history of production scheduling." In *Handbook of Production Scheduling*, 1-22. Springer US.

Kumar, Vipin. 1992. "Algorithms for constraint satisfaction problem." *AI Magazine* 13 (I).

MBM Italia S.r.l. n.d. "ESS - Enterprise Scheduling System. Approfondimenti."

MBM Italia S.r.l. n.d. *MBM Management Systems.* Accessed 03 06, 2017. http://www.mbm.it.

Pinedo, Michael. 2008. *Scheduling Theory, Algorithms and Systems.* Vols. Part I-II. Springer.

Russel, Stuart, and Peter Norvig. 2009. "Chapter 5: Constraint satisfaction problem." In *Artificial intelligence: a modern approach*. Prentice Hall.

Russo, Salvatore. 2015. "La logistica di produzione del gruppo Danieli. La pianificazione diventa lean." *Il giornale della logistica.*

Slack, Brandon-Jones , Johnston, Vinelli, Romano, and Danese. 2013. *Gestione delle operations e dei processi.* Pearson.

Vollmann, Thomas E., William L. Berry, and Clay D. Whybark. 1997. *Manufacturing planning and control systems.* Quarta edizione. Irwin McGraw-Hill.