UNIVERSITA' DEGLI STUDI DI PADOVA

FACOLTA' DI INGEGNERIA

Corso di laurea in Ingegneria Informatica

TESI DI LAUREA

Ottimizzazione dello stampaggio di articoli tecnici in resina termoplastica

Relatore: Prof. Matteo Fischetti

Laureando: Michele Gallina
Matr. n. 422399*/IF*

ANNO ACCADEMICO 2003/2004

Sommario

Argomento di questa tesi è lo sviluppo e l'implementazione di un algoritmo euristico per un problema di ottimizzazione nel campo della logistica aziendale. Durante un breve periodo di stage presso un'azienda specializzata nello stampaggio di articoli tecnici in resina termoplastica si è studiato ed esaminato in dettaglio il problema di ottimizzare la pianificazione delle lavorazioni sulle presse dell'azienda. Si è passati quindi ad un'analisi approfondita dei vincoli del problema e degli obbiettivi che si intendevano ottimizzare ed esso è stato classificato nell'ambito dei problemi di scheduling su macchine parallele. Nel nostro caso specifico l'obbiettivo della pianificazione deve essere duplice: da un lato bisogna cercare di minimizzare i ritardi delle lavorazioni rispetto alla due date concordata con il cliente, dall'altro la sequenza con la quale le lavorazioni vengono pianificate sulle presse deve cercare di minimizzare i tempi di setup (che dipendono dalla sequenza delle lavorazioni). Il problema è classificabile, nell'ambito della teoria della complessità computazionale, come NP-hard e non sono noti, per questo tipo di problemi, algoritmi in grado di ottenere una soluzione ottima in un tempo polinomiale nella taglia dell'istanza del problema. Si è dunque progettato, e successivamente implementato sfruttando il linguaggio di programmazione C++, un algoritmo euristico che utilizza la tecnica nota come Tabu Search per ottenere "buone" soluzioni in tempi ragionevoli. I primi risultati computazionali ottenuti sull'unica istanza reale del problema che era a disposizione mostrano già l'efficacia del software progettato sia in termini della qualità delle soluzioni ottenute, sia in termini del risparmio di tempo garantito (il programma si graverebbe infatti di buona parte del lavoro attualmente svolto dal personale incaricato di redigere le pianificazioni). Al momento della stesura di questa tesi si stanno ultimando le ultime modifiche al codice del programma per consentire l'inizio di un periodo di sperimentazione del software nel contesto aziendale, durante il quale si cercherà di capire quali cambiamenti apportare (se necessario) alla struttura base del programma per renderlo di fatto operativo.

Indice

Introduzione

Il problema di ottimizzazione oggetto di questa tesi rientra nell'ambito dei problemi che riguardano la logistica aziendale (argomento trattato in maniera esaustiva in [4]), problemi che una qualsiasi impresa si trova a dover affrontare nella prassi manageriale quotidiana e che sono fondamentali per la sopravvivenza e la competitività di ogni azienda.

Nel nostro caso l'azienda, impegnata nel settore dello stampaggio di materie plastiche, intendeva ottimizzare il processo di pianificazione delle lavorazioni sulle proprie macchine. Questo processo rappresenta l'ultima fase operativa nel contesto più ampio di programmazione della produzione aziendale; questa fase, di notevole criticità e complessità, prende il nome di *programmazione operativa* (o anche *scheduling*) e si occupa del sequenziamento delle lavorazioni sulle macchine tenendo conto delle caratteristiche delle lavorazioni stesse, delle caratteristiche dell'impianto a disposizione per la produzione e degli obbiettivi della programmazione. La criticità dello scheduling deriva dal fatto che questa operazione risulta fondamentale per una produzione efficiente nel rispetto dei tempi e delle quantità previste a livello strategico, permettendo così un'elevata tempestività e puntualità nelle consegne. La sua complessità, invece, deriva dall'elevato numero di combinazioni possibili in cui può essere programmata la produzione, dai vincoli che rendono più complesso l'insieme delle configurazioni possibili e dagli obbiettivi, spesso contrastanti, che si vuole raggiungere. Un'efficace ed efficiente metodologia di risoluzione di questo problema risulta dunque un'ottima arma competitiva: attraverso un'attenta pianificazione si possono infatti ridurre al minimo i tempi di setup (durante i quali sono eseguite le operazioni di cambio-lavorazione) con un conseguente aumento dell'efficienza produttiva, cercando allo stesso tempo di garantire un elevato livello di servizio al cliente (minimizzando i ritardi sulle consegne) ottenendo così un considerevole successo nel mercato.

Il problema dello scheduling è stato ampiamente studiato in letteratura e un'ottima fonte utile sia per avere una classificazione dei modelli analizzati, sia per conoscere gran parte dei risultati ottenuti e degli algoritmi elaborati durante tutti questi anni di studio è rappresentata dal libro di M.Pinedo (cfr. [13]).

Data la complessità intrinseca dei problemi di scheduling e in particolare del modello in esame, abbiamo adottato un approccio di tipo euristico per la progettazione di un algoritmo adatto a risolvere il problema. In particolare abbiamo utilizzato la tecnica GRASP (Greedy Randomized Adaptive Search Procedures), descritta ampiamente in [6] e [7], con l'utilizzo della regola di carico ATCS (Apparent Tardiness Cost with Setups)(cfr. [11]) per la prima fase del nostro algoritmo in cui viene costruita una pianificazione iniziale delle lavorazioni.

Nella seconda fase dell'algoritmo abbiamo invece applicato la tecnica di Tabu Search, proposta da Glover in [8] e [9], per migliorare la soluzione iniziale. Questa tecnica è classificata tra le procedure euristiche iterative di ricerca nell'intorno di una soluzione (Neighborhood Search); ad ogni iterazione viene costruito l'intorno della soluzione corrente contenente tutte le possibili soluzioni ottenibili a partire da questa attraverso l'applicazione di una "mossa" e tra esse viene selezionata quella con il minore valore della funzione obbiettivo. Grazie ad un intelligente utilizzo della memoria questa tecnica è in grado a differenza di altre di non restare intrappolata in punti di minimo locale, garantendo così un'esplorazione accurata dello spazio delle soluzioni.

Dopo aver progettato l'algoritmo esso è stato implementato con l'impiego del linguaggio di programmazione C++.

Si descrive ora brevemente la struttura di questa tesi riportando in sintesi il contenuto dei vari capitoli.

Analisi del problema In questo capitolo, dopo un'introduzione al processo di stampaggio ad iniezione di materie plastiche, viene esaminato e classificato il sistema produttivo dell'azienda e vengono analizzati tutti i componenti coinvolti nel processo di produzione e di pianificazione.

La pianificazione della produzione In questo capitolo, dopo una sistemazione dello scheduling nell'ambito della programmazione della produzione, viene introdotta la notazione matematica utilizzata nella classificazione dei modelli del problema di scheduling. In base ad essa viene classificato il problema in esame; nella sezione 2.3 si accenna quindi alla complessità computazionale del problema e nell'ultima sezione del capitolo vengono introdotte alcune delle tecniche euristiche impiegate dall'algoritmo progettato.

Prima fase: costruzione di una soluzione iniziale All'inizio di questo capitolo si trova una descrizione generale della tecnica GRASP e della regola di carico ATCS, utilizzate nella prima fase dell'algoritmo per la costruzione di una soluzione iniziale al problema. Nell'ultima sezione del capitolo viene descritto come queste tecniche sono adattate al problema in esame.

Seconda fase: miglioramento della soluzione iniziale In questo capitolo viene dapprima descritta in dettaglio l'euristica Tabu Search e quindi viene esposto come essa viene implementata durante la seconda fase dell'algoritmo.

Implementazione In questo capitolo vengono descritte le soluzioni software adottate nell'implementazione dell'algoritmo. Nella prima sezione del capitolo viene descritta l'interfaccia grafica del programma attraverso la quale si impostano i dati sulle lavorazioni che si vuole pianificare; nella seconda sezione viene invece illustrata la struttura dell'applicativo C++ che si occupa dell'elaborazione dei dati.

Test e risultati computazionali Nella prima parte del capitolo vengono descritti i test che sono stati effettuati per calibrare alcuni parametri dell'algoritmo e i risultati che essi hanno prodotto. Nella seconda parte del capitolo viene effettuato un confronto sull'istanza reale di dati a disposizione tra la soluzione ottenuta utilizzando il software progettato e la pianificazione redatta dal personale dell'azienda.

Appendice A In questa appendice è riportato il codice sorgente dell'applicativo C++ sviluppato. Sono riportati tutti i file di programma che costituiscono l'applicativo.

Capitolo 1

Analisi del problema

Lo stage è avvenuto presso la ditta Plasticwork S.r.l. del dott. Scamperle Renzo. L'azienda opera nel settore delle materie plastiche ed in particolare è specializzata nello stampaggio di articoli tecnici in resina termoplastica tra cui pezzi per elettropompe, parti di autovetture e accessori per rubinetterie.

Gli articoli vengono prodotti tramite il processo di *stampaggio ad iniezione* grazie alle 21 presse di cui l'azienda è dotata. Lo stampaggio a iniezione (cfr. [5]) dei materiali termoplastici è la tecnologia più diffusa nella trasformazione delle materie plastiche. Si producono, in modo discontinuo, pezzi di forma e dimensioni diverse e dal peso variabile da pochi grammi a svariati chili. Viene ora esposta un breve descrizione di questo processo di produzione.

1.1 Lo stampaggio ad iniezione

Il materiale plastico da trasformare è il componente fondamentale che, dopo un eventuale pretrattamento di essiccazione o deumidificazione, viene aspirato attraverso un sistema di alimentazione all'interno del *cilindro di plastificazione* della pressa (13 in fig. 1.1). Nel cilindro è situata una *vite punzonante di plastificazione* (12 in fig. 1.1) che, ruotando e traslando per mezzo di *pistoni idraulici* (15 in fig. 1.1), crea un attrito che, unito al contributo termico generato dalle *resistenze elettriche* (10 in fig. 1.1) situate sul cilindro di plastificazione, provoca la fusione del materiale.

Dopo che il materiale ha raggiunto la viscosità necessaria, viene iniettato ad una certa velocità all'interno dello *stampo* (5 e 6 in fig. 1.1), passando attraverso opportuni canali e riempiendo la cavità che rappresenta in negativo il pezzo (fase di iniezione o riempimento). Riempita la cavità inizia la fase di mantenimento durante la quale il polimero viene tenuto sotto pressione allo scopo di compensare con altro materiale l'aumento della densità connesso con la diminuzione di temperatura e la solidificazione che avvengono durante il raffreddamento del pezzo.

Il polimero fuso entra nella cavità attraverso l'*ugello di iniezione* (7 in fig. 1.1) o gate; la solidificazione del polimero al gate determina la fine della fase di mantenimento. Una volta che il gate si è solidificato, non può entrare più polimero in cavità qualunque sia la pressione che esercita la vite e inizia la fase di raffreddamento durante la quale il manufatto continua la solidificazione.

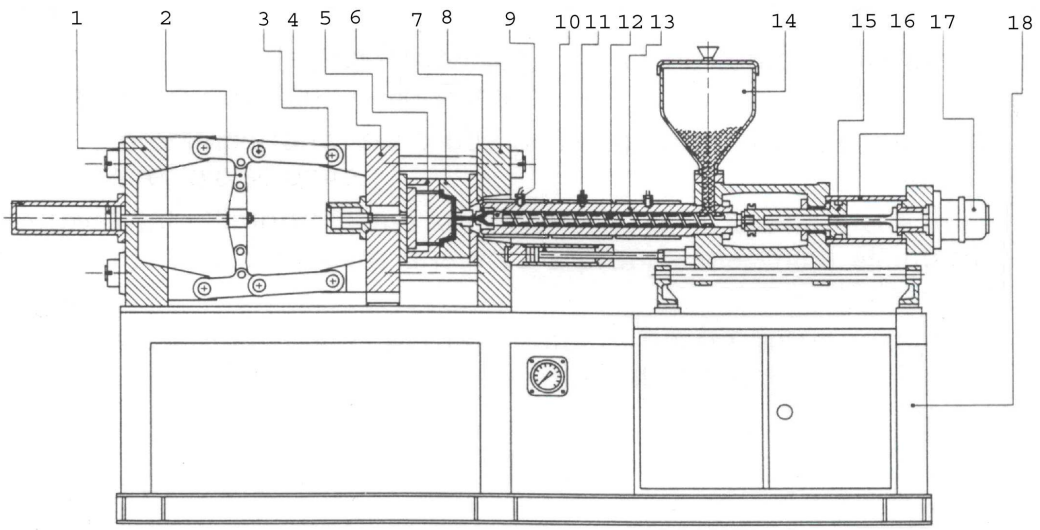

Figura 1.1: Sezione longitudinale di una pressa ad iniezione a vite punzonante del tipo a doppia ginocchiera.
Legenda: 1)Testata stampo 2)Chiusura a doppia ginocchiera 3)Estrattore oleodinamico 4)Piastra mobile 5)Semistampo mobile 6)Semistampo fisso 7)Ugello di iniezione 8)Piastra fissa 9)Camera di plastificazione 10)Resistenze elettriche di riscaldamento 11)Termocoppia 12)Vite punzonante di plastificazione 13)Cilindro di plastificazione 14)Tramoggia 15)Pistone iniezione 16)Cilindro iniezione 17)Motore oleodinamico 18)Basamento.

Lo stampaggio a iniezione è un processo discontinuo, quindi esso è composto da diverse fasi che costituiscono un *ciclo*. Riassumendo, le fasi di un ciclo di stampaggio sono:

1. Chiusura stampo: in questa fase la *piastra mobile* (4 in fig. 1.1) della pressa si chiude serrando le due *metà dello stampo* (5 e 6 in fig. 1.1).

2. Accostamento del gruppo di iniezione: l'accostamento avviene in maniera semiautomatica durante il primo ciclo.

3. Iniezione: il materiale fuso situato nella *camera di plastificazione* (9 in fig. 1.1) viene spinto dalla vite entro la cavità dello stampo con una certa velocità di plastificazione.

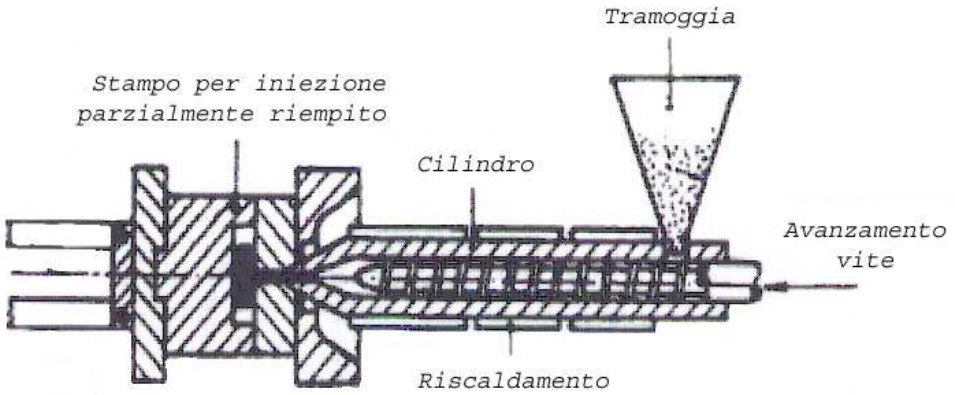

Figura 1.2: Riempimento stampo

4. Mantenimento e dosaggio: il polimero fuso che è all'interno dello stampo viene tenuto sotto pressione in modo da evitare il riflusso del materiale dallo stampo alla camera di plastificazione. La vite viene fatta ruotare per permettere al materiale di passare dalla *tramoggia* (14 in fig. 1.1) (il dispositivo sopra il cilindro della macchina usato per contenere i granuli di resina) alla zona di accumulo (zona prossima all'ugello del cilindro di plastificazione) e, trovando davanti all'ugello il materiale solidificato, il materiale per spinta fa arretrare la vite.

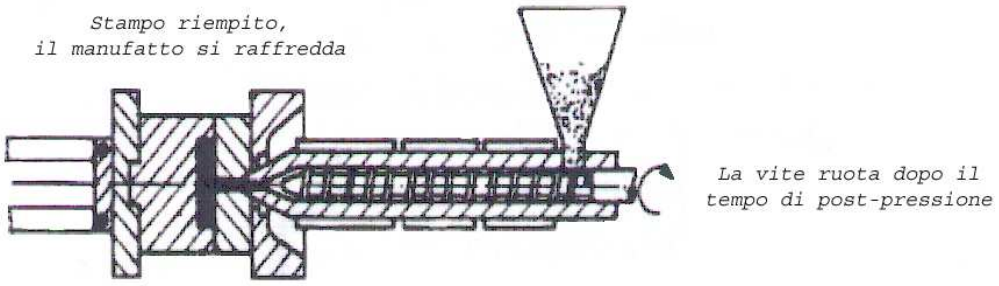

Figura 1.3: Arretramento vite

5. Raffreddamento: in questa fase la pressa non esegue alcun movimento. La durata del tempo di raffreddamento è funzione del tipo di polimero utilizzato e della geometria del manufatto e deve permettere al materiale di solidificarsi all'interno dello stampo.

6. Apertura stampo ed estrazione: in questa fase si apre la parte mobile dello stampo con la piastra mobile della pressa; arrivato a fine corsa entra in funzione il gruppo di estrazione della pressa che agendo sul sistema di estrazione dello stampo espelle il pezzo stampato.

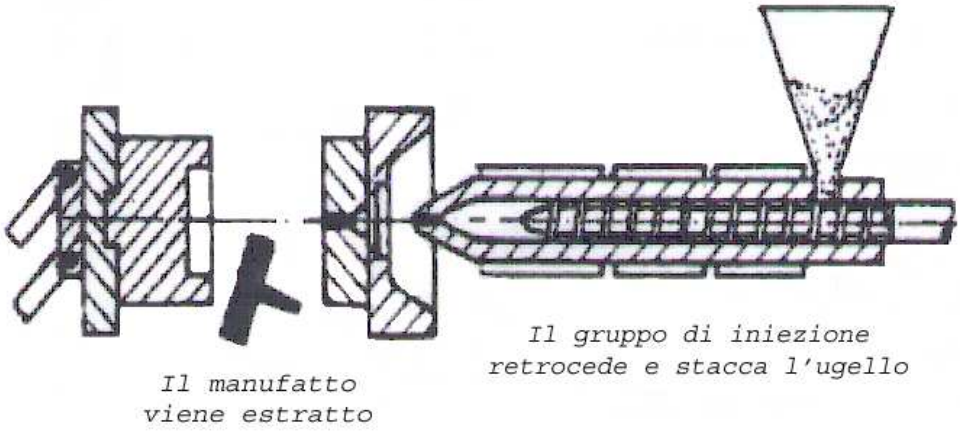

Figura 1.4: Apertura stampo ed estrazione del pezzo

7. Chiusura dello stampo: finita l'estrazione la piastra mobile si richiude e riparte così il ciclo.

Vediamo ora di introdurre brevemente il contesto produttivo in cui l'azienda opera.

1.2 Il sistema produttivo

Vi sono diversi parametri che bisogna analizzare quando si vuole classificare il sistema produttivo di un'azienda. Il modello (schematizzato in fig. 1.5) che ho preso in considerazione (cfr. [4]) tiene conto di tre profili:

1. la modalità di manifestazione della domanda;
2. la modalità di predisposizione dell'offerta;
3. le caratteristiche intrinseche del prodotto.

Per quanto concerne il primo profilo si possono individuare tre casi tipici:

Produzioni su commesse singole. In questo caso l'azienda riceve una serie di ordini diversi per singoli prodotti, differenziati anche notevolmente, per i quali solitamente occorre sviluppare il progetto (parzialmente o totalmente) ed elaborare il ciclo di lavorazione.

Produzioni su commesse ripetitive. Rientrano in questa classe sia le imprese che realizzano una gamma di prodotti dalle caratteristiche definite per un gruppo di clienti abbastanza stabile che richiede forniture scaglionate nel tempo, sia le aziende che producono "su catalogo" ma solo dopo il manifestarsi dell'ordine.

Produzioni su previsione. In questo caso l'azienda produce volumi abbastanza elevati dei suoi prodotti prima del manifestarsi degli ordini, basandosi sulla previsione della domanda.

Per quanto concerne il secondo profilo, cioè le modalità secondo le quali viene realizzato l'output, anche in questo caso abbiamo tre diverse situazioni possibili:

Produzioni unitarie. In questo caso l'attività produttiva è organizzata in funzione delle richieste dei singoli ordini.

Produzioni intermittenti. In questo caso i prodotti sono realizzati in lotti di entità superiore ai fabbisogni immediati, così da formare delle scorte che possono essere utilizzate in seguito.

Produzioni continue. In quest'ultimo caso i cicli produttivi restano costanti per lunghi periodi di tempo dando luogo ad un flusso ininterrotto di prodotti dalle caratteristiche omogenee.

Il terzo ed ultimo profilo utilizzato per caratterizzare un sistema produttivo riguarda il modo in cui i prodotti vengono realizzati. Si possono individuare due casi:

Produzioni per processo. In questo caso gli elementi che costituiscono il bene finale non possono essere facilmente identificati; il prodotto non può cioè essere scomposto a ritroso perché i componenti originari non sono più distinguibili o hanno cambiato natura.

Produzioni per parti. In questo caso il bene ottenuto è costituito da un certo numero di componenti in genere di diversa natura. Il processo è costituito sia dalla fase di *fabbricazione* sia da quella successiva di *montaggio*.

Il sistema produttivo dell'azienda è classificabile come "produzioni su commesse ripetitive" rispetto al primo profilo, infatti tutte le produzioni dipendono dagli ordini dei clienti. Rispetto al secondo profilo di classificazione il sistema è di tipo "produzioni unitarie", in quanto ogni lavorazione produce il numero di pezzi richiesti da un ordine di un cliente e non vengono tenute scorte degli articoli (salvo casi particolari); infine rispetto al terzo profilo il sistema è di tipo "produzione per processo", in quanto tramite il processo di stampaggio descritto si ha la trasformazione del materiale plastico nel prodotto finale.

L'azienda intende ottimizzare il processo di pianificazione (scheduling) delle lavorazioni sulle presse, cercando sia di riuscire a soddisfare l'esigenza di rispettare per quanto possibile le date di consegna imposte dai clienti sia di ridurre al minimo i tempi di setup durante le operazioni di cambio-stampo che comportano uno stop temporaneo alla produzione. Prima di analizzare approfonditamente il problema dello scheduling (Cap. 2), si esaminano i componenti coinvolti nel processo di produzione e di pianificazione, le operazioni di cambio-stampo e infine il sistema attualmente utilizzato per la pianificazione delle lavorazioni. I dettagli "anagrafici" dei componenti del sistema produttivo sono memorizzati nei database del software del sistema informatico attualmente utilizzato per la gestione degli ordini e la pianificazione delle produzioni.

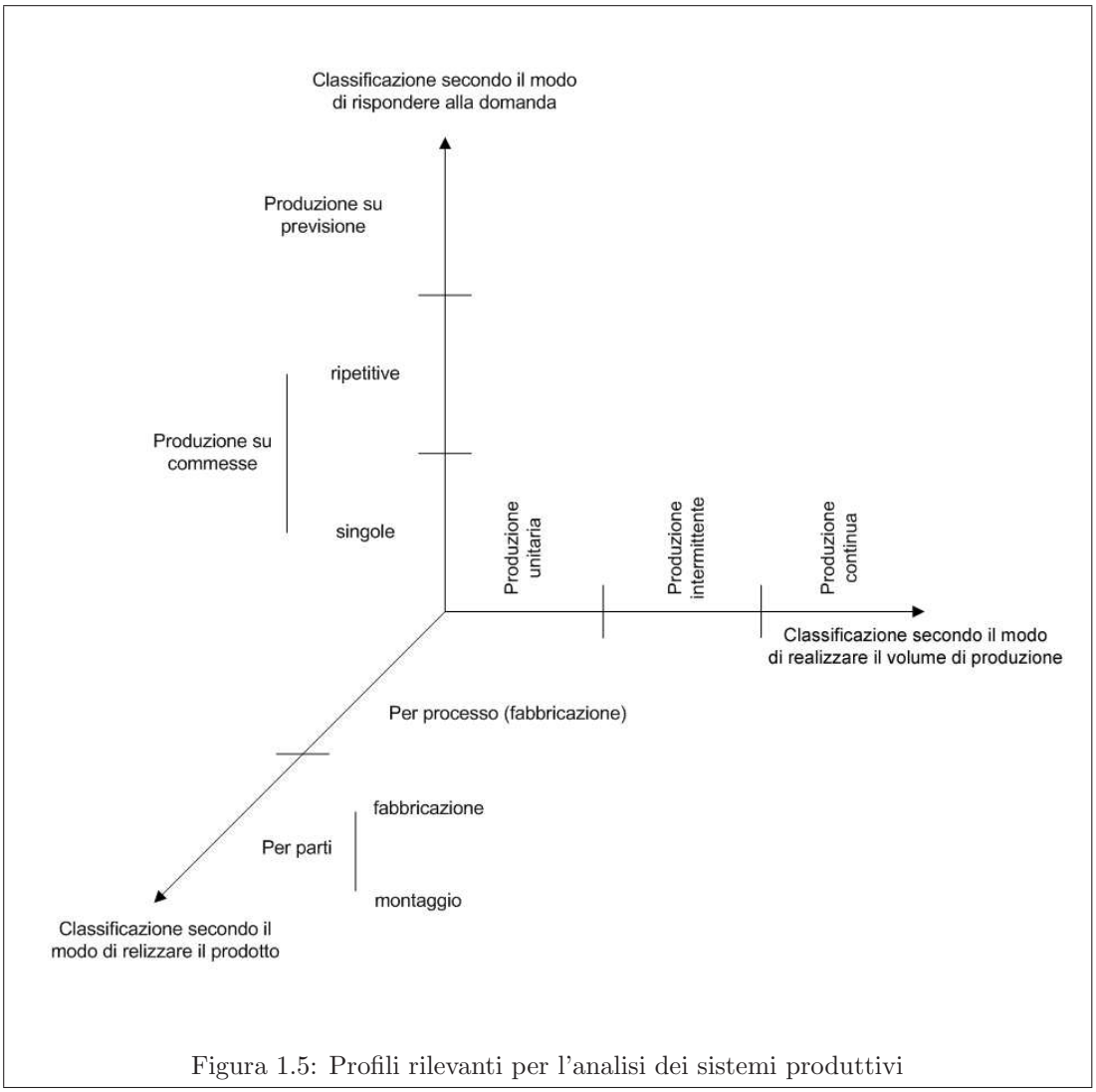

Figura 1.5: Profili rilevanti per l'analisi dei sistemi produttivi

1.3 Gli ordini dei clienti

Gli ordini arrivano con frequenza giornaliera dai vari clienti che si riforniscono dall'azienda. Ogni commessa è caratterizzata dai seguenti parametri:

- il *cliente* che ha effettuato l'ordine;
- il *prodotto* che si richiede;
- la *data di consegna*, cioè la data entro cui si vorrebbe avere a disposizione la merce;
- la *quantità richiesta*, espressa come numero di pezzi dell'articolo.

Non tutti i clienti rivestono la stessa importanza per l'azienda; per alcuni di essi sono ammessi (e nel mercato odierno sono ormai all'ordine del giorno) dei ritardi nelle consegne, mentre per altri si ha un occhio di riguardo cercando di soddisfare per tempo, qualora possibile, le loro richieste. Possiamo dire che ad ogni commessa è associata una *priorità* dovuta principalmente all'importanza del relativo cliente.

1.4 I prodotti

I prodotti rappresentano il risultato finale, pronto quindi per la consegna ai clienti, del processo di produzione. Gli articoli plastici prodotti sono caratterizzati da:

- il *materiale plastico* che deve essere usato per lo stampaggio;
- il *colore*, che può essere proprio del materiale usato oppure ottenuto con l'aggiunta di un master (pigmento colorante);
- lo *stampo* che deve essere utilizzato, con l'eventuale indicazione della *versione* nel caso di stampo *multiversione*;
- la presenza o meno di *inserti* metallici.

1.5 Le materie plastiche

Le materie plastiche utilizzate per lo stampaggio ad iniezione si dividono in due grandi categorie:

- resine termoindurenti;
- resine termoplastiche;

Le prime induriscono sotto l'azione del calore e della pressione e loro caratteristica peculiare è che il processo non è reversibile e quindi, una volta stampate, non possono essere riutilizzate se non come materiale riempitivo.

Le resine termoplastiche al contrario fondono sotto l'azione del calore fino a raggiungere condizioni di fluidità tali da poter essere iniettate in uno stampo; raffreddandosi esse si induriscono ma se successivamente riscaldate, fondono nuovamente garantendo la reversibilità del processo.

Ogni resina (termoplastica o termoindurente che sia) può essere più o meno caricata con altri materiali (vetro, amianto, mica, fibre di carbonio, ecc...) che vengono aggiunti per migliorare le caratteristiche di resistenza alla flessione, alla trazione, all'urto, di durezza, di elasticità, per facilitare la messa in opera, nonché in alcuni casi per ridurne il prezzo. Ogni materiale, in base al processo chimico (detto di polimerizzazione) attraverso il quale è stato ottenuto, ai polimeri

che lo compongono ed agli eventuali additivi aggiunti appartiene dunque ad uno specifico *tipo* (o *famiglia*) di resine (polimeri dello stirolo, poliuretani, policarbonati, poliesteri, ecc. . .).

In alcuni casi il materiale plastico utilizzato può avere già un suo specifico *colore*; in altre situazioni l'aggiunta di un pigmento denominato *master* serve ad ottenere la colorazione desiderata.

1.6 Gli stampi

Lo stampo è un pezzo meccanico progettato per dare forma e dimensioni desiderate ad un volume di materiale fuso. È costituito da due parti separabili in modo da poter estrarre il pezzo finito. Ogni stampo è caratterizzato dai seguenti parametri:

- la caratteristica di essere o meno uno stampo *multiversione*;
- il numero di *impronte*, cioè le cavità identiche dello stampo che rappresentano il negativo dell'articolo che si vuole stampare;
- il *tempo ciclo*, cioè il tempo che intercorre tra un certo punto del ciclo di stampaggio e lo stesso punto nel ciclo successivo;
- il *tempo di Avviamento*, che rappresenta il tempo, stimato in minuti, necessario per la messa a regime dello stampo per operare in condizioni di produzione di qualità.
- il *tempo di Attrezzaggio*, cioè il tempo, stimato in minuti, necessario per il montaggio dello stampo su di una pressa;
- il *tempo di Disattrezzaggio*, cioè il tempo, stimato in minuti, necessario per eseguire l'operazione di smontaggio dello stampo a fine produzione;
- l'indicazione delle *presse* sulle quali lo stampo può venire montato.

Se uno stampo è *multiversione*, questo significa che aggiungendo o rimuovendo alcune tasselli allo stampo è possibile ottenere una conformazione differente delle impronte e quindi in definitiva una nuova *versione* dell'articolo prodotto. Ogni stampo multiversione è dunque caratterizzato anche da una sigla che ne specifica la sua *versione*.

1.7 Le presse

Le presse sono classificate in base alla *forza di chiusura* che, nello stampaggio ad iniezione, è la pressione, esercitata dal gruppo di chiusura, applicata allo stampo per tenerlo chiuso in opposizione alla pressione del materiale fuso compresso all'interno della cavità e del canale di alimentazione. La forza di chiusura (misurata in tonnellate) deve essere superiore alla pressione d'iniezione per evitare che lo stampo si apra. Le 21 presse dell'azienda sono così suddivise in base alla forza di chiusura:

- 2 presse da 65 tonnellate;
- 1 pressa da 80 tonnellate;
- 6 presse da 100 tonnellate;
- 3 presse da 120 tonnellate;
- 1 pressa da 125 tonnellate;
- 1 pressa da 130 tonnellate;

- 2 presse da 135 tonnellate;
- 1 pressa da 200 tonnellate;
- 1 pressa da 220 tonnellate;
- 1 pressa da 350 tonnellate;
- 1 pressa da 440 tonnellate;
- 1 pressa da 800 tonnellate;

Tutti gli stampi necessari per la produzione degli articoli plastici possono essere montati su di un sottoinsieme delle 21 presse; le due caratteristiche che si considerano per stabilire se un articolo plastico può essere stampato su di una pressa sono la *forza di chiusura* della pressa e la sua *lunghezza di estrazione* che stabilisce date le dimensioni dello stampo se la pressa è in grado di ospitarlo.

1.8 Le operazioni di cambio-stampo

Il tempo si setup è definito come *l'intervallo di tempo che intercorre tra la fine della produzione dell'ultimo "pezzo" conforme (senza difetti) del lotto precedente il setup, e l'inizio della produzione del primo "pezzo" conforme del lotto successivo.* In particolare il tempo di setup è suddivisibile in due entità ben distinte:

- tempo di ***setup interno,***
- tempo di ***setup esterno.***

Il primo è definito come quell'intervallo di tempo durante il quale la macchina (o la linea, o il processo produttivo) deve essere fermata altrimenti non sarebbe possibile effettuare il setup. Questo è il vero e proprio tempo di setup, che inizia alla fine del lotto precedente e termina all'inizio del lotto successivo. Durante quest'intervallo di tempo non si aggiunge alcun valore al prodotto.

Il tempo di setup esterno è invece definito come quell'intervallo di tempo che trascorre durante le operazioni produttive sia del lotto precedente che di quello successivo, durante il quale si effettuano alcune attività necessarie per il setup (come portare o rimuovere materiali e prodotti, preparare o mettere a posto attrezzi, ecc...). Una parte di tali attività "esterne", che possono essere effettuate da personale vario (operatori, tecnici, manovalanza...), viene eseguita prima delle attività di setup interne e una parte dopo. Vediamo allora nel nostro caso quali sono le operazioni effettuate nelle due distinte fasi di setup.

Setup Esterno

Prima che la produzione del lotto precedente sia terminata, il materiale necessario alla lavorazione successiva viene preparato, iniziando eventualmente qualche ora prima il pretrattamento di essiccazione o di deumidificazione, nei silos del sistema centralizzato tramite cui sono alimentate tutte le presse dell'azienda. Lo stampo necessario alla lavorazione successiva e il suo sistema di raffreddamento o di riscaldamento viene preparato a bordo pressa insieme a tutta l'attrezzatura necessaria al suo montaggio. Dopo l'inizio della nuova produzione (terminata cioè la fase di setup interno) si provvede a riporre in magazzino lo stampo utilizzato dopo averlo trattato con alcuni oli protettivi, e a pulire il silos contenente il materiale plastico della lavorazione precedente.

Setup Interno

Questa come detto è la fase più importante del setup, in cui la produzione viene arrestata e che quindi bisogna cercare di abbreviare il più possibile. In questa fase si provvede a smontare lo stampo appena utilizzato e tutti i collegamenti necessari al suo sistema di raffreddamento o riscaldamento e a montare il nuovo stampo collegando i dispositivi di riscaldamento o raffreddamento. Se lo stampo utilizzato per la lavorazione successiva è lo stesso della lavorazione appena conclusa o è semplicemente una sua diversa versione, queste operazioni sono ovviamente decisamente più rapide. Dopo avere provveduto alla sostituzione dello stampo si passa alla fase di "spurgo" della pressa.

Lo spurgo è il processo di rimozione della resina dal cilindro della pressa. Quando si deve sostituire la resina, il materiale residuo nel cilindro viene spurgato prima dell'introduzione di un'altra resina. Lo spurgo è inoltre importante nelle fasi d'avvio e d'arresto, quando la resina potrebbe essere esposta per molto tempo ad alte temperature. Lo spurgo può in alcuni casi essere effettuato anche per eliminare del materiale degradato dal cilindro e dalla vite nella pressa. La fase di spurgo ha una durata che dipende considerevolmente dalle resine coinvolte; infatti se le due resine che devono alternarsi sulla pressa fanno parte della stessa "famiglia" lo spurgo avviene semplicemente caricando il nuovo materiale nella tramoggia e svuotando completamente il cilindro della pressa facendo avanzare più volte la vite punzonante senza ovviamente che il gruppo di iniezione sia collegato allo stampo. Quando invece le due resine sono molto differenti e anche qualora il colore della nuova resina sia meno coprente del colore della resina precedente, per pulire a fondo il cilindro di plastificazione così da non rischiare di ottenere dei pezzi non conformi (ad esempio con delle striature di un altro colore), viene caricato nella tramoggia un materiale di spurgo per elevate temperature e l'operazione prosegue fino a che il materiale spurgato non appare trasparente. Un altro fattore importante da considerare per cercare di ridurre i tempi durante questa fase di setup è la temperatura a cui il materiale deve essere mantenuto nel cilindro di plastificazione; se infatti passiamo da una resina con una temperatura di 350 °C ad una con temperatura di 130 °C impiegheremo molto tempo aspettando che il cilindro della pressa si raffreddi. Queste temperature dipendono ancora dal tipo di resina considerato; resine appartenenti alla stessa "famiglia" avranno ovviamente temperature molto simili e quindi questa fase del processo risulterà abbreviata.

Una volta spurgata la pressa e montato il nuovo stampo, vengono impostati i parametri ottimali di stampaggio attraverso alcune stampate di prova prima di ottenere il primo pezzo "conforme" del nuovo lotto di produzione (tempo di Avviamento).

1.9 Il sistema attualmente in uso

L'azienda utilizza attualmente un software che permette la gestione delle "anagrafiche" dei materiali e dei coloranti (master) utilizzabili per lo stampaggio, delle presse, degli stampi e dei prodotti. Oltre alla gestione delle anagrafiche il software consente la pianificazione delle commesse sulle diverse presse; tuttavia la scelta effettiva della sequenza con la quale si vogliono effettuare le produzioni sulle presse è lasciata totalmente all'operatore. Quest'ultimo ogni qualvolta riceve una nuova commessa da pianificare, la inserisce nel database scegliendo la pressa e la posizione relativa alle altre commesse già pianificate sulla stessa pressa nella maniera che ritiene più opportuna. Per esempio potrebbe scegliere di inserire la nuova commessa subito dopo la fine della commessa attualmente in produzione sulla pressa, oppure potrebbe inserirla tra due commesse future già pianificate, o ancora pianificarla come ultimo lavoro.

Il programma provvede a calcolare in base ai dati della commessa (quali la quantità di pezzi richiesta e il tipo di articolo) la durata della produzione calcolando l'istante iniziale e finale per la lavorazione inserita e per quelle già pianificate che venissero spostate a causa dell'inserimento della nuova commessa. Come detto l'onere computazionale legato alla scelta dell'ordinamento

delle commesse grava interamente sull'operatore; poiché nuove commesse arrivano quotidianamente succede spesso di dover cambiare radicalmente una pianificazione precedentemente redatta per soddisfare alcuni criteri di ottimalità (riduzione dei ritardi delle consegne, minimizzazione dei tempi di setup). Tale cambiamento comporta ogni volta un consistente dispendio in termini di tempo ed energie che l'azienda intenderebbe risparmiare con lo sviluppo di un software ad hoc che gestisca automaticamente lo scheduling delle commesse sulle presse.

Il programma potrebbe in questo modo sostituire quasi completamente il lavoro dell'operatore; oltre a garantire un risparmio in termini di tempo il software richiesto deve cercare di offrire soluzioni con una qualità, in termini dei criteri di ottimalità sopracitati, paragonabile a quelle ottenibili attualmente dall'operatore senza l'ausilio di alcun programma ma in base all'esperienza accumulata in anni di lavoro. Come detto tutte le informazioni riguardanti le presse, i materiali ed i master, gli stampi ed i prodotti finali sono memorizzate nei database del software attualmente utilizzato; per gli stampi manca tuttavia l'indicazione delle presse su cui possono essere montati. Infatti questa informazione non era necessaria in quanto era l'operatore a pianificare le lavorazioni sulle macchine.

Capitolo 2

La pianificazione della produzione

2.1 Il contesto generale

In generale si può definire la *programmazione della produzione* come quell'insieme di attività che permettono, a fronte di ordini o previsioni di vendita dei prodotti finali, di:

- generare gli ordini di produzione;
- assegnare gli ordini di produzione alle differenti unità produttive;
- pianificare i fabbisogni delle materie prime;
- sequenziare le lavorazioni sulle singole macchine operatrici.

Data la notevole complessità di questo processo un approccio generalmente utilizzato per affrontare la programmazione è quello di tipo *gerarchico*: esso consiste nel dividere il processo nelle seguenti fasi:

1. *Programmazione di lungo periodo,* il cui risultato è un'indicazione di massima di "quanto" si dovrà produrre e di "quante" e "quali" risorse produttive (manodopera, impianti, materie prime) si avrà bisogno;
2. *Programmazione di medio periodo,* che ha come obbiettivo la formulazione del Master Production Schedule (MPS) che rappresenta un piano principale di produzione per ogni periodo dell'orizzonte temporale considerato;
3. *Programmazione di breve periodo* (detta anche programmazione operativa o Scheduling), che prende decisioni riguardo al sequenziamento delle operazioni da effettuare, alla gestione della movimentazione interna, all'assegnazione del personale ai singoli centri di lavoro tenendo conto di un elevato numero di vincoli quali la capacità produttiva effettivamente disponibile, le date di consegna richieste per i singoli prodotti, l'effettiva presenza delle materie prime, lo stato delle attrezzature e così via. Essa è generalmente effettuata a valle della fase di pianificazione dei fabbisogni di componenti e materie prime.
4. *Controllo della produzione,* il cui compito è controllare l'esecuzione del piano operativo e restituire al sistema di programmazione lo stato di avanzamento della produzione.

La struttura complessiva della Programmazione della Produzione è descritta dallo schema in fig. 2.1.

La parte di questa struttura che ci interessa maggiormente, per quanto riguarda il problema oggetto di questa tesi, è la fase di *Scheduling*, nella quale si deve riuscire a produrre quanto

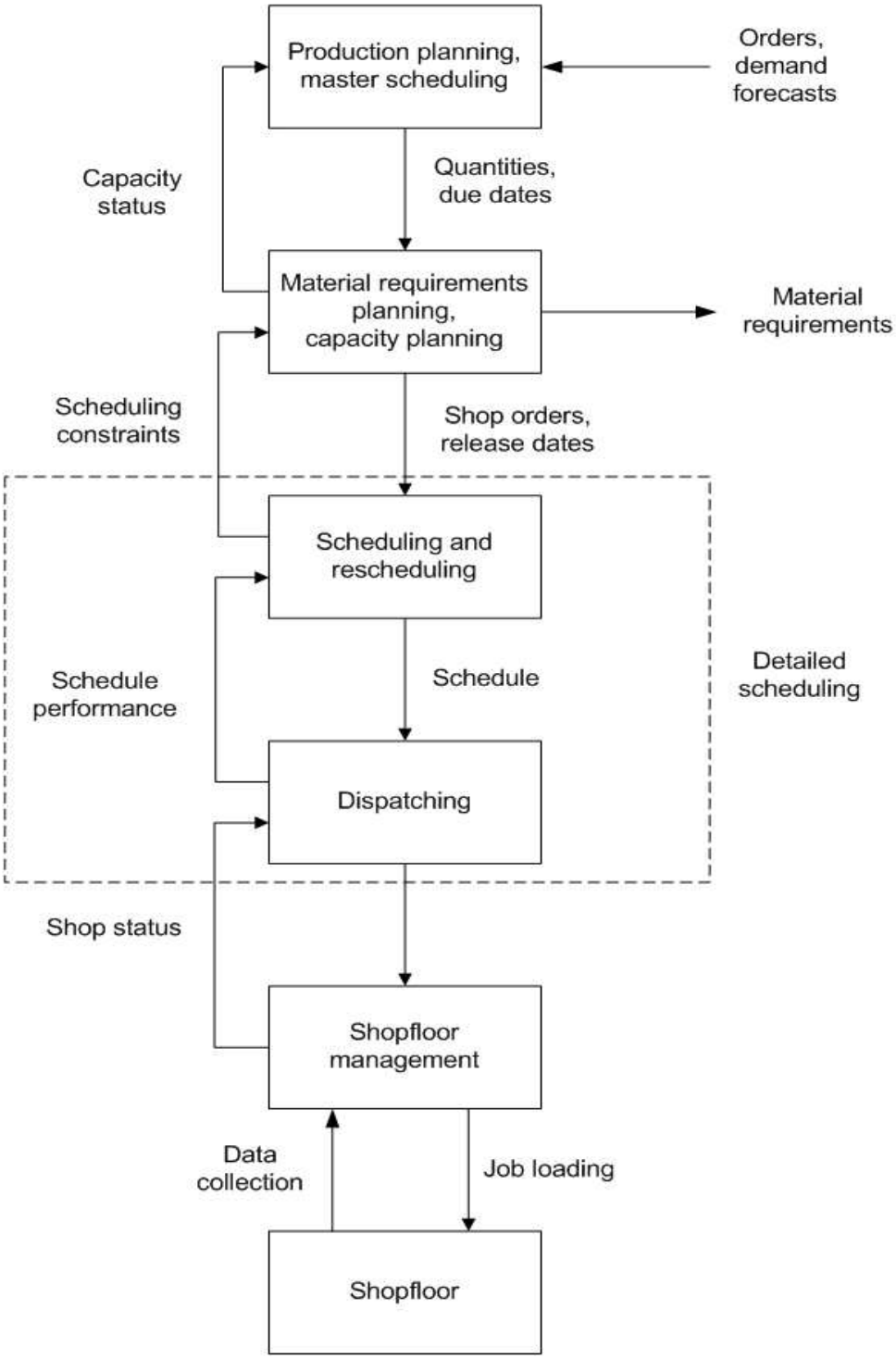

Figura 2.1: Schema della programmazione della produzione

deciso in fasi più a monte avvalendosi di materiali e risorse fissati a loro volta nelle fasi precedenti del processo di produzione, nel rispetto dei vincoli già elencati. Nella fase di programmazione operativa è necessario formulare una serie di ipotesi indispensabili per poter trattare i problemi reali sia da un punto di vista teorico, sia pratico-operativo. Alcune di queste ipotesi sono:

- le risorse utilizzate sono note a questo livello della programmazione e non è possibile stabilire aumenti di orario o di macchinario;
- nella maggior parte dei modelli la risorsa critica è costituita dalle macchine del sistema;
- le lavorazioni sono completamente definite, e per ognuna di esse è nota la data di possibile inizio lavorazione (*release date*) e la data di consegna (*due date*) (se considerate dal modello);
- i tempi di trasporto sono trascurabili;
- tutte le lavorazioni devono essere compiute, non si ammettono annullamenti di ordini;
- una macchina non può lavorare più di una commessa alla volta;
- nella maggior parte dei modelli (e anche nel nostro) una commessa non può essere lavorata contemporaneamente su più macchine;
- data la brevità dell'orizzonte temporale considerato vengono trascurati i costi di mantenimento a scorta, cioè non viene assegnata alcuna penalità all'"anticipo" di una commessa.

Le ipotesi viste, in particolare la presenza di tempi di lavorazione assegnati e l'impossibilità di lavorare contemporaneamente più commesse su di una stessa macchina sottointendono l'esistenza di una capacità produttiva limitata delle macchine. Per questo motivo a volte la programmazione operativa è indicata genericamente come programmazione a capacità finita, in contrapposizione alla programmazione a capacità infinita tipica delle fasi a monte di pianificazione dei fabbisogni.

2.2 Lo Scheduling

Nelle ultime quattro decadi il problema dello Scheduling è stato al centro di un considerevole studio di ricerca teorica che ha portato all'introduzione di una notazione matematica per la classificazione dei diversi modelli deterministici del problema. Prima di esporre questa notazione vengono descritte le due entità fondamentali di ogni problema di Scheduling, ovvero i lavori e le macchine:

Lavoro (job) Con questo termine si intende identificare l'operazione di produzione che viene processata su di una macchina.

Macchina (machine) Con questo termine si intende il dispositivo (inteso come singolo dispositivo o come insieme di dispositivi che lavorano congiuntamente) atto all'esecuzione dei lavori.

2.2.1 La notazione matematica

Nei problemi di scheduling si assume che il numero delle macchine e dei lavori sia sempre finito; generalmente si indica con m il numero delle macchine, mentre si usa n per indicare il numero dei lavori. La lettera i viene utilizzata come pedice di una grandezza per riferirsi alla macchina, mentre con j ci si riferisce al lavoro. Le grandezze che si utilizzano per descrivere un job j sono:

Processing time (p_{ij}) (Tempo di lavorazione) che rappresenta il tempo di produzione del lavoro j sulla macchina i; nel caso in cui il tempo dipenda esclusivamente dalla lavorazione e non dalla macchina su cui viene eseguita si indica semplicemente con $\boldsymbol{p_j}$.

Release date (r_j) (Data di possibile inizio) che rappresenta l'istante di tempo prima del quale non si può iniziare l'esecuzione della lavorazione: può rappresentare, ad esempio, l'istante in cui diventano disponibili le materie prime o i semilavorati per la lavorazione di un certo prodotto.

Due date (d_j) (Data di consegna) che rappresenta l'istante di tempo entro cui si vorrebbe che la lavorazione venisse completata (in modo da rispettare la data di consegna promessa al cliente). Il completamento della produzione di un job dopo la sua due date è permesso, a patto di pagare una *penalità*; nel caso in cui la due date debba assolutamente essere rispettata, essa prenda il nome di **deadline** e si indica con $\overline{\boldsymbol{d}}_{\boldsymbol{j}}$.

Weight (w_j) (Peso) che rappresenta un indice di priorità della lavorazione j rispetto agli altri job del sistema.

Un problema di scheduling è descritto da una tripletta $\alpha|\beta|\gamma$ in cui il campo α descrive il sistema produttivo considerato, il campo β fornisce dei dettagli sulle caratteristiche del processo e sui vincoli che devono essere rispettati, mentre γ descrive la funzione obbiettivo che si vuole minimizzare. Esaminiamo più in dettaglio questi tre campi.

Parametro α

I possibili valori che il parametro può assumere sono:

Single machine (1) (Macchina singola) In questo caso la pianificazione riguarda un'unica risorsa produttiva. Il caso di macchina singola è il caso più semplice tra i vari sistemi produttivi considerati e per questo anche uno tra i più studiati in letteratura.

Identical machines in parallel (Pm) (Macchine parallele identiche) In questo caso i prodotti possono essere indifferentemente lavorati su m macchine identiche. Se un job j non può essere lavorato su qualsiasi macchina ma soltanto su di uno specifico sottoinsieme M_j delle m macchine, allora la notazione $\boldsymbol{M_j}$ viene utilizzata nel campo β come vedremo in seguito.

Machines in parallel with different speed (Qm) (Macchine parallele generiche) In questo caso ci sono ancora m macchine come nel caso precedente ma i tempi di lavorazione dipendono oltre che dal job j anche dalla macchina i su cui il lavoro viene eseguito. Ogni macchina è caratterizzata da una sua velocità v_i.

Unrelated machines in parallel (Rm) (Macchine parallele scorrelate) Il sistema produttivo è una generalizzazione del caso precedente. La macchina i può eseguire il job j con una velocità v_{ij}. Se le velocità delle macchine fossero indipendenti dai job (cioè $v_{ij} = v_i \ \forall j$) si ricadrebbe nel caso precedente.

Flow shop (Fm) In questo caso ci sono m macchine in serie ed ogni job è caratterizzato da un ciclo tecnologico che richiede l'intervento di ognuna di esse; l'ordine con cui le operazioni vengono eseguite sulle diverse macchine (routing) è tuttavia lo stesso per tutti i job.

Flexible flow shop (FFc) È una generalizzazione dei sistemi produttivi di flow shop e di parallel machines. Invece di avere m macchine in serie, vi sono c stadi di produzione in serie, ognuno con un certo numero di identiche macchine in parallelo. Ogni job deve essere lavorato prima allo stadio 1, poi allo stadio 2 e così via, e ad ogni stadio può essere lavorato su una qualunque delle macchine di quello stadio.

Job shop (Jm) Anche in questo caso ogni job è caratterizzato da un ciclo tecnologico che richiede l'intervento di più macchine diverse; al contrario del flow shop, l'ordine con cui le operazioni vengono eseguite sulle diverse macchine è però diverso da job a job.

Flexible job shop (FJc) È una generalizzazione dei sistemi produttivi di job shop e di parallel machines. Invece di m macchine ci sono c centri di lavoro (work center) ognuno con un certo numero di identiche macchine in parallelo. Ogni job deve essere processato, seguendo un suo ordine ben preciso, in tutti i centri di lavoro e in ognuno di questi può essere lavorato su qualsiasi macchina.

Open shop (Om) Anche in questo caso ogni job è caratterizzato da un ciclo tecnologico che richiede l'intervento di più macchine diverse; tuttavia l'ordine con cui le operazioni vengono effettuate sulle diverse macchine può essere qualsiasi. La scelta del routing di ogni job può essere determinata quindi in base a considerazioni di tipo gestionale non essendo imposta da ragioni tecnologiche. Nell'open shop è previsto inoltre che un job j possa avere un processing time p_{ij} nullo su di una macchina i.

Parametro β

In questo campo possono venire specificati più valori scelti tra:

Release dates (r_j) Se questo simbolo appare nel campo β allora ogni job j non può essere lavorato prima della sua release date r_j. Al contrario le due dates non vengono specificate nel campo β, perché la funzione obbiettivo specificata in γ fornisce indicazioni sufficienti sul fatto che esse siano o meno considerate.

Sequence dependent setup times (s_{jk}) (Tempi di setup dipendenti dalla sequenza) Il simbolo s_{jk} rappresenta il tempo di setup dipendente dalla coppia di job adiacenti j e k; s_{0k} rappresenta il tempo di setup per il job k se esso è il primo nella sequenza dei lavori sulla macchina; s_{j0} rappresenta invece il tempo di pulizia (cleanup time) se il job j è l'ultimo nella sequenza dei job pianificati sulla macchina. Sia s_{0k} che s_{j0} possono essere nulli. Se il tempo di setup dipende oltre che dalla coppia di job j e k anche dalla macchina i su cui vengono lavorati allora si utilizza la notazione s_{ijk}. Se nel campo β non appare alcuna indicazione sui tempi di setup essi sono assunti nulli o indipendenti dalla sequenza dei lavori e in tal caso vengono considerati nel tempo di lavorazione p_{ij} (processing time).

Preemptions ($prmp$) Se la preemption è ammessa significa che è possibile l'interruzione della lavorazione di un job e una sua successiva ripresa dopo la lavorazione di altri job. Se il simbolo è assente significa che la preemption non è ammessa e dunque un job deve essere completato una volta che la sua lavorazione è stata iniziata.

Precedence constraints ($prec$) (Vincoli di precedenza) Nel caso questo simbolo venga specificato significa che esistono delle relazioni di precedenza tra i job. Può accadere infatti, in un contesto di single machine o parallel machines, che un job debba aspettare il completamento di altri job prima di essere iniziato a lavorare.

Breakdowns ($brkdwn$) Specificando questo simbolo si assume che le macchine non siano continuamente disponibili per le lavorazioni; il periodo durante il quale una macchina non è disponibile si assume fissato (ad esempio per una manutenzione programmata).

Machine eligibility restrictions (M_j) Questo simbolo può apparire come già detto solo nel contesto di parallel machines (Pm). Se presente esso indica che il job j può essere lavorato solo dal sottoinsieme M_j delle m macchine disponibili, in caso contrario esso può venire lavorato su qualsiasi macchina.

Permutation (*prmu*) Questo vincolo si applica solo nel contesto del flow shop. Se presente significa che non è ammesso il sorpasso (passing) tra i job; questo significa che su ogni macchina del flow shop viene mantenuta la stessa sequenza di job. In caso contrario è ammesso il sorpasso tra i job, e quindi la sequenza dei job sulle varie macchine del sistema non è la stessa.

Recirculation (*recrc*) Il fenomeno di recirculation può verificarsi nei contesti di flow shop e flexible flow shop quando un job può essere lavorato su di una macchina o in un centro di lavoro (work center) più di una volta durante il suo ciclo di produzione.

In alcuni casi ulteriori specificazioni possono essere utilizzate in questo campo come ad esempio la notazione $p_j = p$ ad indicare che tutti i tempi di lavorazione dei job sono uguali.

Parametro γ

Prima di vedere alcuni tipi di funzioni obbiettivo che vengono utilizzate nei problemi di scheduling è necessario definire alcuni parametri di prestazione:

Completion time (C_j) (Tempo di completamento) Rappresenta l'istante in cui il job j termina la sua lavorazione sull'ultima macchina in cui doveva essere processato.

Lateness (L_j) Questo parametro è definito come:

$$L_j = C_j - d_j$$

ed è positivo se la lavorazione del job j è completata in ritardo rispetto alla sua due date d_j e negativo se è completata in anticipo.

Tardiness (T_j) Il tardiness di un job j è definita come:

$$T_j = \max(C_j - d_j, 0) = \max(L_j, 0)$$

La differenza rispetto al lateness sta nel fatto che il tardiness non può mai essere negativo; esso non considera un vantaggio il fatto che un job sia in anticipo rispetto alla sua due date.

Earliness (E_j) È un parametro di scarso interesse così definito:

$$E_j = \min(L_j, 0)$$

Esso è pari al lateness se il job è in anticipo ed è nullo se è in ritardo.

Unit penalty (U_j) Esso è definito come:

$$U_j = \begin{cases} 1 & \text{se } C_j > d_j \\ 0 & \text{altrimenti} \end{cases}$$

Dopo l'introduzione di queste grandezze possiamo ora esaminare alcuni esempi di funzioni obbiettivo che devono essere **minimizzate**:

Makespan (C_{max}) Il makespan è definito come:

$$C_{max} = \max(C_1, \ldots, C_n)$$

cioè risulta equivalente al completion time dell'ultimo job in lavorazione. Minimizzare questo valore solitamente permette di ottenere un alto coefficiente di utilizzazione delle macchine.

Maximum Lateness (L_{max}) Esso è definita come:

$$L_{max} = \max(L_1, \ldots, L_n)$$

e misura quindi la peggiore violazione delle due date.

Total weighted completion time ($\sum w_j C_j$) La somma dei completion time degli n job è solitamente indicata in letteratura come *flow time*. Il total weighted completion time è quindi indicato anche come *weighted flow time*.

Total weighted tardiness ($\sum w_j T_j$) Questa è una funzione di costo più generale della ***total weighted completion time***, che rappresenta la somma pesata dei ritardi delle lavorazioni rispetto alle loro due date.

weighted number of tardy jobs ($\sum w_j U_j$) Questa funzione obbiettivo rappresenta il numero di lavorazioni in ritardo rispetto alla propria due date ed è spesso utilizzata nella pratica.

Total setup time ($\sum SU_i$) Rappresenta il tempo di setup complessivo ed è definita come:

$$\sum_{i=1}^{m} SU_i$$

dove SU_i è il tempo di setup sulla macchina i per lavorare l'insieme di job assegnato.

Spesso nel passato la ricerca si è concentrata soprattutto su modelli che cercavano di minimizzare funzioni costituite da un solo obbiettivo, mentre recentemente si è cominciato ad analizzare anche modelli caratterizzati da funzioni multiobbiettivo (*multiple objective functions*). Una notazione standard per queste funzioni non è ancora stata definita, tuttavia se indichiamo con γ_1 e γ_2 i due obbiettivi che vogliamo minimizzare, la funzioni complessiva si può indicare come $\theta_1\gamma_1 + \theta_2\gamma_2$ dove θ_1 e θ_2 rappresentano i pesi assegnati ai due obbiettivi.

Si può quindi ora rappresentare con la notazione appena introdotta il problema in esame.

2.2.2 Classificazione del problema in esame

Per quanto riguarda il sistema produttivo in esame (parametro α), ci troviamo nel contesto di macchine parallele identiche. Infatti la velocità con la quale i pezzi vengono stampati è dovuta alla durata del ciclo di stampaggio che è una caratteristica dello stampo e non della pressa su cui lo stampo viene montato. Quindi si ha:

$$\alpha = Pm \tag{2.1}$$

Le caratteristiche del sistema e i vincoli che esso deve rispettare (parametro β) si possono così riassumere:

- ***Sequence dependent setup times (s_{jk})***. I tempi di setup delle lavorazioni dipendono infatti, per i motivi già illustrati, dalla sequenza delle stesse.
- ***Machine eligibility restrictions (M_j)***. Ogni commessa può venire lavorata solo da un certo sottoinsieme delle 21 presse dell'azienda.

È stato necessario introdurre un ulteriore vincolo per rappresentare una caratteristica intrinseca del sistema in esame:

- ***Not contemporaneity (cont)***. Questo vincolo impedisce che due commesse che necessitano dello stesso stampo possano essere lavorate contemporaneamente su due presse distinte; questa possibilità non è ammessa poiché esiste un solo esemplare per ogni modello di stampo.

Inoltre specificheremo con l'utilizzo del simbolo $\boldsymbol{p_j}$ che il tempo di lavorazione dei job è indipendente dalla macchina sulla quale vengono eseguiti. In definitiva si ha:

$$\beta = p_j, s_{jk}, M_j, cont \tag{2.2}$$

Per quanto riguarda la scelta della funzione obbiettivo si è tenuto conto delle due esigenze fondamentali dell'azienda:

1. rispettare le date di consegna (due date) dei prodotti finiti ai clienti;
2. cercare di ridurre il più possibile i tempi delle operazioni di cambio-stampo sulle presse tra una lavorazione e l'altra (minimizzare i tempi di setup).

Per soddisfare entrambe le esigenze si è pensato di utilizzare una funzione multiobbiettivo che combina le funzioni di ***total weighted tardiness*** e di ***total setup time*** in un'unica espressione:

$$\gamma = \sum w_j T_j + \sum_{i=1}^{m} SU_i \tag{2.3}$$

Riassumendo, da 2.1, 2.2, 2.3, per caratterizzare il problema in esame si utilizza la notazione seguente:

$$Pm \Big| p_j, s_{jk}, M_j, cont \Big| \sum w_j T_j + \sum_{i=1}^{m} SU_i \tag{2.4}$$

2.3 La complessità computazionale

Un considerevole lavoro di ricerca nel campo dello scheduling è stato impiegato per trovare algoritmi di risoluzione efficienti, cioè con un tempo di esecuzione polinomiale rispetto alla taglia dell'istanza del problema, per molti dei modelli proposti in letteratura. Tuttavia durante tutti questi anni di studio molti problemi di scheduling sono stati classificati, nell'ambito della teoria della complessità computazionale, nella classe dei problemi NP-hard. Per questi problemi si pensa, sebbene la questione sia ancora aperta, che non possa esistere alcun algoritmo in grado di fornire una soluzione esatta in tempo polinomiale. Tra i problemi che sono stati dimostrati essere NP-hard vi è anche il problema $1|\sum w_j T_j$ (cfr. [13]). Si può dimostrare, utilizzando il concetto di riduzione polinomiale tra problemi (polynomial problem reduction), che questo problema è riducibile al problema in esame che quindi è a sua volta NP-hard. Basta considerare infatti un'istanza del problema in esame costituita da una sola macchina e con tempi di setup nulli per rappresentare una qualsiasi istanza del problema $1|\sum w_j T_j$.

2.4 Gli algoritmi euristici per lo scheduling

Per questi problemi (NP-hard) così detti "difficili", sono state studiate e sviluppate diverse procedure *euristiche* che possono essere adattate a una grande varietà di problemi di scheduling ed implementate abbastanza facilmente nei diversi sistemi produttivi industriali. Tutte queste tecniche non garantiscono di ottenere una *soluzione ottima* al problema, ma si prefiggono di trovare *buone soluzioni* in *tempi relativamente brevi*.

Ci occuperemo sostanzialmente di due tipologie di algoritmi euristici applicabili al problema dello scheduling:

- gli euristici costruttivi (constructive heuristic);
- gli euristici migliorativi (improvement heuristic).

Per quanto riguarda le procedure del primo tipo esse operano costruendo gradualmente una soluzione ammissibile (una pianificazione nel caso dello scheduling), cercando di contenere il costo della funzione obbiettivo della soluzione stessa; gli euristici migliorativi al contrario partono già da una soluzione (che può anche non essere ammissibile) e cercano, modificandola un po' ad ogni passo, di ottenere nuove soluzioni con un valore migliore della funzione obbiettivo. Una classe importante di algoritmi di questo tipo è rappresentata dalle procedure di ricerca locale (local search procedures) tra cui viene annoverata anche la tecnica del *Tabu Search*, che verrà ampiamente descritta in seguito (cfr. sezione 4.1). Le *regole di carico* (*dispatching rules*) appartengono invece agli euristici di tipo costruttivo e stabiliscono un ordine di priorità secondo un determinato criterio tra i vari job che devono ancora essere pianificati sulle macchine; ad ogni iterazione viene pianificato il job con la priorità maggiore. Le regole di carico appartengono perciò alla categoria degli algoritmi di tipo greedy, nei quali viene sempre fatta la scelta che sembra la migliore nel momento in cui si effettua. Questa strategia euristica può portare ad una soluzione ottima per alcuni problemi di scheduling particolarmente semplici, ma nella stragrande maggioranza dei problemi di scheduling che si incontrano nella realtà dei sistemi produttivi industriali questo approccio miopico non risulta particolarmente efficiente.

L'algoritmo che ho progettato sfrutta entrambi i tipi di euristiche presentati. Esso combina la tecnica del *GRASP* (Greedy Randomized Adaptive Search Procedures) con la tecnica Tabu Search, ed è strutturato concettualmente in due fasi:

Prima fase: la fase costruttiva della procedura GRASP viene utilizzata sfruttando la regola di carico ATCS (Apparent Tardiness Cost with Setups) per creare una pianificazione iniziale delle lavorazioni.

Seconda fase: la pianificazione sviluppata nella prima fase è usata come soluzione iniziale per una procedura Tabu Search che cerca di generare una pianificazione migliore.

Lo schema generale dell'algoritmo è rappresentato in fig. 2.2.

Nei prossimi due capitoli (capitolo 3, capitolo 4) analizzeremo in dettaglio le due fasi in cui è strutturato l'algoritmo, descrivendo le tecniche generali che vengono utilizzate e illustrando come esse vengono adattate al problema specifico. Il caricamento dei dati e il salvataggio della soluzione migliore trovata sono invece trattati nel capitolo 5 dove si discute l'implementazione software dell'algoritmo.

Figura 2.2: Schema complessivo dell'algoritmo

Capitolo 3

Prima fase: costruzione di una soluzione iniziale

Durante questa fase l'algoritmo crea, a partire dai dati sulle commesse da pianificare e sulle commesse attualmente in produzione sulle presse, una pianificazione iniziale sfruttando le caratteristiche del GRASP (cfr. [7]) e la regola di carico ATCS (cfr. [11]) che illustriamo nelle prossime sezioni. Dopo una descrizione generale di queste tecniche, vedremo come esse sono state adattate al problema specifico in esame.

3.1 La procedura GRASP

Il GRASP (Greedy Randomized Adaptive Search Procedures) è una procedura euristica utilizzata per risolvere *problemi di ottimizzazione combinatoria* (*combinatorial optimization problems*). Questo genere di problemi si incontra abbastanza frequentemente nel campo dell'industria e della scienza; per ognuno di essi è teoricamente possibile enumerare tutte le combinazioni di soluzioni (che sono in numero finito) e valutarle rispetto ad una funzione obbiettivo fissata. Tuttavia, da un punto di vista pratico, è impossibile seguire una tale strategia di completa enumerazione poiché il numero di combinazioni spesso cresce esponenzialmente nella taglia del problema. Negli ultimi 40 anni un notevole lavoro di ricerca è stato sviluppato per cercare degli algoritmi in grado di risolvere questi problemi e in alcuni casi (come quello della *programmazione lineare*) si sono ottenuti notevoli successi. Tuttavia come già detto alcuni problemi sono per loro natura "intrattabili", e in questi casi metodi euristici sono impiegati per cercare di trovare buone anche se non ottime soluzioni. L'efficacia di questi metodi dipende dalla loro abilità di adattarsi ad una particolare realizzazione, di evitare di rimanere intrappolati in punti di *ottimo locale* e di sfruttare la struttura base del problema. Molte procedure euristiche sono state sviluppate durante questi anni e si sono rivelate molto utili nell'ottenere buone soluzioni a problemi di ottimizzazione combinatoria anche particolarmente complessi. Le più promettenti di queste tecniche includono il *simulated annealing*, il *Tabu Search*, gli *algoritmi genetici* (*genetic algorithms*) e il *GRASP* appunto.

Il GRASP è una procedura iterativa in cui ogni iterazione è caratterizzata da due fasi:

- fase costruttiva;
- fase di ricerca locale.

La soluzione migliore trovata durante tutte le iterazioni viene mantenuta come risultato della procedura. Lo pseudocodice della procedura è questo:

Procedura GRASP	
Passo 1.	Carica i dati di ingresso del problema e inizializza la miglior soluzione trovata (BSF).
Passo 2.	Poni $S = CostructGreedyRandomizedSolution()$.
Passo 3.	Poni $S = LocalSearch(S)$.
Passo 4.	Se $S < BSF$ poni $BSF = S$.
Passo 5.	Se si verifica una condizione di termine **FINE**, altrimenti torna al passo 2.

Le condizioni di termine possono essere di diversa natura; la procedura può ad esempio terminare qualora sia stato raggiunto il numero massimo di iterazioni consentite, oppure sia stata trovata una soluzione il cui valore della funzione obbiettivo sia al di sotto di un valore di soglia prefissato.

Durante la fase costruttiva della procedura (Passo 2) una soluzione ammissibile viene costruita in maniera iterativa aggiungendo un elemento alla volta; ad ogni iterazione di questa fase la scelta dell'elemento che deve essere aggiunto è determinata ordinando, rispetto ad una funzione di tipo greedy, tutti gli elementi in una lista di "candidati". Questa funzione misura il beneficio (miopico) associato alla scelta di ciascun elemento. L'euristica è definita come "adaptive" (adattabile) perché i benefici associati ad ogni elemento sono aggiornati ad ogni iterazione della fase di costruzione per riflettere i cambiamenti avvenuti in seguito alla selezione dell'elemento nell'iterazione precedente. La componente probabilistica della procedura si manifesta nella scelta dell'elemento che ad ogni iterazione deve essere aggiunto per formare la soluzione; esso è infatti scelto casualmente (randomly) tra i migliori candidati nella lista e non è detto quindi che sia il migliore in assoluto. La lista contenente i migliori "candidati" che possono essere scelti viene indicata come RCL (restricted candidate list). Questa tecnica di scelta permette dunque di ottenere soluzioni diverse ad ogni iterazione della procedura GRASP. Lo pseudocodice della procedura CostructGreedyRandomizedSolution() è riportato di seguito:

Procedura ConstructGreedyRandomizedSolution()	
1	Solution = {}
2	**for** Solution construction not done →
3	$MakeRCL(RCL)$;
4	$s = SelectElementAtRandom(RCL)$;
5	$Solution = Solution \cup \{s\}$;
6	$AdaptGreedyFunction(s)$;
7	**rof;**
8	**return** Solution;

Tuttavia le soluzioni trovate dopo aver completato la fase costruttiva non è detto che siano localmente ottime; per questo motivo si applica una ricerca locale (Passo 3 della procedura GRASP) a queste soluzioni per esplorarne l'intorno cercando soluzioni migliori. Un algoritmo di ricerca locale sfrutta un procedimento iterativo in cui ad ogni iterazione la soluzione corrente S viene rimpiazzata dalla migliore soluzione nell'intorno $N(S)$ di quest'ultima; l'algoritmo termina quando non esiste nell'intorno una soluzione migliore della soluzione corrente; la soluzione così trovata è localmente ottima. Talvolta queste procedure di ricerca locale possono impiegare molto tempo, partendo da un'arbitraria soluzione iniziale, prima di trovare un ottimo locale; si è dimostrato sperimentalmente che la loro efficienza migliora notevolmente se migliora la soluzione iniziale da cui partono. Quindi un'implementazione accurata della fase costruttiva permette di ottenere buone soluzioni iniziali e di rendere quindi più efficiente la fase di ricerca locale. Lo pseudocodice della procedura LocalSearch(S) è riportato di seguito:

Procedura LocalSearch(S)		
1	**for**	S not locally optimal $\rightarrow$
2		Find a better solution $t \in N(S)$;
3		Let $S = t$
4	**rof;**	
5	**return** S;	

Una caratteristica molto interessante del GRASP è la facilità con cui questa tecnica può essere implementata. Infatti solo pochi parametri (la taglia della lista RCL e il numero di iterazioni) necessitano di essere impostati e calibrati e perciò lo sviluppo si può concentrare sull'implementazione di strutture dati efficienti per migliorare la velocità di un'iterazione.

Nell'algoritmo progettato, ho fatto uso della sola fase costruttiva della tecnica GRASP per ottenere una soluzione iniziale che viene poi utilizzata come soluzione di partenza dalla procedura di Tabu Search nella seconda fase dell'algoritmo. La funzione di tipo greedy utilizzata per l'ordinamento dei job nella fase costruttiva è un adattamento della regola di carico (dispatching rule) ATCS.

3.2 Le regole di carico

La filosofia delle regole di carico è semplicemente quella di scegliere, fra i job che attendono di essere lavorati da una macchina, quello da caricare sulla macchina stessa non appena questa si libera (cioè termina la lavorazione precedente). In pratica la regola permette di identificare, ogni volta che la macchina si libera, quel job che ha la priorità più elevata fra quelli in attesa (ciò può corrispondere al valore più alto o più basso dell'indice associato ad ogni job, a seconda della regola usata).

Non si possono formulare delle ipotesi generali di applicabilità di questo metodo, poiché esse variano da una regola all'altra e da un caso all'altro. Regole che forniscono prestazioni molto buone rispetto ad un criterio di misura (obbiettivo da raggiungere) risultano spesso meno buone rispetto ad altri criteri, per cui un primo fattore critico nella valutazione dei risultati ottenibili con l'applicazione di regole di carico sta nell'individuazione del parametro che la regola deve ottimizzare.

Le regole di carico si dimostrano spesso utili quando si sta cercando di trovare una buona pianificazione rispetto ad un solo obbiettivo come la minimizzazione del makespan, o del total completion time o del massimo lateness. Tuttavia spesso ci si trova di fronte ad obbiettivi più complessi che possono essere la combinazione di più obiettivi base. Per fronteggiare situazioni di questo tipo sono state elaborate *regole di carico composte* (*composite dispatching rules*) che ordinano i job secondo un indice ottenuto combinando un certo numero di *regole di carico elementari*.

Una regola di carico elementare è una funzione degli attributi dei job e/o delle macchine. Esempi di attributi dei job sono il peso (weight), il tempo di lavorazione (processing time), la data di consegna (due date), mentre tra gli attributi delle macchine ci sono la velocità (speed), il numero di lavori in attesa di essere processati e così via. Ogni regola di carico elementare all'interno di una regola composta è caratterizzata da un parametro (*scaling parameter*) che serve per calibrare opportunamente il contributo della regola elementare nell'espressione finale dell'indice di ordinamento. Questi parametri possono essere fissi e decisi da chi ha progettato la regola o possono essere calcolati ad ogni esecuzione in base a delle opportune statistiche relative agli attributi dei job e delle macchine. Le funzioni che legano i parametri alle statistiche sono determinate dal progettista della regola di carico in base alla propria esperienza e spesso dopo avere analizzato i risultati dei test delle simulazioni effettuate al computer. Ogni volta dunque che la regola di carico composta viene utilizzata per una pianificazione è necessario calcolare dapprima alcune statistiche proprie della particolare istanza considerata (cioè degli attributi

dei job e delle macchine), utilizzare queste statistiche per impostare i valori dei parametri per le varie regole di carico elementari e solo a questo punto applicare effettivamente la regola all'insieme dei job considerati.

La regola di carico composta che ho deciso di utilizzare nella prima fase dell'algoritmo è una versione adattata al contesto produttivo di macchine parallele della regola ATCS (Apparent Tardiness Cost with Setups).

3.2.1 La regola ATCS

La regola di carico ATCS è una generalizzazione della regola ATC che meglio si adatta a quei sistemi in cui i tempi di setup sono dipendenti dalla sequenza con la quale i job vengono pianificati sulle macchine. Essa è stata progettata per il problema $1|s_{jk}|\sum w_j T_j$ in cui l'obbiettivo è quello di minimizzare il weighted tardiness in un contesto produttivo di macchina singola con tempi di setup dipendenti dalla sequenza delle lavorazioni. La regola ATCS combina tre regole di carico elementari:

1. la regola WSPT (Weighted Shortest Processing Time first) secondo cui i job vengono ordinati in ordine decrescente del valore di w_j/p_j;
2. la regola MS (Minimum Slack first) secondo cui quando una macchina si libera al tempo t, viene selezionato per il caricamento il job con il valore minimo dell'espressione $\max(d_j - p_j - t, 0)$;
3. la regola SST (Shortest Setup Time first) secondo cui ogni volta che una macchina si libera dopo aver completato l'esecuzione di una lavorazione, si seleziona per il caricamento il job con il più piccolo tempo di setup.

Tutte queste tre regole vengono combinate in un'unica espressione che permette di calcolare un indice in base al quale ordinare i job. L'espressione che permette di calcolare l'indice del job j al tempo t quando il job l ha terminato la sua lavorazione sulla macchina è questa:

$$I_j(t,l) = \frac{w_j}{p_j} exp\left(-\frac{\max(d_j - p_j - t, 0)}{K_1 \overline{p}}\right) exp\left(-\frac{s_{lj}}{K_2 \overline{s}}\right) \tag{3.1}$$

In 3.1 $\overline{p}$ rappresenta la media dei processing time (p_j) dei job, $\overline{s}$ rappresenta la media dei setup time dei job mentre K_1 e K_2 rappresentano rispettivamente lo scaling parameter per le due date e per i setup time. Come detto i parametri K_1 e K_2 possono essere fissati a priori oppure possono essere legati al valore di alcune statistiche che caratterizzano ogni istanza del problema. Tra le statistiche più utilizzate vi sono:

- *due date tightness factor* τ definito come:

$$\tau = 1 - \frac{\sum d_j}{nC_{max}} \tag{3.2}$$

  Valori di τ prossimi a 1 indicano che le due date sono abbastanza "rigide" mentre valori prossimi a 0 indicano il contrario.

- *Range factor* R definito come

$$R = \frac{d_{max} - d_{min}}{C_{max}} \tag{3.3}$$

  Un elevato valore di questo parametro indica un ampia estensione dell'intervallo delle due date.

- *setup time severity factor* η definito come:

$$\eta = \frac{\overline{s}}{\overline{p}} \tag{3.4}$$

Per quanto riguarda il valore del makespan C_{max} esso deve essere chiaramente stimato; nel contesto di m macchine parallele ed n job si può utilizzare la seguente approssimazione:

$$C_{max} = \frac{\sum_{j=1}^{n} p_j + n\overline{s}}{m} \tag{3.5}$$

Uno studio sperimentale della regola ATCS (cfr. [11]) suggerisce di utilizzare il range factor R definito in 3.3 per calcolare il valore di K_1 come:

$$K_1 = \begin{cases} 4.5 + R & \text{se } R < 0.5 \\ 6 - 2R & \text{se } R \geqslant 0.5 \end{cases} \tag{3.6}$$

mentre le statistiche in 3.2 e in 3.4 sono utilizzate per calcolare il valore di K_2 come:

$$K_2 = \frac{\tau}{2 \cdot \sqrt{\eta}} \tag{3.7}$$

3.3 Applicazione al problema in esame

Come già detto in precedenza l'algoritmo sviluppato sfrutta solamente la fase costruttiva della procedura GRASP per costruire una soluzione iniziale al problema che verrà quindi migliorata con l'applicazione della tecnica di Tabu Search. La funzione di tipo greedy utilizzata per costruire la pianificazione iniziale si basa sulla regola di carico ATCS appena descritta. Per adattare questa regola di carico al contesto di macchine parallele del problema in esame ho deciso di selezionare di volta in volta la macchina che termina per prima l'esecuzione del suo ultimo job pianificato; a questo punto i job che possono essere lavorati sulla pressa sono ordinati attraverso l'applicazione della formula 3.1 e viene così costruita la lista RCL dei job candidati ad essere lavorati sulla pressa. Da questa lista viene quindi selezionato casualmente (randomly) il job che deve essere pianificato sulla pressa. Dopo aver calcolato la nuova fine prevista dell'ultima lavorazione sulla macchina, il procedimento riparte. In questo modo si garantisce che il carico lavorativo risulti equamente distribuito sulle varie presse. Lo pseudocodice è dunque questo:

Prima fase dell'algoritmo	
Passo 1.	Calcola le statistiche τ, R, η come definite in 3.2, 3.3, 3.4 e usa i valori trovati per calcolare gli scaling parameter K_1 e K_2 secondo 3.6, 3.7.
Passo 2.	Crea la lista dei job da pianificare (LJP) e la coda prioritaria delle presse (CP).
Passo 3.	Seleziona ed estrai la pressa m_i in testa alla coda CP.
Passo 4.	Se non ci sono job in LJP che possono essere lavorati su m_i torna al passo 3.
Passo 5.	Costruisci la lista dei "candidati" RCL ordinando i job in LJP che possono essere lavorati su m_i secondo l'equazione 3.1.
Passo 6.	Seleziona casualmente (randomly) in RCL il job j che si vuole pianificare su m_i.
Passo 7.	Pianifica il job j su m_i calcolando la nuova fine prevista dell'ultima lavorazione su m_i.
Passo 8.	Reinserisci la pressa m_i nella coda CP con la nuova priorità calcolata.
Passo 9.	Elimina il job j da LJP.
Passo 10.	Se LJP è vuota **FINE**, altrimenti torna al passo 3.

Nella coda prioritaria CP le presse sono ordinate in modo decrescente in base all'istante in cui termina l'ultima lavorazione pianificata sulla macchina. Quando la coda viene creata si fa riferimento all'istante in cui termina la lavorazione attualmente in produzione sulla pressa. La prima fase dell'algoritmo termina quando non vi sono più job nella lista LJP; a questo punto tutti i job sono stati pianificati su qualche pressa e abbiamo quindi ottenuto una soluzione iniziale al problema con un valore della funzione obbiettivo che indicheremo con FOSI (Funzione Obbiettivo Soluzione Iniziale).

Vedremo nel prossimo capitolo come questa soluzione venga migliorata con l'applicazione della tecnica di Tabu Search.

Capitolo 4

Seconda fase: miglioramento della soluzione iniziale

In questa fase dell'algoritmo viene applicata la tecnica euristica nota come *Tabu Search* partendo dalla soluzione iniziale costruita durante la prima fase. Dopo un'esauriente descrizione di quest'euristica vedremo come essa è stata applicata al problema in esame.

4.1 Tabu Search

4.1.1 Introduzione

Nel campo scientifico ed ingegneristico esistono molti problemi di ottimizzazione intrinsecamente complessi, nei quali la ricerca della *soluzione ottima* mediante metodi esatti risulta improponibile, in quanto non sono noti algoritmi capaci di ottenerla senza impiegare un tempo di calcolo eccessivo. Diventa quindi fondamentale l'applicazione di *algoritmi euristici* che consentono, nella maggioranza dei casi, di ottenere soluzioni accettabili in tempi ragionevoli.

La *Tabu Search* è una metodologia euristica molto flessibile, che può essere applicata a un'estesa classe di problemi. Ciò che la contraddistingue è la capacità di superare i confini, generalmente trattati come barriere, imposti dall'ammissibilità di una soluzione o dall'ottimalità locale, e di permettere (mediante la rimozione temporanea di alcuni vincoli o l'accettazione di soluzioni intermedie peggiorative) una valida esplorazione dello spazio delle soluzioni.

La sistemazione teorica della Tabu Search e la sua divulgazione nella forma con cui è conosciuta oggi è attribuibile principalmente al lavoro di *Fred Glover* (cfr. [8], [9]), dell'Università di Colorado, a partire dal 1986. Da allora essa è stata applicata con successo a numerosi problemi di ottimizzazione in diversi campi, come ad esempio la minimizzazione quadratica, lo *scheduling*, l'instradamento ottimo, la colorazione dei grafi. Al momento, non esiste alcuna prova formale (se non per alcune forme probabilistiche) della convergenza dell'algoritmo di Tabu Search.

Da un punto di vista formale, un *problema di ottimizzazione* può essere posto in questi termini: dato un insieme S di soluzioni ammissibili ed una funzione obiettivo $f : S \mapsto \mathbb{R}$, trovare una soluzione i^* tale che $f(i^*)$ sia ottima rispetto a qualche criterio; generalmente si richiede che $f(i^*) \leq f(i) \quad \forall i \in S$. La Tabu Search, al pari del Simulated Annealing, è una *procedura iterativa* basata sulla *Ricerca nell'Intorno* di una soluzione (*Neighborhood Search*). Il passo generale di una procedura iterativa consiste nel costruire una nuova soluzione j a partire dalla soluzione corrente i e valutare se fermarsi o attuare un nuovo passo. I metodi di ricerca in un intorno (Neighborhood Search) sono un sottoinsieme delle tecniche iterative in cui si definisce, per ogni soluzione ammissibile i, un insieme $N(i)$ (l'intorno della soluzione i) a cui deve appartenere la nuova soluzione j. Di questa classe di tecniche fa parte il famoso *algoritmo*

di discesa (*Descent Method*) per la determinazione di un'approssimazione di un minimo di una funzione reale f definita su un insieme S :

Algoritmo di discesa	
Passo 1.	Scegli una soluzione iniziale $i \in S$.
Passo 2.	Genera un sottoinsieme $V^* \subseteq N(i)$.
Passo 3.	Trova la miglior soluzione $j \in V^*$ tale che $f(j) \leq f(k) \quad \forall k \in V^*$.
Passo 4.	Se $f(j) \geq f(i)$ allora STOP; altrimenti poni $i = j$ e torna al Passo 2.

Il sottoinsieme V^* può coincidere con l'intorno $N(i)$; tuttavia spesso risulta molto complesso (soprattutto in termini di tempo) calcolare l'intero intorno di una soluzione e un'appropriata scelta di V^* consente un notevole miglioramento delle prestazioni.

La Tabu Search è considerato un metodo di ricerca in un intorno (Neighborhood Search) più elaborato del semplice algoritmo di discesa. Si analizzano ora più in dettaglio le caratteristiche di questa tecnica.

4.1.2 Caratteristiche generali

La Tabu Search (TS) è una tecnica di discesa che prevede un uso efficiente della "memoria" per uscire da minimi locali alla ricerca di soluzioni migliori. A questo proposito viene tenuta traccia non solo delle informazioni locali (come il valore corrente della funzione obbiettivo) ma anche di parametri che hanno caratterizzato il processo di esplorazione, allo scopo di restringere la scelta della prossima soluzione j da esplorare ad un sottoinsieme di $N(i)$. Più precisamente la struttura dell'intorno $N(i)$ varia da iterazione a iterazione e per questo motivo, l'euristica è stata inserita fra le tecniche a ricerca dinamica in un intorno (*Dynamic Neighborhood Search*). Il ruolo della "memoria" è quello di aiutare a definire un insieme

$$N^*(i) \subseteq N(i) \tag{4.1}$$

detto intorno ridotto della soluzione i, entro il quale scegliere la soluzione successiva j, tenendo conto della "storia" degli stati attraversati durante la ricerca.

La semplice tecnica di discesa va incontro infatti al rischio di rimanere "intrappolata" in un punto di minimo locale, che potrebbe essere anche molto distante dal punto di minimo globale. Un metodo per risolvere questo problema è quello di accettare, in alcune circostanze, delle mosse peggiorative da una soluzione i ad una soluzione $j \in N(i)$. Questo accorgimento ha tuttavia lo svantaggio di rendere possibile che l'algoritmo cicli su un numero finito di soluzioni. Si supponga infatti che, al passo k-esimo, la soluzione corrente sia i, e non esista alcuna soluzione $j \in N(i)$ tale che $f(j) \leq f(i)$. Accettando una mossa peggiorativa, l'algoritmo sceglie come soluzione corrente (in base ad un qualche criterio) una soluzione p tale che $f(p) > f(i)$. Al passo $(k+1)$-esimo, se la "vecchia" soluzione i appartiene all'intorno di p, e se non esiste alcuna soluzione $q \in N(p)$ tale che $f(q) \leq f(i) < f(p)$, allora l'algoritmo individua i come soluzione migliorativa e compie un ciclo, riportandosi alla situazione del passo k-esimo.

Per ridurre la probabilità di compiere dei cicli, è necessario l'impiego da parte dell'algoritmo delle informazioni relative al "passato", ovvero relative al percorso seguito per giungere alla soluzione corrente. Queste informazioni possono essere usate per proibire a priori la valutazione da parte dell'algoritmo delle soluzioni in $N(i)$ visitate nei passi precedenti. La composizione dell'intorno di una soluzione viene quindi a dipendere dall'iterazione k, oltre che dalla soluzione i. D'ora in avanti, perciò, l'intorno di una soluzione verrà indicato con la notazione

$$N(i, k). \tag{4.2}$$

Formalizzando i concetti introdotti, si può riscrivere la generica iterazione dell'algoritmo di discesa modificandola in questo modo:

Algoritmo di discesa modificato	
Passo 1.	Scegli una soluzione iniziale $i \in S$. Poni $i^* = i$ e $k = 0$.
Passo 2.	Poni $k = k + 1$ e genera il sottoinsieme $V^* \subseteq N(i, k)$.
Passo 3.	Scegli il miglior $j \in V^*$ rispetto ad f oppure ad $\widetilde{f}$.
Passo 4.	Se $f(j) \leq f(i^*)$ allora poni $i^* = j$.
Passo 5.	Fermati se si verifica una condizione di termine, altrimenti poni $i = j$ e torna al passo 2.

Per condizione di termine si intende una clausola che, se verificata, determina la fine del processo di ricerca, come ad esempio:

- $N(i, k + 1) = \emptyset$;
- k è maggiore del massimo numero di iterazioni consentite;
- il numero di iterazioni compiute dall'ultimo miglioramento di i^* è maggiore di un numero prestabilito;
- si può provare di aver ottenuto una soluzione ottima.

Rimane da chiarire in che modo la conoscenza dell'itinerario percorso per arrivare alla soluzione corrente possa essere utilizzata convenientemente per modificare la strutture di $N(i, k)$. Le soluzioni recentemente visitate vengono escluse dall'intorno della soluzione corrente; tale accorgimento ha l'effetto di impedire l'instaurarsi di cicli. Quando una soluzione viene eliminata dall'insieme delle soluzioni candidate, si dice che essa è una soluzione "tabù" (*tabu solution*). Si definisce *lista tabù* (*tabu list*) l'insieme T delle ultime soluzioni esplorate. Si ha allora:

$$N(i, k) = N(i) \setminus T \tag{4.3}$$

È chiaro che il mantenimento di una lista tabù contenente le ultime $|T|$ soluzioni esplorate previene la formazione di cicli di lunghezza non superiore a $|T|$.

La gestione di una lista di soluzioni può essere molto dispendiosa dal punto di vista delle risorse di calcolo (memoria utilizzata, tempo impiegato), soprattutto nei casi in cui le soluzioni stesse siano oggetti molto complessi.

Si preferisce allora descrivere il processo di esplorazione delle soluzioni in termini delle *"mosse"* effettuate per passare da una soluzione ad un'altra. Con il termine generico di *mossa* si intende l'insieme delle operazioni elementari di modifica da compiere per passare da una soluzione ad un'altra.

Formalmente, si definisce $M(i)$ l'insieme delle mosse m che possono essere applicate ad i per ottenere una nuova soluzione j, e si usa la notazione $j = i \oplus m$. Da ciò discende che $N(i) = \{j \mid \exists m \in M(i) \text{ con } j = i \oplus m\}$. In generale le mosse sono reversibili: per ogni mossa m esiste una mossa m^{-1} tale che $(i \oplus m) \oplus m^{-1} = i$.

Così, invece di tenere traccia delle ultime $|T|$ soluzioni esplorate, è sufficiente memorizzare le ultime $|T|$ mosse effettuate, o le ultime $|T|$ mosse inverse associate a quelle effettivamente compiute. È chiaro che questa restrizione comporta una perdita di informazione, e non garantisce quindi a priori che non si possano instaurare cicli di lunghezza minore o uguale a $|T|$.

Per efficienza, può essere conveniente usare diverse liste tabù T_r allo stesso tempo; un parametro t_r di una soluzione i o di una mossa m può avere un valore tale da impedire una particolare mossa. Si possono quindi formulare le *condizioni tabù* come segue:

$$t_r(i, m) \in T_r \quad r \in \{1, \ldots, t\} \tag{4.4}$$

Una mossa è definita tabù se tutte le condizioni tabù sono soddisfatte.

4.1.3 Criterio di aspirazione

La perdita di informazione causata dalla semplificazione adottata (il fatto cioè di considerare le mosse invece che le soluzioni) determina anche un altro svantaggio: può succedere in alcuni casi che una mossa m considerata tabù e quindi non applicabile secondo quanto detto finora, porti ad una soluzione j in realtà non ancora esplorata e per la quale si potrebbe verificare che:

$$f(j) < f(i^*) \tag{4.5}$$

Le restrizioni introdotte dall'interdizione di alcune mosse renderebbero dunque irraggiungibile questa soluzione.È necessario quindi introdurre un criterio che, in questi casi, consenta il rilassamento dei vincoli imposti dalla lista tabù.

Per ogni mossa tabù m che, applicata alla soluzione corrente i, dia luogo ad una soluzione j migliorativa, viene allora definito il *livello di aspirazione* $a(i, m)$ e si impone che la mossa tabù possa essere effettuata quando il valore di tale livello è maggiore di una soglia $A(i, m)$ assegnata. Solitamente $A(i, m)$ può essere considerato un insieme di valori privilegiati per una funzione $a(i, m)$. Le condizioni d'aspirazione si possono esprimere dunque come:

$$a_r(i, m) \in A_r(i, m) \quad r \in \{1, \ldots, a\} \tag{4.6}$$

Se almeno una di queste condizioni è soddisfatta da una mossa tabù m applicata alla soluzione corrente i, la mossa verrà eseguita ugualmente nonostante la sua condizione tabù.

4.1.4 Intensificazione e Diversificazione

Nei paragrafi precedenti è stato illustrato come le informazioni sulle soluzioni esplorate nelle ultime $|T|$ iterazioni possano essere utilizzate per "proibire" l'effettuazione di mosse che potrebbero condurre la ricerca a soluzioni già visitate, instaurando un ciclo. Si parla in questo caso di uso della memoria per l'orientamento a corto termine (*short term memory*).

La conoscenza delle informazioni relative alle mosse precedenti consente inoltre di guidare il processo di ricerca anche in altri modi. Può infatti essere vantaggioso, in alcune circostanze, intensificare la ricerca in alcune regioni nelle quali è stata rilevata un'elevata concentrazione di punti di minimo locale. Questo obiettivo può essere raggiunto assegnando un'alta priorità alle soluzioni che presentano delle caratteristiche in comune con la soluzione corrente. È necessario allora introdurre nella funzione obiettivo un termine aggiuntivo f_i che penalizzi le soluzioni "lontane" da quella corrente, e che abbia valore diverso da zero solo per quelle iterazioni per cui si ritiene opportuno effettuare una fase di *intensificazione*. In altre circostanze, tipicamente al termine di una fase di intensificazione, può risultare vantaggioso guidare la ricerca verso zone inesplorate dello spazio delle soluzioni. La *diversificazione* può essere forzata introducendo nella funzione obiettivo un termine aggiuntivo f_d che penalizzi le soluzioni che siano troppo "vicine", cioè simili a quella corrente; la funzione obbiettivo così modificata verrà mantenuta per un numero di iterazioni sufficienti a spostarsi in una diversa regione delle soluzioni ammissibili.

La funzione obiettivo modificata viene indicata con $\widetilde{f}$ e si ha per essa:

$$\widetilde{f} = f + f_i + f_d \tag{4.7}$$

dove ai termini aggiuntivi f_i ed f_d sono associati opportuni pesi variabili che permettono all'algoritmo di alternare fasi di intensificazione a fasi di diversificazione.

Relativamente a queste due modalità di ricerca, si parla talora di uso della memoria per l'orientamento a lungo termine (*long term memory*).

Possiamo dunque riassumere le caratteristiche ora esposte della procedura di Tabu Search:

Tabu Search	
Passo 1.	Scegli una soluzione iniziale $i \in S$. Poni $i^* = i$ e $k = 0$.
Passo 2.	Poni $k = k+1$ e genera un sottoinsieme di soluzioni $V^* \subseteq N(i,k)$ tale che $\forall j = (i \oplus m) \in V^*$ almeno una delle condizioni tabù $t_r(i,m) \in T_r \quad r \in \{1,\ldots,t\}$ sia violata oppure valga almeno una tra le condizioni d'aspirazione $a_r(i,m) \in A_r(i,m) \quad r \in \{1,\ldots,a\}$.
Passo 3.	Scegli il miglior $j = (i \oplus m) \in V^*$ rispetto ad f o ad $\widetilde{f}$.
Passo 4.	Se $f(j) \leq f(i^*)$ allora poni $i^* = j$.
Passo 5.	Aggiorna le condizioni tabù e di aspirazione.
Passo 6.	Fermati se si verifica una condizione di termine, altrimenti poni $i = j$ e torna al passo 2.

4.1.5 Reactive Tabu Search

La ricerca reattiva (*Reactive Search*), proposta da R. Battiti in [2] e [3], propone l'introduzione di schemi di retroazione (*feedback schemes*) nelle euristiche per l'ottimizzazione discreta. Si tratta di applicare una semplice forma di apprendimento con rinforzo al sistema, in modo che i parametri di ricerca possano dinamicamente modificarsi a seguito dell'influenza del comportamento del sistema negli eventi passati; si introduce, quindi, il concetto di esperienza nella ricerca. Spesso, infatti, i vari parametri (costi, lunghezza delle liste, ecc...) possono variare secondo uno schema indipendente dall'istanza trattata; il che comporta che soluzioni accettabili possano essere ottenute dopo una accurata taratura dei valori tramite procedure "trial and error" (per tentativi). Risulta evidente, quindi, l'opportunità di utilizzare tecniche a parametri adattativi (*self tuning*) , per garantire una sufficiente flessibilità del metodo.

Si prenda, ad esempio, la lunghezza della lista tabù. Più grande è T, più lunga sarà la "traiettoria" che l'euristica dovrà compiere prima di permettere che una soluzione possa essere ripresa in considerazione; al contrario se la dimensione di T è troppo piccola è più probabile incappare in un ciclo. Aumentare la lunghezza di T quindi è un modo valido per attuare la fase di diversificazione, mentre diminuirla può favorire la fase di intensificazione. Tuttavia, non si può far crescere troppo T, in quanto potrebbe verificarsi il caso in cui nessun'altra soluzione sia possibile. La variazione adattativa del parametro T può quindi permettere notevoli miglioramenti della tecnica complessiva.

4.2 Applicazione al problema in esame

In questa sezione esaminiamo come viene applicata l'euristica Tabu Search nell'algoritmo sviluppato. Per prima cosa definiamo l'*intorno di una soluzione* e il corrispettivo concetto di *mossa*. Abbiamo infatti visto che data una soluzione i l'intorno $N(i)$ di questa soluzione è definito come:

$$N(i) = \{j \mid \exists m \in M(i) \text{ con } j = i \oplus m\} \tag{4.8}$$

dove $M(i)$ rappresenta l'insieme delle mosse applicabili ad i per ottenere una nuova soluzione j.

Nel nostro problema, una soluzione rappresenta una pianificazione delle commesse sulle presse; una mossa m è definita come lo spostamento di una commessa c pianificata su di una pressa p in un'altra posizione sulla stessa pressa p o su di un'altra pressa q sulla quale la commessa può essere lavorata. In fig. 4.1 è rappresentata la configurazione delle commesse pianificate sulle presse prima dell'esecuzione della mossa (le commesse sono rappresentate da una lista collegata alla pressa sulla quale sono state pianificate); in fig. 4.2 viene rappresentata la nuova configurazione delle commesse, dopo lo spostamento della commessa evidenziata dalla pressa "mac01" alla pressa "mac03".

L'intorno $N(i)$ di una soluzione i è dunque rappresentato da tutte le soluzioni j ottenibili a partire dalla pianificazione i applicando una mossa m ammissibile. Una mossa m è ammissibile se essa rispetta il vincolo di ***machine eligibility restrictions*** $\boldsymbol{M_j}$ definito nella sezione 2.2.1 (cioè la nuova pressa sulla quale la commessa spostata viene pianificata appartiene al sottoinsieme delle presse su cui la commessa può essere lavorata) e il vincolo di ***not contemporaneity (cont)*** definito nella sezione 2.2.2 (non ci sono cioè, nella configurazione finale delle commesse, sovrapposizioni tra gli intervalli di utilizzo degli stampi).

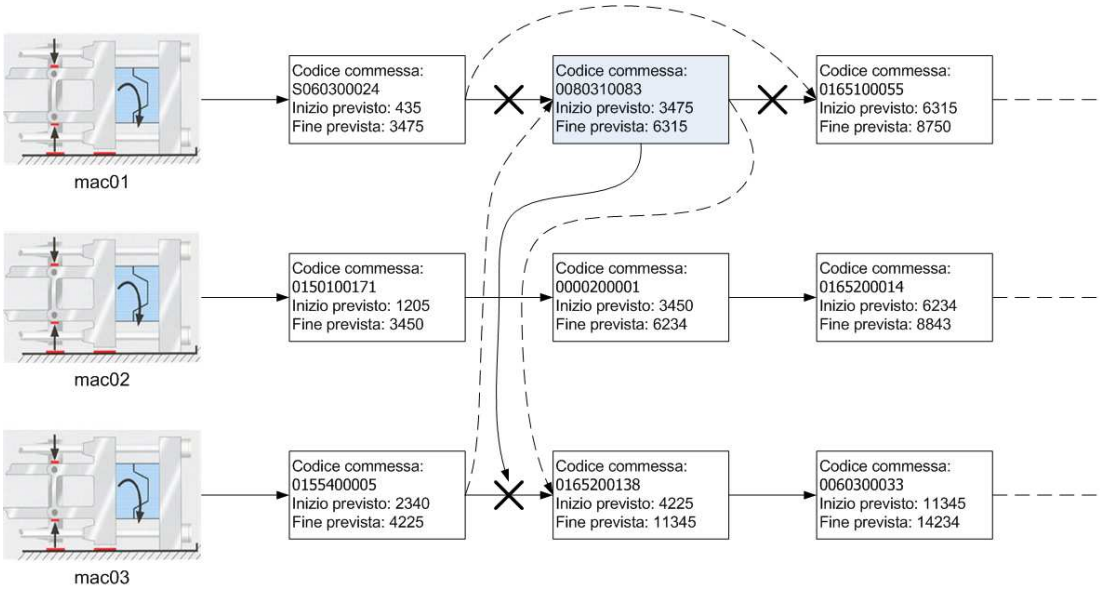

Figura 4.1: Pianificazione delle commesse prima della mossa.

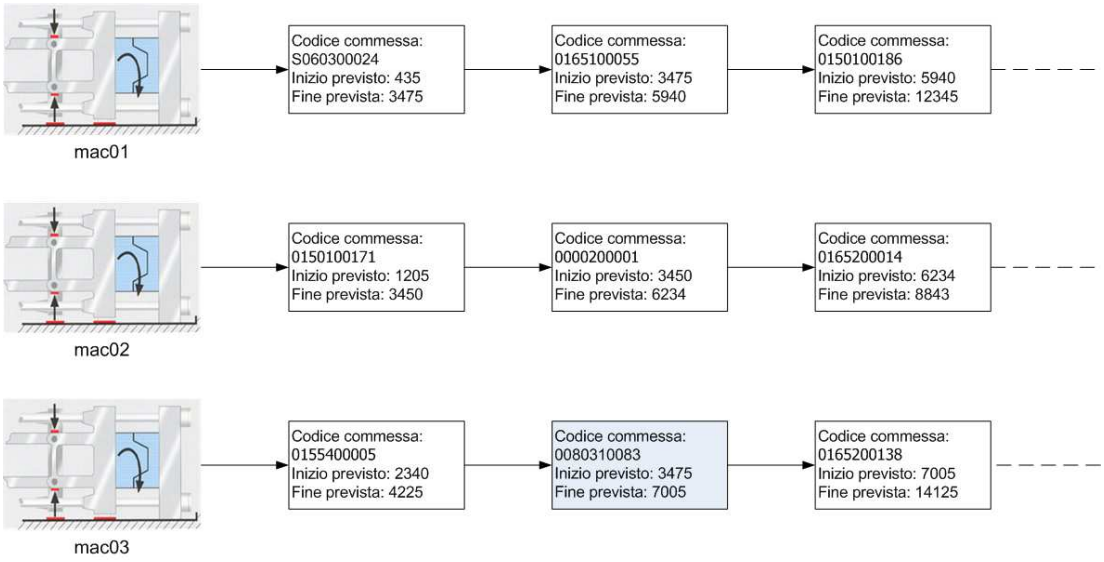

Figura 4.2: Nuova pianificazione in seguito all'esecuzione della mossa.

Ogni mossa m che porta da una pianificazione i ad una nuova pianificazione j è caratterizzata

da:

- la commessa c che viene spostata;
- la pressa vm sulla quale c era pianificata;
- la nuova pressa nm sulla quale c viene pianificata;
- la commessa vp che precedeva c su vm;
- la commessa np che precede c su nm.

Ogni mossa verrà dunque memorizzata nella lista tabù delle mosse salvando questi cinque parametri che la caratterizzano.

L'algoritmo alterna continuamente per un certo lasso di tempo fasi di intensificazione a fasi di diversificazione dell'euristica Tabu Search. Durante la fase di intensificazione viene utilizzata la funzione obbiettivo reale

$$\widetilde{f} = f = \sum w_j T_j + \sum_{i=1}^{m} SU_i \tag{4.9}$$

per ordinare tutte le mosse che servono a costruire l'intorno $N(i)$ della soluzione i; le mosse vengono cioè ordinate in maniera decrescente secondo il valore della funzione obbiettivo della soluzione $j = i \oplus m$ associata alla mossa m.

L'algoritmo utilizza due diversi modi per implementare la fase di diversificazione, alternando dopo una fase di intensificazione ora l'uno ora l'altro. Nel primo di questi due modi le mosse vengono ordinate secondo la funzione modificata

$$\widetilde{f} = \sum_{i=1}^{m} SU_i \tag{4.10}$$

che tiene conto solamente dei tempi di setup tra lavorazioni successive nella pianificazione e non dei valori del *tardiness* delle commesse; nel secondo modo invece le mosse vengono ordinate in maniera completamente casuale. Entrambe questi approcci garantiscono comunque di spostarsi in una regione dello spazio delle soluzioni ammissibili diversa da quella esplorata nella fase di intensificazione.

Lo pseudocodice della seconda fase dell'algoritmo è riportato qui di seguito. Diamo prima una spiegazione degli acronimi utilizzati.

- SC rappresenta la soluzione corrente e $FOSC$ è il valore della funzione obbiettivo f di SC;
- SI rappresenta la soluzione iniziale costruita durante la prima fase dell'algoritmo e $FOSI$ è il valore della funzione obbiettivo di SI;
- BSF rappresenta la miglior soluzione trovata dall'algoritmo e $FOBSF$ il valore della sua funzione obbiettivo;
- $FASE$ è l'indicatore della fase in cui si trova l'algoritmo: può assumere il valore I se ci troviamo nella fase di intensificazione o il valore D se siamo nella fase di diversificazione;
- N è il numero di mosse totali per ogni fase: può assumere il valore NI che indica il numero di mosse per una fase di intensificazione o il valore ND che indica il numero di mosse di una fase di diversificazione;
- n è il contatore delle mosse eseguite; è azzerato all'inizio di ogni fase (intensificazione, diversificazione);

- LT è la lista delle mosse tabù; le mosse vengono inserite ogni volta dalla testa e la mossa "più vecchia" viene estratta dalla coda quando la taglia della lista ($Size(LT)$) supera la dimensione $MaxSize$.
- CM rappresenta la coda delle mosse;
- FOm è il valore della funzione obbiettivo obbiettivo della soluzione che si ottiene a partire da SC applicando la mossa m;
- CT è il cronometro utilizzato per misurare la durata di questa fase dell'algoritmo; essa termina quando CT sorpassa la durata massima consentita $Maxtime$.

Seconda fase dell'algoritmo	
Passo 1.	Poni $SC = SI$, $BSF = SI$, $FOBSF = FOSI$, $FOSC = FOSI$, $FASE = I$, $N = NI$, $n = 0$, $LT = listavuota$, $CM = codavuota$. Avvia il cronometro CT.
Passo 2.	Calcola tutte le mosse ammissibili per SC e ordinale nella coda CM secondo la funzione $\widetilde{f}$ associata al valore di $FASE$.
Passo 3.	Estrai la mossa m in testa a CM.
Passo 4.	Se $((m^{-1} \in LT)And(FOm \geq FOBSF))$ torna la Passo 3.
Passo 5.	Se $((m^{-1} \in LT)And(FOm < FOBSF))$ esegui la mossa, poni $n = n + 1$, $SC = SC \oplus m$, $BSF = SC$, $FOBSF = FOSC$ e vai al Passo 9.
Passo 6.	Esegui la mossa m, poni $n = n + 1$, $SC = SC \oplus m$.
Passo 7.	Se $FOSC < FOBSF$, poni $BSF = SC$ e $FOBSF = FOSC$.
Passo 8.	Aggiungi m in testa a LT. Se $Size(LT) > MaxSize$ elimina la mossa in coda a LT.
Passo 9.	Svuota la coda CM.
Passo 10.	Se $n < N$ torna al Passo 2.
Passo 11.	Se $FASE = I$ poni $FASE = D$ e $N = ND$, altrimenti poni $FASE = I$ e $N = NI$. Poni $n = 0$.
Passo 12.	Se $CT > Maxtime$ allora **FINE**, altrimenti torna al passo 2.

Come si vede dallo pseudocodice, la condizione di termine della fase di applicazione del Tabu Search è temporale; alla fine di ogni fase di intensificazione o diversificazione si controlla se è trascorso il tempo massimo previsto per la ricerca di una soluzione migliore e in caso affermativo l'algoritmo termina e la soluzione migliore trovata è salvata in BSF.

Per quanto concerne il criterio di aspirazione (cfr. sezione 4.1.3), come si vede esso è stato implementato nella sua forma più immediata: una mossa tabù viene ugualmente eseguita dall'algoritmo se il valore della funzione obbiettivo della soluzione che si ottiene applicando la mossa (FOm) è minore del valore della funzione obbiettivo della migliore soluzione trovata (FOBSF).

Abbiamo inoltre utilizzato una lista tabù LT con $|LT|$ costante al variare dell'iterazione k; implementazioni più complesse suggeriscono di variare anche tale parametro a seconda dello stato attuale e della "storia" recente, applicando in tale modo le fasi di diversificazione e intensificazione. A tale tecnica ci si riferisce come Reactive Tabu Search (cfr. 4.1.5).

Capitolo 5

Implementazione

Per l'implementazione dell'algoritmo euristico descritto mi sono avvalso del linguaggio orientato agli oggetti C++ (cfr. [12]) per quanto riguarda il cuore dell'applicazione (l'aspetto computazionale del problema in esame), mentre ho deciso, dovendo utilizzare i database di Microsoft® Access® già esistenti, di sfruttare le funzionalità offerte da questo programma (cfr. [14]), unite all'impiego del linguaggio Microsoft® Visual Basic® Applications Edition (VBA), per la creazione delle *maschere* utilizzate per il caricamento dei dati.

5.1 Il caricamento dei dati

La progettazione e lo sviluppo dell'interfaccia grafica utilizzata per il caricamento dei dati è avvenuta seguendo le indicazioni dell'azienda sull'aspetto e sulle funzionalità desiderate. Si è così deciso di realizzare una *maschera* a più pagine tramite la funzione fornita dal controllo *struttura a schede* di Microsoft® Access®. La maschera è caratterizzata da tre schede:

- la scheda ***"In produzione"*** che consente di impostare i dati per le commesse attualmente in lavorazione;
- la scheda ***"Da pianificare"*** che consente di impostare i dati delle commesse che si vogliono pianificare;
- la scheda ***"Presse in funzione"*** che permette di selezionare le presse che si vuole utilizzare per la pianificazione e di eseguire la codifica dei dati e l'applicazione C++.

Si analizzano ora più in dettaglio queste sezioni della maschera.

5.1.1 La scheda "In produzione"

Tramite questa scheda (fig. 5.1) si impostano i dati relativi alle commesse attualmente in produzione sulle presse. I *controlli* (oggetti dell'interfaccia grafica che consentono di controllare il programma) visibili nella scheda sono cinque, di cui tre *caselle di testo*, due *caselle combinate* (combo box) ed un *pulsante di comando* che esegue l'eliminazione di una commessa (da utilizzarsi quando la commessa ha terminato la sua lavorazione).

Il campo *"Codice commessa"* deve venire impostato manualmente e rappresenta un identificatore univoco della lavorazione; la casella combinata *"Codice prodotto"* è associata al medesimo campo della Tabella "Prodotti" del database, nella quale sono memorizzati i dati di tutti gli articoli plastici; in questo modo è sufficiente selezionare dal menù a scorrimento il prodotto attualmente in lavorazione. Una volta selezionato il *"Codice prodotto"* viene automaticamente inserita una breve descrizione dell'articolo nel campo *"Descrizione prodotto"*. Allo stesso modo

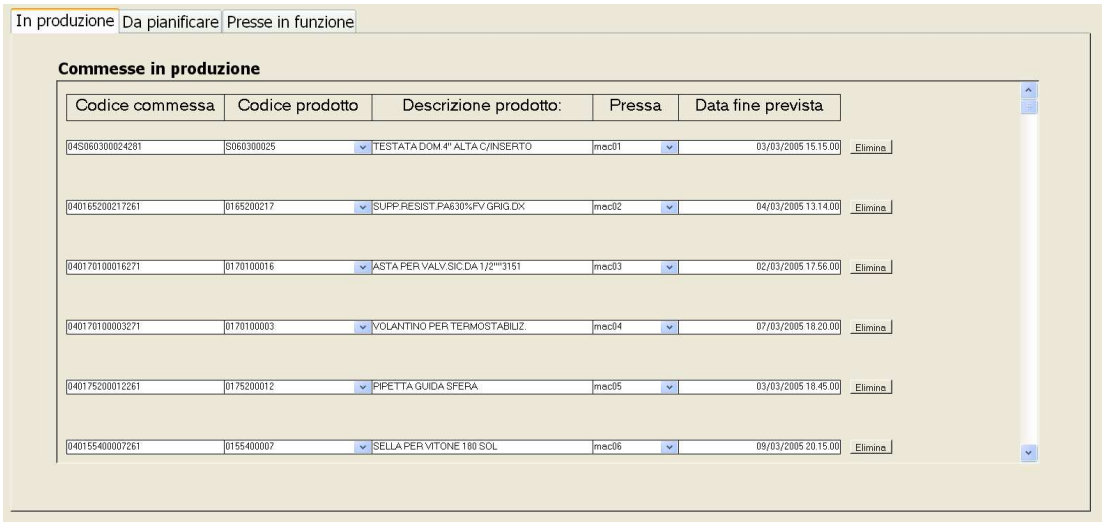

Figura 5.1: Snapshot della scheda *"In produzione"*.

dalla casella combinata *"Pressa"* si può selezionare la pressa su cui si sta stampando il prodotto; infine nel campo *"Data fine prevista"* si deve inserire il giorno e l'ora previste per la fine della lavorazione.

5.1.2 La scheda "Da pianificare"

Tramite questa scheda (fig. 5.2) è possibile impostare i dati per le lavorazioni che si intende pianificare sulle presse. I controlli visibili sono dieci di cui cinque *caselle di testo*, una *casella combinata* e quattro *pulsanti di comando*.

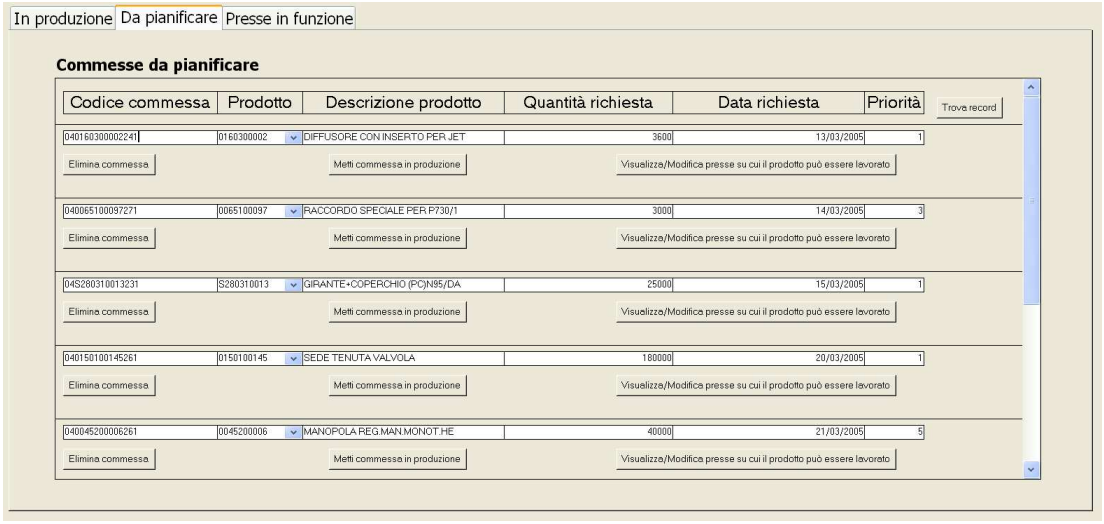

Figura 5.2: Snapshot della scheda *"Da pianificare"*.

Il campo *"Codice commessa"* ha il medesimo significato già spiegato sopra, come pure *"Prodotto"* (identico significato di *"Codice prodotto"*) e *"Descrizione prodotto"*. Nel campo *"Quan-*

tità richiesta" deve essere inserito il numero di pezzi dell'articolo che si intende stampare; il campo *"Data richiesta"* rappresenta la data entro la quale si vorrebbe che la lavorazione terminasse (imposta dall'ordine del cliente), infine il campo *"Priorità"* permette di assegnare un indice di importanza alla commessa (solitamente assegnato in base al cliente).

Per quanto riguarda i pulsanti di comando, *"Trova Record"* permette la ricerca di un elemento all'interno delle commesse da pianificare, che sono mediamente più di 80, *"Elimina commessa"* rimuove semplicemente il record selezionato dalla lista delle commesse da pianificare mentre il pulsante di comando *"Metti commessa in produzione"* permette di spostare la commessa selezionata dalla lista delle commesse da pianificare a quella delle commesse in produzione utilizzando un'altra maschera nella quale viene richiesto di inserire la pressa su cui la commessa è in produzione e la data di fine produzione. Infine il pulsante ''*Visualizza/Modifica presse su cui il prodotto può essere lavorato"* apre una maschera (fig. 5.3) in cui è possibile visualizzare e modificare le presse su cui può essere montato lo stampo relativo al prodotto considerato. Tale maschera si apre anche automaticamente ogni qual volta viene inserita una nuova commessa relativa ad un prodotto il cui stampo non è associato a nessuna pressa (questo perché come detto queste informazioni non erano presenti nei database utilizzati). In questo modo il processo di archiviazione di queste nuove informazioni può avvenire in maniera graduale.

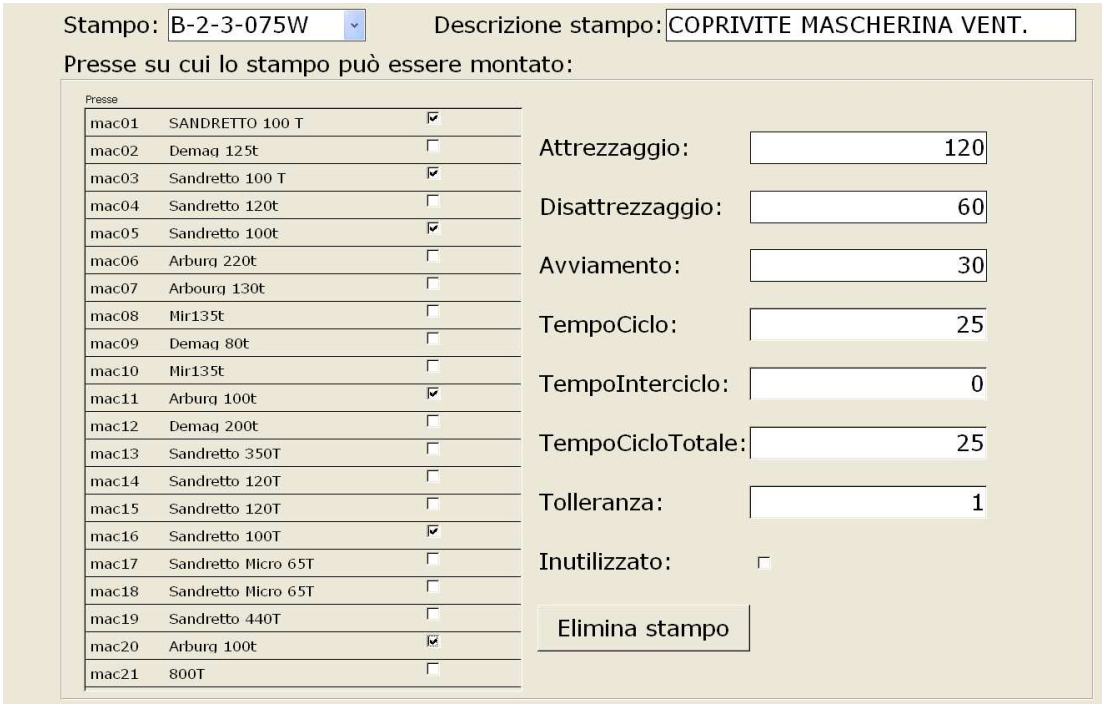

Figura 5.3: Snapshot della maschera *"Relazione stampo-macchina"*.

5.1.3 La scheda "Presse in funzione"

In questa scheda (fig. 5.4) è possibile selezionare le presse che si intende utilizzare nella pianificazione dei lavori; si può decidere infatti di escludere volutamente una pressa dalla pianificazione, deselezionando la *casella di controllo* accanto al nome della pressa, o si può essere costretti a farlo in quanto la pressa risulta inutilizzabile perché ad esempio guasta.

Il pulsante di comando *"Codifica su database"* controlla la correttezza e congruenza dei dati inseriti (avvisa ad esempio nel caso in cui non sia specificata alcuna commessa nella scheda ***In***

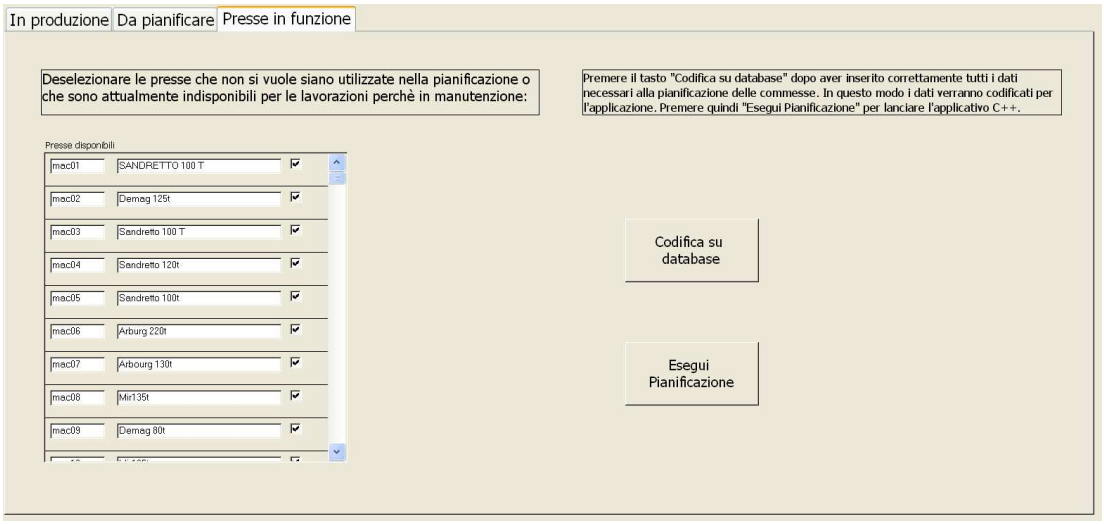

Figura 5.4: Snapshot della scheda *"Presse in funzione"*.

produzione per una pressa selezionata per la pianificazione) e in caso di verifica positiva prepara le tabelle necessarie all'applicativo C++. Il pulsante di comando *"Esegui Pianificazione"* lancia l'applicativo C++.

5.1.4 Struttura delle tabelle

Una volta compilate in maniera corretta le schede appena esaminate, quando viene premuto il pulsante *"Codifica su database"* della scheda *"Presse in funzione"* i dati vengono organizzati in tre tabelle:

- ***Commesse in produzione*** contiene i dati delle lavorazioni in produzione sulle presse;
- ***Commesse da pianificare*** contiene i dati delle lavorazioni da pianificare;
- ***Relazione prodotto-pressa*** associa ogni commessa da pianificare alle presse su cui può venire lavorata.

Tabella "Commesse in produzione"

La struttura della tabella è illustrata in fig. 5.5. I significati dei campi *"CodiceCommessa"*,

CodiceCommessa	CodiceProdotto	Pressa	DataFinePrevista	Materiale	TMateriale	ScalaColore	Stampo	Versione	DurataPreparazione05
04S060300024281	S060300025	mac01	1716	4010744042	320	13	C-4-1-014W	Mono	60
040165200217261	0165200217	mac02	3035	0790742002	250	21	B-4-2-090	Mono	30
040170100016271	0170100016	mac03	437	0770746001	200	2	G-2-3-002	Mono	60
040170100003271	0170100003	mac04	7661	4010743016	200	3	F-2-2-004	Mono	60
040175200012261	0175200012	mac05	1926	0770746001	200	16	D-4-2-011	S13	60
040155400007261	0155400007	mac06	10656	0770746001	200	35	G-3-3-002	Mono	60

Figura 5.5: Snapshot della tabella *"Commesse in produzione"*.

"CodiceProdotto", *"Pressa"*, *"DataFinePrevista"* sono già stati illustrati; per quest'ultimo è necessaria una precisazione: per ovvie necessità di calcolo (effettuare le operazioni dell'algoritmo maneggiando un formato "data" era impensabile) tutti i campi in formato "data" vengono trasformati da un'apposita *routine* al momento della codifica in un valore numerico rappresentante

i minuti lavorativi (tenendo cioè conto dei soli giorni lavorativi dell'azienda) che intercorrono tra il momento in cui la codifica è eseguita e la data specificata. Il campo *"Materiale"* e *"TMateriale"* rappresentano rispettivamente il codice del materiale plastico che viene utilizzato per lo stampaggio del prodotto e il codice della "famiglia" di resine termoplastiche (poliuretani, poliammidi, policarbonati, ecc...) alla quale esso appartiene. *"ScalaColore"* è un indice numerico (tra 0 e 100) che rappresenta la *forza coprente* del colore associato al prodotto. *"Stampo"* rappresenta il codice dello stampo del prodotto, *"Versione"* ne specifica la versione nel caso di stampo *multiversione* e infine *"DurataPreparazione05"* è il nome con cui nei database già esistenti viene identificato il tempo di *disattrezzaggio* (cfr. sezione 1.6) associato allo stampo.

Tabella "Commesse da pianificare"

La struttura della tabella è illustrata in fig. 5.6. In questa tabella sono presenti alcuni nuovi

CodiceCommessa	Prodotto	Materiale	TMateriale	ScalaColore	Stampo	Versione	NumImpr	TempoCicloTotale	DurataPreparazione00	DurataPreparazione01	DurataPreparazione05	QuantitaRichiesta	DataRichiesta	Priorita
040065100097343	S280310023	4010723032	230	55	C-2-3-04	Mono	1	20	120	30	60	25000	33243	3
040150100134444	0150100335	4010222032	120	80	C-1-4-04	S02	4	35	60	30	30	14000	24232	1
040045200045454	0160300202	4010743032	200	90	D-2-3-42	Mono	1	42	120	30	60	15000	11112	1
040175200011245	0175202211	0010742032	120	43	H-3-3-22	Mono	1	24	75	15	30	300000	22315	1
040160300002223	0065123097	1010742032	230	26	G-6-6-76	Mono	1	16,5	120	30	60	420000	14543	5
04S280310013231	S280310013	4010745043	230	65	C-2-3-04	Mono	1	44	120	30	60	25000	17968	1
040150100145261	0150100145	4010742032	120	14	C-3-3-24	Mono	8	16	60	10	30	180000	25168	1
040160300002241	0160300002	4010744001	280	15	C-4-4-04	Mono	1	36	120	30	60	3600	15088	1
040175200011211	0175200011	0770746001	200	6	D-4-1-04	S15	16	16,5	120	30	60	610000	41008	2
040045200006261	0045200006	6030743019	120	3	D-6-3-43	Mono	8	30,3	60	15	30	40000	26608	5
040065100097271	0065100097	0770746001	200	0	F-4-4-0W	Mono	4	47	120	30	60	3000	16528	3

Figura 5.6: Snapshot della tabella *"Commesse da pianificare"*.

campi che necessitano di una breve spiegazione. *"NumImpr"* rappresenta il numero di impronte dello stampo, *"TempoCicloTotale"* è il tempo complessivo del ciclo di stampaggio, *"DurataPreparazione00"* e *"DurataPreparazione01"* rappresentano rispettivamente il tempo di *attrezzaggio* ed il tempo di *avviamento* associati allo stampo. *"QuantitaRichiesta"*, *"DataRichiesta"* e *"Priorita"* hanno il significato già spiegato nella sezione 5.1.2.

Tabella "Relazione prodotto-pressa"

La struttura della tabella è illustrata in fig. 5.7. Ogni record della tabella associa il prodotto

CodiceCommessa	Macchina
040045200006261	mac01
040045200006261	mac03
040045200006261	mac05
040065100097271	mac02
040065100097271	mac03
040150100145261	mac01
040150100145261	mac05
040160300002241	mac01
040160300002241	mac03
040160300002241	mac05
040175200011211	mac02
040175200011211	mac04
040175200011211	mac06
04S280310013231	mac01
04S280310013231	mac02
04S280310013231	mac03
04S280310013231	mac04
04S280310013231	mac05
04S280310013231	mac06
040172300011234	mac12
040172300011234	mac16
040172300011234	mac20
040172300011234	mac05

Figura 5.7: Snapshot della tabella "Relazione prodotto-pressa".

relativo alla lavorazione da pianificare identificata dal campo *"CodiceCommessa"* ad una pressa

identificata dal campo *"Macchina"* sulla quale il prodotto può essere stampato; ovviamente vi possono essere (e solitamente vi sono) più record associati alla stessa lavorazione (cioè con lo stesso valore per *"CodiceCommessa"*) ad indicare appunto che l'articolo plastico relativo alla commessa può essere lavorato su più presse.

5.2 Elaborazione dei dati

Una volta preparate le tabelle contenente i dati delle lavorazioni che si vogliono pianificare, l'elaborazione dell'algoritmo è affidata come detto ad un'applicazione C++. La struttura del programma ricalca l'algoritmo che deve implementare. Tutte le funzioni, le classi e le strutture dati create sono state racchiuse all'interno del *namespace* "Plasticwork". Per spiegare il funzionamento del software esaminerò in dettaglio:

- le ***classi*** che ho definito;
- le ***strutture dati globali*** che vengono utilizzate dalle varie funzioni;
- le ***principali funzioni*** che compongono il programma.

Iniziamo dunque dalle classi.

5.2.1 Le classi

Gli *oggetti* che ho definito sono cinque:

- la classe ***Lotto***;
- la classe ***LottoDaPia***;
- la classe ***Pressa***;
- la classe ***Stampo***;
- la classe ***mossa***.

Lotto

Questa classe rappresenta le commesse in produzione sulle presse al momento in cui la pianificazione viene eseguita. Essa è caratterizzata dai seguenti *dati membro*:

Dati membro	**Descrizione**
int _Id	Questo campo rappresenta un'indice univoco della commessa che viene utilizzato nell'indirizzamento delle strutture dati.
string _CodiceCommessa	Questo campo contiene il codice della commessa.
string _CodiceProdotto	Questo campo contiene il codice del prodotto.
string _Materiale	Questo campo contiene il codice del materiale plastico utilizzato per lo stampaggio del prodotto.
string _TMateriale	Questo campo contiene il codice della famiglia di resine plastiche a cui il materiale appartiene.
string _Stampo	Questo campo contiene il codice dello stampo che si utilizza per la lavorazione.
string _Versione	Questo campo contiene il codice della versione nel caso di stampo multiversione.

Dati membro	Descrizione
int _ScalaColore	Questo campo contiene l'indice relativo alla forza coprente del colore associato al prodotto.
int _Disattrezzaggio	Questo campo contiene una stima del tempo (in minuti) necessario per smontare lo stampo dalla pressa.
long _FinePrevista	Questo campo contiene l'istante (in minuti) in cui terminerà la produzione della commessa.

Una serie di *funzioni membro* per impostare o per ottenere il valore dei vari campi definiti (tutti dichiarati nella sezione "protected") e per stampare i parametri di un oggetto a video o su file correda la definizione della classe contenuta nel file header Lotto.h (cfr. **??**); le definizioni dei *costruttori* della classe e delle funzione non *inline* si trovano invece nel file Lotto.cpp (cfr. **??**). Le funzioni *"EseguiMossa()"* e *"ApplicaTabu()"* (cfr. sezione 5.2.3) sono dichiarate "friend" della classe Lotto ed hanno così accesso diretto ai suoi membri "protected".

LottoDaPia

Questa classe rappresenta le commesse che devono essere pianificate sulle presse. Essa è una *classe derivata* dalla *classe base* Lotto di cui eredita dunque tutti i dati e le funzioni membro. In aggiunta definisce i seguenti dati:

Dati membro	Descrizione
int _NumImpr	Questo campo contiene il numero di impronte dello stampo.
float _TempoCicloTotale	Questo campo contiene il tempo complessivo del ciclo di stampaggio.
int _Avviamento	Questo campo contiene la stima del tempo di avviamento (cfr. sezione 1.6) dello stampo.
int _Attrezzaggio	Questo campo contiene la stima del tempo necessario per montare lo stampo sulla pressa.
int _DurataProduzione	Questo campo contiene il tempo calcolato della durata della lavorazione (escludendo i tempi di setup ovviamente). Il suo valore si ricava dal tempo complessivo di un ciclo, dalla quantità di pezzi richiesti e dal numero di impronte dello stampo.
long _QuantitaRichiesta	Questo campo contiene il numero di pezzi richiesti.
long _DataRichiesta	Questo campo contiene la *due date* della commessa.
long _InizioPrevisto	Questo campo contiene l'istante di inizio della lavorazione nella pianificazione considerata; viene utilizzato dalle varie funzioni nel corso dell'algoritmo.
long _FinePrevista	Questo campo ereditato dalla classe Lotto indica l'istante previsto per la fine della lavorazione nella pianificazione considerata; anch'esso viene usato nel corso dell'algoritmo.
int _Priorita	Questo campo indica la priorità della lavorazione.
string _InLavorazioneSu	Questo campo indica la pressa sulla quale la lavorazione è pianificata.
int _IdPredecessore	Questo campo indica il valore del campo _Id della lavorazione che precede la commessa sulla pressa; viene impostato su -1 se la commessa è stata pianificata subito dopo la lavorazione in produzione.
int _IdSuccessore	Questo campo indica il valore del campo _Id della lavorazione che segue la commessa sulla pressa; vale -1 se la commessa è l'ultima pianificata sulla pressa.

Dati membro	Descrizione
long _wjTj	Questo campo indica il valore del tardiness pesato (weighted tardiness) della commessa.
int _Setup	Questo campo indica il tempo di Setup rispetto alla lavorazione che, nella pianificazione considerata, precede la commessa.
list<string> _MontareSuPresse	Questa lista contiene le presse sulla quale il prodotto può essere lavorato.
vector<int> _SetupTimes	Questo vettore contiene i tempi di Setup rispetto a tutte le commesse (sia quelle in produzione sulle presse, sia quelle da pianificare).

Anche in questo caso le funzioni membro che corredano la definizione della classe servono per impostare o per ottenere i valori dei campi elencati (tutti dichiarati nella sezione "private" della classe) e per una stampa a video o su file di un'istanza della classe. Le funzioni *"EseguiMossa()"*, *"ApplicaTabu()"*, *"CalcolaParametriATCS()"* e *"ApplicaATCS()"* (cfr. sezione 5.2.3) sono dichiarate "friend" della classe ed hanno così accesso diretto ai suoi membri "private". La dichiarazione della classe, insieme alla definizione delle sue funzioni inline si trova nel file Lotto.h (cfr. **??**), mentre la definizione dei costruttori e delle funzioni non inline si trova nel file Lotto.cpp (cfr. **??**).

Pressa

Questa classe rappresenta le presse sulle quali le commesse vengono pianificate. Si è scelto di utilizzare una struttura dati di tipo *"list"* per mantenere le informazioni sulle lavorazioni pianificate sulla pressa; più precisamente viene impiegata una lista di puntatori agli oggetti "LottoDaPia" delle commesse da pianificare dato che la copia e deallocazione di un puntatore a un oggetto non richiede l'invocazione né del *costruttore per copia* né del *distruttore* della classe considerata, garantendo quindi un notevole incremento delle prestazioni del programma. I dati membro della classe sono:

Dati membro	Descrizione
Lotto _CommessaInProduzione	Questo campo rappresenta l'oggetto "Lotto" della commessa in produzione sulla pressa ad inizio pianificazione.
string _Pressa	Questo campo è l'identificatore della pressa.
long _FineUltimaCommessa	Questo campo indica l'istante in cui termina l'ultima commessa pianificata sulla pressa.
map<int,int> _MappaCommDaPia	Questa struttura dati di tipo mappa è utilizzata dalla funzione ApplicaATCS (cfr. 5.2.3) durante la prima fase dell'algoritmo in cui viene costruita la soluzione iniziale.
list<LottoDaPia*> _LottiSchedulati	Questa struttura dati di tipo lista rappresenta le commesse pianificate sulla pressa.

Le funzioni membro della classe permettono di impostare o di ottenere i valori dei campi specificati (tutti dichiarati nella sezione "private") e di stampare a video o su file i parametri di un'istanza della classe. Le funzioni *"EseguiMossa()"*, *"ApplicaTabu()"* e *"ApplicaATCS()"* (cfr. sezione 5.2.3) sono dichiarate "friend" della classe ed hanno così accesso diretto ai suoi membri "private". La dichiarazione della classe, insieme alla definizione delle sue funzioni inline si trova nel file Pressa.h (cfr. **??**), mentre la definizione dei costruttori e delle funzioni non inline si trova nel file Pressa.cpp (cfr. **??**).

Stampo

Questa classe rappresenta gli stampi che vengono utilizzati per la fabbricazione degli articoli plastici. Poiché più prodotti possono utilizzare per la loro lavorazione il medesimo stampo (che è unico) bisogna garantire che la pianificazione dei lavori non permetta che lo stesso stampo sia montato su due o più presse contemporaneamente. Per garantire questo si è scelto di salvare in ogni oggetto Stampo una lista degli intervalli in cui lo stampo è utilizzato; ogni volta che una funzione dell'algoritmo proverà a porre in lavorazione un prodotto su di una pressa, dovrà prima accertarsi che l'intervallo di tempo per l'utilizzo del corrispondente stampo non si sovrapponga a qualche intervallo in cui lo stampo risulta già utilizzato. Il tipo di dati *"Intervallo"* è definito come *pair<long,long>* in cui il primo numero della coppia rappresenta l'istante iniziale della lavorazione e il secondo numero l'istante finale. I dati membro della classe sono:

Dati membro	**Descrizione**
string _Stampo	Questo campo è il codice che identifica lo stampo.
list<Intervallo> _ListaIntervalli	Questa struttura dati di tipo lista memorizza gli intervalli in cui lo stampo è utilizzato.
list<string> _MontareSuPressa	Questa struttura dati di tipo lista memorizza i codici delle presse sulle quali lo stampo può essere montato.

Tutti i dati membro sono dichiarati "private". La funzione membro *"inline bool CheckIntervallo(Intervallo _I)"* viene utilizzata per controllare se un Intervallo _I si sovrapponga o meno agli intervalli di _ListaIntervalli. Essa restituisce ***True*** nel caso l'Intervallo _I sia compatibile (cioè non si sovrapponga) con tutti gli intervalli della lista _ListaIntervalli, ***False*** nel caso ciò non accada. Le altre funzioni membro permettono di aggiungere o di rimuovere un Intervallo dalla lista _ListaIntervalli, di impostare la lista _MontareSuPressa delle presse su cui lo stampo può essere montato e di stampare a video o su file i parametri di un'istanza dell'oggetto. La dichiarazione della classe e delle sue funzioni inline si trova nel file Stampo.h (cfr. **??**), mentre la definizione dei costruttori e delle funzioni non inline si trova nel file Stampo.cpp (cfr. **??**).

mossa

Questa classe è stata creata per rappresentare le *mosse* dell'algoritmo durante l'applicazione della tecnica di Tabu Search. Come spiegato è infatti necessario tenere traccia nella *lista tabù*, delle ultime mosse effettuate dall'algoritmo per evitare di incappare in un ciclo. I dati membro della classe sono:

Dati membro	**Descrizione**
long _Idmossa	Questo campo è un indice numerico assegnato ad una mossa, utilizzato esclusivamente per scopi di debug.
int _IdLotto	Questo campo contiene il codice della commessa che viene spostata in seguito all'esecuzione della mossa.
string _VecchiaPressa	Questo campo contiene il codice della pressa su cui la commessa era in lavorazione prima dell'esecuzione della mossa.
string _NuovaPressa	Questo campo contiene il codice della pressa su cui la commessa viene pianificata dopo l'esecuzione della mossa.
int _IdVecchioPredecessore	Questo campo contiene il codice della lavorazione che precedeva la commessa che si sta spostando tramite la mossa. Il valore di -1 è utilizzato per indicare che la commessa era pianificata come prima lavorazione sulla pressa.

Dati membro	Descrizione
int _IdNuovoPredecessore	Questo campo contiene il codice della lavorazione che precede la commessa sulla nuova pressa dopo l'esecuzione della mossa. Il valore di -1 è utilizzato per indicare che la commessa è pianificata come prima lavorazione sulla pressa.
long _fmossa	Questo campo contiene il valore della funzione obiettivo rispetto al weighted tardiness ($\sum w_j T_j$) della soluzione s che si ottiene a partire dalla pianificazione corrente c applicando la mossa.
long _fbarmossa	Questo campo contiene il valore della funzione obiettivo rispetto ai tempi di setup ($\sum_{i=1}^{m} SU_i$) della soluzione s che si ottiene a partire dalla pianificazione corrente c applicando la mossa.

Tutti i dati membro della classe sono dichiarati "private". Le funzioni membro della classe permettono di impostare o di ottenere i valori dei campi elencati, nonché di stampare a video o su file i parametri di un'istanza della classe. Le funzioni *"EseguiMossa()"* e *"Tabu()"* (cfr. sezione 5.2.3) sono dichiarate "friend" della classe ed hanno così accesso diretto ai suoi membri "private". La dichiarazione della classe e la definizione delle sue funzioni inline si trova nel file mossa.h (cfr. **??**), mentre la definizione dei costruttori e delle funzioni non inline si trova nel file mossa.cpp (cfr. **??**).

5.2.2 Le strutture dati globali

Per la memorizzazione degli oggetti "LottoDaPia", "Pressa", "Stampo" e "mossa" ho utilizzato alcuni tipi dei contenitori astratti che la libreria Standard del C++ mette a disposizione. Più precisamente ho definito le seguenti strutture dati globali:

Strutture dati	Descrizione
map<string,Pressa> MappaPresse	Rappresenta il contenitore di tipo *map* delle presse. La chiave della mappa è la stringa che identifica univocamente ogni pressa.
vector<LottoDaPia> LottiDaPianificare	Rappresenta il contenitore di tipo *vector* per gli oggetti LottoDaPia rappresentanti le commesse da pianificare.
map<string,Stampo> Stampi	Rappresenta il contenitore di tipo *map* per gli oggetti Stampo rappresentanti gli stampi dei prodotti.
priority_queue<finemac> CodaPresse	Rappresenta il contenitore di tipo *priority_queue* (coda prioritaria) per il tipo *"finemac"* definito come *pair<long,string>* in cui il primo campo indica l'istante in cui termina l'ultima lavorazione sulla pressa identificata dal campo string della coppia. Questa coda prioritaria è utilizzata dalle funzioni che costruiscono una pianificazione iniziale applicando la regola di carico ATCS.
list<mossa> CodaTabu	Rappresenta il contenitore di tipo *list* che viene utilizzato come una lista di lunghezza fissa per contenere gli oggetti mossa che rappresentano le ultime mosse eseguite dall'algoritmo durante la fase di applicazione della tecnica Tabu Search.

La scelta di utilizzare il tipo vector per gli oggetti "LottoDaPia" deriva dalla possibilità di utilizzare la sintassi dello *stile array* (cioè con indirizzamento diretto tramite l'operatore di

subscript) al posto dello *stile STL* (cioè tramite gli *iteratori*) per accedere agli elementi in esso contenuti, utilizzando come indice il campo _Id della classe. Per l'implementazione della lista delle mosse tabù ho utilizzato un contenitore di tipo *list* in cui ogni nuova mossa viene inserita in testa alla lista e la mossa "più vecchia" viene estratta dalla coda se la dimensione della lista ha raggiunto il valore massimo consentito. Altre strutture dati, in particolare di tipo *priority_queue* (coda prioritaria) sono state utilizzate nel corpo di alcune funzioni del programma. Tutte le strutture dati globali sono definite nel file utility.cpp (cfr. **??**).

5.2.3 Le principali funzioni

Le principali funzioni vengono richiamate durante l'esecuzione del programma dalla funzione *"main()"* che rappresenta il punto di ingresso di ogni applicativo C++. Il codice della funzione si trova nel file Principale.cpp (cfr. **??**) ed è riportato di seguito:

```
1    int main()
2    {
3        using namespace Plasticwork;
4        string risp;
5        while (!((risp==1)||(risp==2)||(risp==3)))
6        {
7           cout << "Scegliere il tipo di operazione che si
8            intende eseguire digitando il numero
9            corrispondente:" << endl;
10           cout << "1. Eseguire la pianificazione con i dati
11           memorizzati nel database;" << endl;
12           cout << "2. Eseguire la pianificazione con dati
13           random generati in maniera volatile;" << endl;
14           cout << "3. Generare dati random, salvarli nel
15           database ed utilizzarli per la pianificazione;"
16           << endl;
17           cin >> risp;
18       }
19       if (risp=="1")
20           CodificaDati();
21       else if (risp=="2")
22           CodificaDatiRandom();
23       else
24       {
25           SalvaSoluzioneRandom();
26           CodificaDati();
27       }
28       CalcolaSetup();
29       CalcolaParametriATCS();
30       ApplicaATCS();
31       time_t start;
32       time_t end;
33       double tempo;
34       cout << "Inserire il tempo in secondi per cui si
35       intende far lavorare il programma:" << endl;
36       cin >> tempo;
37       time(&start);               // Avvio il cronometro
38       int iterazioni(1);
```

```
39      while (true)
40      {
41          Tabu(1);                //Fase intensificazione
42          time(&end);
43          if (difftime(end,start)>tempo)
44              break;
45          if (iterazioni%2)
46              Tabu(2);            //Fase di diversificazione1
47          else
48              Tabu(3);            //Fase di diversificazione2
49          time(&end);
50          if (difftime(end,start)>tempo)
51              break;
52          iterazioni++;
53      }
54  }
```

Le prime funzioni invocate (righe 1–27) sono quelle relative alla creazione degli oggetti e delle strutture dati in base ai dati di ingresso del problema. Il programma prevede sostanzialmente tre metodi di caricamento dei dati:

- caricamento dei dati da database attraverso la funzione *"CodificaDati()"*;
- generazione random dei dati con la funzione *"CodificaDatiRandom()";*
- generazione random e salvataggio dei dati su database attraverso la funzione *"SalvaSoluzioneRandom()"*;

La funzione per generare istanze del problema in maniera random si è resa necessaria per permettere di testare approfonditamente il programma durante il suo sviluppo poiché si disponeva solamente di un'unica istanza reale. La funzione *"SalvaSoluzioneRandom()"* è stata aggiunta in seguito per permettere di salvare i dati generati in maniera random su database in modo da poterli riutilizzare in seguito; infatti, come vedremo meglio nel prossimo capitolo, per calibrare alcuni parametri dell'algoritmo alcuni test dovevano essere ripetuti più volte su uno stesso insieme di istanze. Analizziamo le caratteristiche principali di queste funzioni.

void CodificaDati()

La funzione utilizza alcune delle librerie MFC (Microsoft® Foundation Class) per connettersi al database di Microsoft® Access® tramite ODBC (Open Database Connectivity). Le informazioni contenute nelle tabelle del database sono utilizzate per creare:

- gli oggetti della classe "Pressa", rappresentanti le presse in funzione, che vengono inseriti nel loro contenitore di tipo mappa "MappaPresse";
- gli oggetti della classe "Lotto", rappresentanti le commesse attualmente in lavorazione sulle presse, che vengono inseriti negli oggetti "Pressa" corrispondenti;
- gli oggetti della classe "LottoDaPia", rappresentanti le commesse che devono essere pianificate, che vengono inseriti nel loro contenitore di tipo vector "LottiDaPianificare".
- gli oggetti della classe "Stampo", rappresentanti gli stampi delle commesse attualmente in produzione e delle commesse da pianificare, che vengono inseriti nel loro contenitore di tipo mappa "Stampi".

Negli oggetti "Stampo" rappresentanti gli stampi delle commesse attualmente in produzione viene inserito anche l'intervallo di tempo indicante la fine prevista della lavorazione sulla pressa; negli oggetti "Stampo" e "LottoDaPia" vengono inserite le informazioni riguardanti le presse sulle quali lo stampo può essere montato (e la rispettiva commessa può essere prodotta). Infine nella mappa "_MappaCommDaPia" degli oggetti "Pressa" sono inseriti i codici delle commesse che possono essere pianificate sulla pressa; questa struttura dati viene utilizzata dalla funzione *"ApplicaATCS()"* che vedremo in seguito.

void CodificaDatiRandom()

Per creare delle istanze di dati in maniera random si è prima studiato la struttura dell'istanza reale del problema che si aveva a disposizione, analizzando le caratteristiche e le distribuzioni dei dati relativi alle lavorazioni e cercando di riprodurle nelle istanze generate. Si è visto ad esempio che in media solo 5/6 commesse (su un totale di 80/100 che compongono un'istanza media del problema) possono condividere il proprio stampo oppure che solo il 5% delle lavorazioni sono caratterizzate da un tempo di attrezzaggio del proprio stampo di 180 minuti e si è cercato di riproporre questa struttura anche nelle istanze generate. La funzione genera di volta in volta i dati relativi alle lavorazioni da pianificare e a quelle attualmente in produzione, creando allo stesso modo della funzione *"CodificaDati()"* gli oggetti delle classi "Pressa", "Stampo", "Lotto" e "LottoDaPia".

void SalvaSoluzioneRandom()

Questa funzione utilizza lo stesso sistema di *"CodificaDatiRandom()"* per generare i dati delle lavorazioni; a differenza di quest'ultima però i dati vengono salvati sul database di Microsoft® Access® nelle tre tabelle che abbiamo descritto nella sezione 5.1.4 e possono quindi essere utilizzati più volte tramite la funzione *"CodificaDati()"*.

Prima di esaminare le funzioni che implementano effettivamente l'algoritmo, rimane da descrivere la funzione *"CalcolaSetup()"* che viene invocata (riga 28) dalla funzione *"main()"* subito dopo la preparazione degli oggetti e delle strutture dati avvenuta tramite le funzioni precedentemente descritte.

void CalcolaSetup()

Questa funzione calcola attraverso un semplice algoritmo (descritto in fig. 5.8) i tempi di setup di ogni commessa da pianificare rispetto a tutte le altre commesse (cioè sia quelle da pianificare, sia quelle in produzione) salvandoli nella struttura di tipo vector "_SetupTimes" degli oggetti "LottoDaPia". L'algoritmo tiene conto dei tempi di attrezzaggio, disattrezzaggio e avviamento degli stampi, ma anche dei tempi impiegati per lo spurgo del cilindro delle presse durante l'operazione di cambio-stampo che dipendono dai materiali delle lavorazioni e dai rispettivi colori. Nel corpo della funzione viene effettuato un ciclo esterno sugli elementi del vettore "LottiDaPianificare"; ad ogni iterazione viene selezionato una commessa e quindi eseguito un nuovo ciclo sullo stesso vettore per il calcolo dei tempi di setup rispetto alle commesse da pianificare e un'ulteriore ciclo sulla mappa "MappaPresse" per il calcolo dei tempi di setup rispetto alle commesse in produzione.

Il codice delle funzioni fino ad ora esaminate si trova nel file utility.cpp (cfr. **??**).

A questo punto, tutto è pronto per l'implementazione vera e propria delle due fasi in cui abbiamo visto è concettualmente strutturato il nostro algoritmo. Le funzioni che si occupano durante la prima fase di costruire una soluzione iniziale al problema sono invocate subito dopo *"CalcolaSetup()"* (righe 29–30). Esse sono:

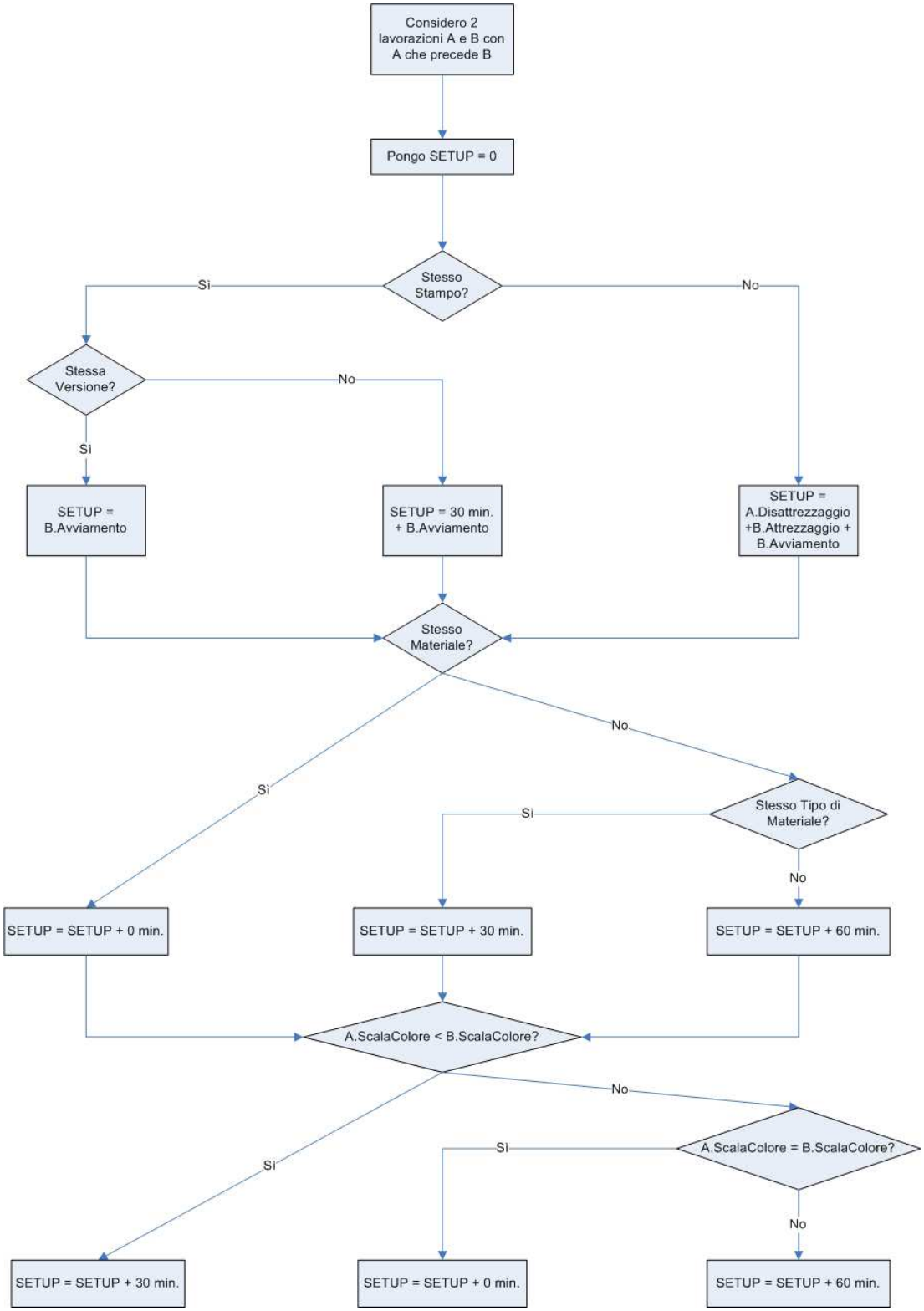

Figura 5.8: Algoritmo per il calcolo del tempo di setup tra due lavorazioni

- *"CalcolaParametriATCS()";*
- *"ApplicaATCS()".*

void CalcolaParametriATCS()

Questa funzione calcola tutti i parametri che vengono utilizzati durante l'applicazione della regola di carico ATCS precedentemente descritta (cfr. sezione 3.2.1). Per il calcolo degli *scaling parameter* K_1 e K_2 (esp. 3.6 e 3.7) che vengono utilizzati nell'espressione 3.1 che assegna ad ogni job ancora da pianificare un indice di priorità, bisogna avere a disposizione i valori di alcune statistiche. Queste statistiche vengono calcolate proprio all'interno di questa funzione secondo le espressioni 3.2, 3.3 e 3.4. I valori degli attributi dei job necessari per il calcolo di queste statistiche vengono recuperati scandendo la struttura dati "LottiDaPianificare" nella quale si trovano gli oggetti "LottoDaPia" rappresentanti le commesse da pianificare sulle presse.

void ApplicaATCS()

Questa funzione costruisce una soluzione iniziale al problema in esame, applicando la fase costruttiva dell'euristica GRASP (cfr. sezione 3.1) e sfruttando come funzione di ordinamento di tipo greedy la regola di carico ATCS. La funzione utilizza la struttura dati di tipo *coda prioritaria* "CodaPresse" per scegliere la pressa su cui pianificare ad ogni passo un nuovo job; la priorità è data dall'istante in cui termina la produzione dell'ultima lavorazione pianificata sulla pressa. Ad ogni iterazione viene selezionata la pressa con la priorità più elevata (cioè quella che termina per prima la lavorazione della sua ultima commessa pianificata) e viene calcolato secondo l'espressione 3.1 un indice di priorità per tutti i job che possono essere pianificati su tale pressa (tali job sono salvati nella mappa "_MappaCommDaPia" di ogni oggetto "Pressa"). Anche per l'ordinamento dei job viene utilizzata una struttura temporanea di tipo *coda prioritaria* in cui la priorità è data proprio dal valore dell'indice calcolato. Il job che deve essere pianificato viene quindi selezionato in maniera casuale tra i primi M job della coda che costituiscono così la lista RCL dei possibili candidati come previsto dalla tecnica GRASP. Quando la lunghezza della coda prioritaria dei job che possono essere pianificati sulla pressa è inferiore alla lunghezza $|M|$ prevista per la lista RCL, viene selezionato sempre il job in testa alla coda di priorità. Una volta selezionato il job che si vuole pianificare, la funzione calcola gli istanti previsti di inizio e fine produzione della lavorazione e controlla che lo stampo necessario non sia già utilizzato nell'intervallo di tempo considerato su qualche altra pressa. Se non ci sono sovrapposizioni il job viene pianificato sulla pressa, aggiungendolo alla lista "_LottiSchedulati" dell'oggetto "Pressa" considerato e rimuovendolo (nel caso fosse presente) dalle mappe "_MappaCommDaPia" di tutte le presse (in modo che non venga più considerato per essere nuovamente pianificato). Viene aggiornato inoltre il campo "_FineUltimaCommessa" sulla pressa considerata, ed essa viene quindi reinserita con la nuova priorità nella coda "CodaPresse". Qualora il job non possa venire pianificato sulla pressa attualmente considerata per il motivo spiegato, esso viene rimosso dalla coda di priorità temporanea dei job e il procedimento di scelta viene ripetuto per un altra lavorazione. Se la coda di priorità dei job si svuota e nessuna lavorazione è stata pianificata la pressa considerata viene rimossa da "CodaPresse" e si seleziona la nuova pressa in testa alla coda.

La tecnica del GRASP proprio per il fattore di casualità che introduce nella scelta delle lavorazioni da pianificare, non garantisce di ottenere sempre la stessa soluzione iniziale partendo ovviamente da una medesima istanza di dati; per effettuare alcuni test si aveva proprio questa esigenza e per questo motivo si è introdotta la possibilità di scegliere se applicare o meno la tecnica GRASP durante l'esecuzione della funzione. Se si sceglie di non applicarla, il job selezionato per la pianificazione è sempre quello con il valore più elevato dell'indice di priorità calcolato in 3.1.

Sia la funzione *"ApplicaATCS()"*, sia la funzione *"CalcolaParametriATCS()"* sono definite nel file ATCS.cpp (cfr. **??**).

Al termine dell'esecuzione della funzione *"ApplicaATCS()"* la prima fase dell'algoritmo è conclusa e abbiamo dunque ottenuto una pianificazione iniziale delle lavorazioni sulle presse. All'interno della funzione *"main()"* viene a questo punto richiesto di inserire un intervallo di tempo per stabilire la durata della seconda fase dell'algoritmo (righe 33–36). Subito dopo viene azionato un cronometro (riga 37) ed ha inizio un ciclo (righe 39–53) nel quale viene invocata la funzione *"Tabu()"* che ogni volta esegue la fase di intensificazione o di diversificazione (a seconda del valore dell'argomento passato) della tecnica di Tabu Search. Al termine di ogni fase viene controllato il tempo trascorso e se esso supera l'intervallo stabilito il programma termina la sua esecuzione. Possiamo ora descrivere la funzione *"Tabu()"* che implementa la seconda fase dell'algoritmo nella quale si cerca di migliorare la pianificazione iniziale applicando la tecnica di Tabu Search (cfr. sezione 4.1). Essa invoca al suo interno altre due funzioni che descriveremo brevemente:

- *"ApplicaTabu()"*;
- *"EseguiMossa()"*.

void Tabu(int fase)

Il parametro di tipo int "fase" specifica come detto se ci troviamo nella fase di intensificazione ($fase = 1$) o in una delle due possibili fasi di diversificazione ($fase = 2$ o 3). La funzione *"Tabu()"* inizia impostando il numero di "mosse" che verranno eseguite a seconda della fase in cui ci troviamo, dopo di che inizia ad eseguire un ciclo per un numero di iterazioni pari al numero di mosse. All'interno del ciclo la funzione, ricalcando lo schema della seconda fase dell'algoritmo, esegue le seguenti operazioni:

1. Crea, allocandola dinamicamente, una struttura dati di tipo coda prioritaria ("CodaMosse") che servirà per contenere oggetti "funzmossa" definiti come *pair<long,mossa>* in cui il primo campo della coppia contiene il valore della funzione obbiettivo modificata $\widetilde{f}$ (a seconda della fase considerata) che si ottiene applicando la mossa memorizzata nel secondo campo alla pianificazione corrente. La coda prioritaria sarà ordinata in maniera decrescente in base al valore del primo campo della coppia.

2. Invoca la funzione *"ApplicaTabu()"* che esamineremo in seguito passandole come parametri la coda prioritaria appena creata e l'indicazione della fase (intensificazione o diversificazione) in cui si trova l'algoritmo. Questa funzione come vedremo calcolerà tutte le possibili mosse nell'intorno della soluzione corrente, ordinandole nella coda di priorità.

3. A questo punto la funzione estrae la mossa in testa alla coda di priorità "CodaMosse" e verifica se essa è "l'inversa" di una mossa memorizzata nella "CodaTabu" (la struttura dati globale di tipo lista che contiene le ultime mosse eseguite dall'algoritmo).

   CASO A. La mossa è tabù. La funzione verifica se possa essere applicato il criterio d'aspirazione controllando se il valore della funzione obbiettivo reale che si otterrebbe applicando la mossa è minore del valore della funzione obbiettivo reale della miglior soluzione finora trovata.

   caso A1. Vale il criterio d'aspirazione Viene invocata la funzione *"EseguiMossa()"* che analizzeremo in seguito. Se essa restituisce ***true*** allora la mossa è stata eseguita correttamente, la soluzione migliore viene aggiornata con il valore della nuova soluzione corrente, viene eliminata (deallocandola dalla memoria) la coda di priorità "CodaMosse" e il ciclo ha terminato una sua iterazione. Se

"EseguiMossa()" restituisce ***false*** significa che la mossa non è applicabile, essa viene quindi rimossa da "CodaMosse" e si torna all'inizio del passo 3.

caso A2. Non vale il criterio d'aspirazione La mossa "tabù" viene eliminata da "CodaMosse" e si torna all'inizio del passo 3.

CASO B. La mossa non è tabù. Viene invocata la funzione *"EseguiMossa()"*. Come prima se essa restituisce ***true*** significa che la mossa è stata applicata con successo. A questo punto si controlla se la nuova soluzione corrente è migliore della soluzione ottima finora trovata ed eventualmente si aggiorna la soluzione ottima; si elimina "CodaMosse"; si inserisce la mossa appena eseguita in testa a "CodaTabu" e si elimina la mossa in coda (qualora "CodaTabu" non abbia ancora raggiunto la dimensione massima consentita, si aggiunge solamente la mossa senza ovviamente eliminarne nessuna) e il ciclo ha terminato una sua iterazione. Se *"EseguiMossa()"* restituisce ***false*** significa che la mossa non è applicabile, essa viene quindi rimossa da "CodaMosse" e si torna all'inizio del passo 3.

void ApplicaTabu(int fase, priority_queue<funzmossa> *CodaMosse)

Il parametro di tipo *int* della funzione specifica la fase (intensificazione o diversificazione) in cui l'algoritmo si trova mentre l'altro parametro rappresenta un puntatore alla struttura di tipo coda prioritaria di oggetti "funzmossa" che abbiamo definito sopra. Questa funzione calcola partendo dalla pianificazione corrente tutte le mosse appartenenti all'intorno della soluzione corrente. Per ogni mossa la funzione calcola il valore della nuova funzione obbiettivo che si ottiene in seguito all'applicazione della mossa e inserisce la mossa in "CodaMosse" secondo la priorità determinata dalla fase in cui ci troviamo. Vi sono sostanzialmente tre possibili fasi (una fase di intensificazione e due di diversificazione) caratterizzate da un valore diverso del parametro *fase*:

$\boldsymbol{fase = 1}$ In questa fase di intensificazione le mosse vengono ordinate in maniera decrescente secondo il valore della funzione obbiettivo reale

$$\widetilde{f} = f = \sum w_j T_j + \sum_{i=1}^{m} SU_i \tag{5.1}$$

$\boldsymbol{fase = 2}$ In questa fase di diversificazione le mosse sono ordinate in maniera decrescente secondo il valore della funzione obbiettivo modificata

$$\widetilde{f} = \sum_{i=1}^{m} SU_i \tag{5.2}$$

che tiene conto esclusivamente dei *tempi di setup* tra le lavorazioni.

$\boldsymbol{fase = 3}$ In questa fase di diversificazione le mosse sono ordinate in maniera decrescente secondo un valore puramente random.

Le due fasi di diversificazione vengono quindi alternate durante l'esecuzione del programma. La funzione non si preoccupa di controllare per ogni mossa se essa è applicabile o meno (cioè se in seguito all'applicazione della mossa vi siano sovrapposizioni degli intervalli di utilizzo degli stampi). Questa scelta è stata fatta in quanto risultava troppo oneroso da un punto di vista di calcolo effettuare questo controllo per ogni mossa e, considerando l'esiguo numero di lavorazioni che in un'istanza del problema possono condividere il proprio stampo, solo poche mosse risultano inapplicabili per questo motivo. Si è preferito dunque effettuare il controllo solo al momento dell'esecuzione effettiva della mossa, cioè nella funzione *"EseguiMossa()"*.

bool EseguiMossa(mossa m, int flag)

Il primo parametro della lista dei parametri della funzione è l'oggetto "mossa" che rappresenta la mossa che si vuole applicare alla soluzione corrente; il secondo parametro è un parametro di controllo della funzione utilizzato per distinguere una chiamata esterna da una chiamata ricorsiva (utilizzata per ripristinare una configurazione iniziale quando la mossa non è applicabile). La funzione restituisce ***true*** se la mossa è stata applicata con successo, mentre restituisce ***false*** quando la mossa non è applicabile in quanto ci sono delle sovrapposizioni negli intervalli di utilizzo di alcuni stampi. Nel corpo della funzione sono implementate le operazioni necessarie a modificare tutte le strutture dati coinvolte dalla mossa; tutti i campi degli oggetti "LottoDaPia" che rappresentano le commesse che sono coinvolte dalla mossa devono essere aggiornati, la commessa che viene spostata deve essere tolta dalla lista "_LottiSchedulati" dell'oggetto "Pressa" che rappresentava la vecchia macchina dove era stato pianificato e inserito nella stessa lista della nuova macchina su cui viene pianificato e nella corretta posizione; infine per gli oggetti "Stampo" relativi alle lavorazioni coinvolte devono essere aggiornate le liste "_ListaIntervalli" degli intervalli di utilizzo dello stampo.

Il codice delle tre funzioni esaminate, che implementano la seconda fase del'algoritmo, si trova nel file tabu.cpp (cfr. **??**).

Concludiamo il capitolo descrivendo come vengono salvati i risultati ottenuti dall'algoritmo.

5.2.4 I file di output

Il programma salva le informazioni sulla soluzione iniziale ottenuta dopo l'applicazione della funzione *"ApplicaATCS()"* e sulla soluzione migliore ottenuta dopo l'applicazione ripetuta della funzione *"Tabu()"* in due file di testo aventi la medesima struttura che ora riportiamo.

```
Pressa: mac7
Fine ultima commessa in lavorazione: 82444
Codice commessa in produzione: 040045200006261
Indici dei lotti pianificati sulla pressa:
Codice commessa     Inizio      Fine
040065100097271     8308        51946
040150100145261     51946       82444


Pressa: mac8
Fine ultima commessa in lavorazione: 58309
Codice commessa in produzione: 040172300011234
Indici dei lotti pianificati sulla pressa:
Codice commessa     Inizio      Fine
040150100145261     8492        15988
040160300002241     15988       17665
040175200011211     17665       37870
04S280310013231     37870       58309


Pressa: mac9
Fine ultima commessa in lavorazione: 57607
Codice commessa in produzione: 040172302311234
Indici dei lotti pianificati sulla pressa:
Codice commessa      Inizio      Fine
0S7172306018234      589         23588
```

```
040177870011456       23588       27645
045685320032422       27645       37727
0S3242341789445       37727       57607


Funzione obbiettivo: 5768792
Valore penalità: 30520
Iterazione: 5909
```

Come si vede per ogni pressa è indicato l'istante in cui termina l'ultima lavorazione pianificata, il codice della commessa in produzione e la lista delle commesse che sono state pianificate sulla macchina con l'indicazione del loro codice, dell'istante d'inizio e di fine previsti per la loro lavorazione. Nelle ultime righe del file è indicato il valore della funzione obbiettivo reale ($f = \sum w_j T_j + \sum_{i=1}^{m} SU_i$) diviso nelle sue due componenti:

```
Funzione obbiettivo: 5768792
```

indica il valore rispetto al *weighted tardiness* delle commesse ($\sum w_j T_j$), mentre

```
Valore penalità: 30520
```

ne indica il valore rispetto ai *setup time* ($\sum_{i=1}^{m} SU_i$). Per il file di output contente le informazioni sulla pianificazione migliore ottenuta durante l'applicazione del Tabu Search viene indicata anche l'iterazione alla quale tale pianificazione è stata ottenuta.

Per tenere traccia della "traiettoria" seguita durante la fase di applicazione della tecnica di Tabu Search si è deciso di stampare su di un file i valori della funzione obbiettivo reale della soluzione ottenuta dopo l'applicazione di una mossa ad ogni iterazione. Il formato del file di output è il seguente:

```
0 6885834
1 6729431
2 6623981
3 6572057
4 6523193
5 6489314
6 6460934
7 6434255
8 6397284
9 6362136
10 6340533
11 6319989
12 6299933
13 6279880
14 6248353
15 6229258
16 6211750
17 6195958
18 6182072
19 6169157
20 6157136
21 6145520
22 6134263
23 6122899
24 6112088
```

```
25 6102208
26 6092955
```

Come si vede esso indica semplicemente sulla colonna di sinistra il numero dell'iterazione (l'iterazione 0 indica il valore della soluzione iniziale) e sulla colonna di destra è indicato il valore della funzione obbiettivo reale.

Altri file di output vengono generati durante l'esecuzione del programma, ma essi sono stati per lo più utilizzati a scopo di debug per controllare la correttezza delle operazioni effettuate e non verranno dunque illustrati in dettaglio.

Capitolo 6

Test e risultati computazionali

Come abbiamo visto nel precedente capitolo, apposite funzioni sono state scritte nel codice del programma per generare in maniera random delle istanze del problema che fossero simili nella loro struttura all'unica istanza reale di cui si disponeva. In questo modo è stato possibile testare in maniera accurata il programma nel corso del suo sviluppo e inoltre grazie alla possibilità di salvare le istanze generate su database (tramite la funzione *SalvaSoluzioneRandom()*) è stato possibile eseguire una serie di test per calibrare alcuni parametri dell'algoritmo.

Nella prima parte del capitolo analizzeremo come sono stati effettuati questi test e i risultati che hanno prodotto, mentre nella seconda parte confronteremo le prestazioni fornite dal programma rispetto alla soluzione ottenuta con il sistema attuale di pianificazione per l'istanza reale del problema.

6.1 Impostazione dei parametri dell'algoritmo

Nella seconda fase dell'algoritmo (cfr. cap. 4) viene applicata la tecnica di Tabu Search per un intervallo di tempo specificato. Durante questa fase l'algoritmo alterna continuamente la fase di intensificazione e la fase di diversificazione della procedura per un'esplorazione completa dello spazio delle soluzioni ammissibili del problema. Il *numero di "mosse"* che caratterizzano la fase di intensificazione o diversificazione è un parametro dell'algoritmo che deve essere calibrato per ottenere le migliori prestazioni dal programma.

Un altro parametro che può contribuire ad un miglioramento delle prestazioni complessive dell'algoritmo se calibrato opportunamente è rappresentato dalla *lunghezza della lista tabù*, lista che come detto serve per memorizzare le ultime mosse eseguite dall'algoritmo cercando di impedire l'instaurarsi di cicli durante l'applicazione della procedura di Tabu Search. Entrambi questi parametri sono stati calibrati grazie ai risultati ottenuti dall'applicazione di una serie di test.

6.1.1 I test

I test sono stati effettuati sui dati di 11 istanze del problema:

- l'istanza reale;
- 10 istanze generate in maniera random e salvate su database grazie alla funzione *SalvaSoluzioneRandom()*.

La struttura delle istanze in relazione al numero di presse ed al numero di commesse da pianificare è riportata di seguito.

	Numero presse	Numero commesse
Istanza reale	21	83
Istanza 1	21	105
Istanza 2	21	95
Istanza 3	20	85
Istanza 4	20	100
Istanza 5	20	90
Istanza 6	21	80
Istanza 7	21	110
Istanza 8	19	100
Istanza 9	19	80
Istanza 10	21	100

Tutti i test sono stati effettuati sul medesimo computer (dotato di un processore Intel® Pentium® M 735 con 512 Mb di memoria RAM), garantendo così la stessa potenza computazionale per ognuno di essi.

I parametri che si sono presi in considerazione sono tre:

Parametro LT Rappresenta la lunghezza della lista tabù. $LT = 15$ ed $LT = 25$ sono i due possibili valori considerati per il parametro.

Parametro NI Rappresenta il numero di mosse che verranno effettuate durante ogni fase di intensificazione della procedura Tabu Search. $NI = 400$ ed $NI = 800$ sono i due possibili valori considerati per il parametro.

Parametro ND Rappresenta il numero di mosse che verranno effettuate durante ogni fase di diversificazione della procedura Tabu Search. $ND = 50$ ed $ND = 100$ sono i due possibili valori considerati per il parametro.

Dato che per ognuno dei parametri sono stati considerati, come visto, due valori significativi, vi sono dunque 8 possibili configurazioni:

	Parametro LT	Parametro NI	Parametro ND
Caso 1	15	400	50
Caso 2	15	400	100
Caso 3	15	800	50
Caso 4	15	800	100
Caso 5	25	400	50
Caso 6	25	400	100
Caso 7	25	800	50
Caso 8	25	800	100

Ognuna di queste configurazioni è stata applicata alle 11 istanze che costituiscono il nostro testbed. Durante ogni prova, la costruzione della soluzione iniziale avveniva attraverso l'applicazione della funzione *ApplicaATCS()* secondo le modalità descritte nella sezione 5.2.3, disattivando però la tecnica GRASP cosicché la soluzione di partenza della procedura di Tabu Search nella seconda fase dell'algoritmo fosse sempre la stessa per ogni configurazione dei parametri applicata ad una medesima istanza. Si è fissata la durata dell'applicazione della fase di Tabu Search ad un intervallo di 2 ore di tempo. I test hanno dunque richiesto complessivamente circa 176 ore per la loro esecuzione.

6.1.2 I risultati dei test

Durante ogni prova veniva salvata nel file di output descritto nella sezione 5.2.4 la pianificazione migliore ottenuta durante l'esecuzione del programma, con l'indicazione del valore della funzione obbiettivo reale ($f = \sum w_j T_j + \sum_{i=1}^{m} SU_i$) diviso nelle sue due componenti:

- ($\sum w_j T_j$) che ne indica il valore rispetto al weighted tardiness delle commesse;
- ($\sum_{i=1}^{m} SU_i$) che ne indica il valore rispetto ai setup time delle commesse.

Vengono ora riportati i risultati ottenuti per ogni istanza del nostro testbed (i valori in rosso indicano la configurazione per cui si ottiene il miglior risultato, i valori in blu rappresentano il secondo miglior risultato).

Istanza reale	$\sum w_j T_j$	$\sum_{i=1}^{m} SU_i$	$\sum w_j T_j + \sum_{i=1}^{m} SU_i$
Caso 1	377569	21165	398734
Caso 2	395296	21240	416536
Caso 3	384542	21210	406752
Caso 4	376303	21420	397723
Caso 5	376067	20670	396737
Caso 6	378470	21120	399590
Caso 7	372789	21060	393849
Caso 8	383936	21180	405116

Istanza 1	$\sum w_j T_j$	$\sum_{i=1}^{m} SU_i$	$\sum w_j T_j + \sum_{i=1}^{m} SU_i$
Caso 1	5759626	30760	5790386
Caso 2	5794678	30760	5825438
Caso 3	5820895	30400	5851295
Caso 4	5723629	30700	5754329
Caso 5	5768792	30520	5799312
Caso 6	5783783	30580	5814363
Caso 7	5741915	30760	5772675
Caso 8	5736312	30610	5766922

Istanza 2	$\sum w_j T_j$	$\sum_{i=1}^{m} SU_i$	$\sum w_j T_j + \sum_{i=1}^{m} SU_i$
Caso 1	3072722	27885	3100607
Caso 2	3088936	27615	3116551
Caso 3	3109621	27675	3137296
Caso 4	3099894	27615	3127509
Caso 5	3057452	27705	3085157
Caso 6	3062762	27285	3090047
Caso 7	3071121	27645	3098766
Caso 8	3046559	28065	3074624

Istanza 3	$\sum w_j T_j$	$\sum_{i=1}^{m} SU_i$	$\sum w_j T_j + \sum_{i=1}^{m} SU_i$
Caso 1	2652716	24500	2677216
Caso 2	2569783	24500	2594283

Istanza 3	$\sum w_j T_j$	$\sum_{i=1}^{m} SU_i$	$\sum w_j T_j + \sum_{i=1}^{m} SU_i$
Caso 3	2676379	24260	2700639
Caso 4	2596839	24410	2621249
Caso 5	2634303	24410	2658713
Caso 6	2594554	24140	2618694
Caso 7	2640946	24680	2665626
Caso 8	2644469	24320	2668789

Istanza 4	$\sum w_j T_j$	$\sum_{i=1}^{m} SU_i$	$\sum w_j T_j + \sum_{i=1}^{m} SU_i$
Caso 1	5093057	28340	5121397
Caso 2	5041076	28520	5069596
Caso 3	5106190	28190	5134380
Caso 4	5130377	28940	5159317
Caso 5	5127604	28580	5156184
Caso 6	5244442	28460	5272902
Caso 7	5068626	28760	5097386
Caso 8	5189317	28550	5217867

Istanza 5	$\sum w_j T_j$	$\sum_{i=1}^{m} SU_i$	$\sum w_j T_j + \sum_{i=1}^{m} SU_i$
Caso 1	3011249	26265	3037514
Caso 2	3001853	26385	3028238
Caso 3	3071877	26325	3098202
Caso 4	3038923	26115	3065038
Caso 5	2998855	26535	3025390
Caso 6	3071877	26325	3098202
Caso 7	3054672	26775	3081447
Caso 8	3012483	26475	3038958

Istanza 6	$\sum w_j T_j$	$\sum_{i=1}^{m} SU_i$	$\sum w_j T_j + \sum_{i=1}^{m} SU_i$
Caso 1	3575621	23545	3599166
Caso 2	3544666	23875	3568541
Caso 3	3638704	23665	3662369
Caso 4	3662791	23695	3686486
Caso 5	3662791	23695	3686486
Caso 6	3591094	23515	3614609
Caso 7	3662791	23695	3686486
Caso 8	3629528	23665	3653193

Istanza 7	$\sum w_j T_j$	$\sum_{i=1}^{m} SU_i$	$\sum w_j T_j + \sum_{i=1}^{m} SU_i$
Caso 1	5727297	31375	5758672
Caso 2	5737420	31735	5769155
Caso 3	5751068	32065	5783133
Caso 4	5719240	32245	5751485
Caso 5	5704886	31795	5736681
Caso 6	5711607	31255	5742862

Istanza 7	$\sum w_j T_j$	$\sum_{i=1}^{m} SU_i$	$\sum w_j T_j + \sum_{i=1}^{m} SU_i$
Caso 7	5709689	32005	5741694
Caso 8	5709689	32005	5741694

Istanza 8	$\sum w_j T_j$	$\sum_{i=1}^{m} SU_i$	$\sum w_j T_j + \sum_{i=1}^{m} SU_i$
Caso 1	7476353	28775	7505128
Caso 2	7495394	29375	7524769
Caso 3	7565706	28985	7594691
Caso 4	7565706	28985	7594691
Caso 5	7491255	29135	7520390
Caso 6	7540229	29585	7569884
Caso 7	7565706	28985	7594691
Caso 8	7565706	28985	7594691

Istanza 9	$\sum w_j T_j$	$\sum_{i=1}^{m} SU_i$	$\sum w_j T_j + \sum_{i=1}^{m} SU_i$
Caso 1	2624626	22870	2647496
Caso 2	2605532	22990	2628522
Caso 3	2719174	22660	2741834
Caso 4	2719174	22660	2741834
Caso 5	2599255	22840	2622095
Caso 6	2589636	23350	2612896
Caso 7	2713703	22900	2736603
Caso 8	2713703	22900	2736603

Istanza 10	$\sum w_j T_j$	$\sum_{i=1}^{m} SU_i$	$\sum w_j T_j + \sum_{i=1}^{m} SU_i$
Caso 1	4179109	30170	4209279
Caso 2	4178109	30020	4208129
Caso 3	4204492	29690	4234182
Caso 4	4208592	30200	4238792
Caso 5	4166399	29900	4196299
Caso 6	4197706	29540	4227246
Caso 7	4299229	29420	4328649
Caso 8	4176501	28880	4205381

I risultati così ottenuti sono riassunti nell'istogramma[1] in fig. 6.1. Come si può vedere la configurazione dei parametri che prevede $LT = 25$, $NI = 400$, $ND = 50$ (Caso 5) è quella che ha fornito i migliori risultati sulle istanze del nostro testbed ed è quella che si è scelto dunque di adottare.

[1]Il numero di occorrenze delle seconde miglior soluzioni somma a 12 anziché 11 poiché nei test effettuati sull'Istanza 7, la seconda miglior soluzione è stata ottenuta con due configurazioni dei parametri.

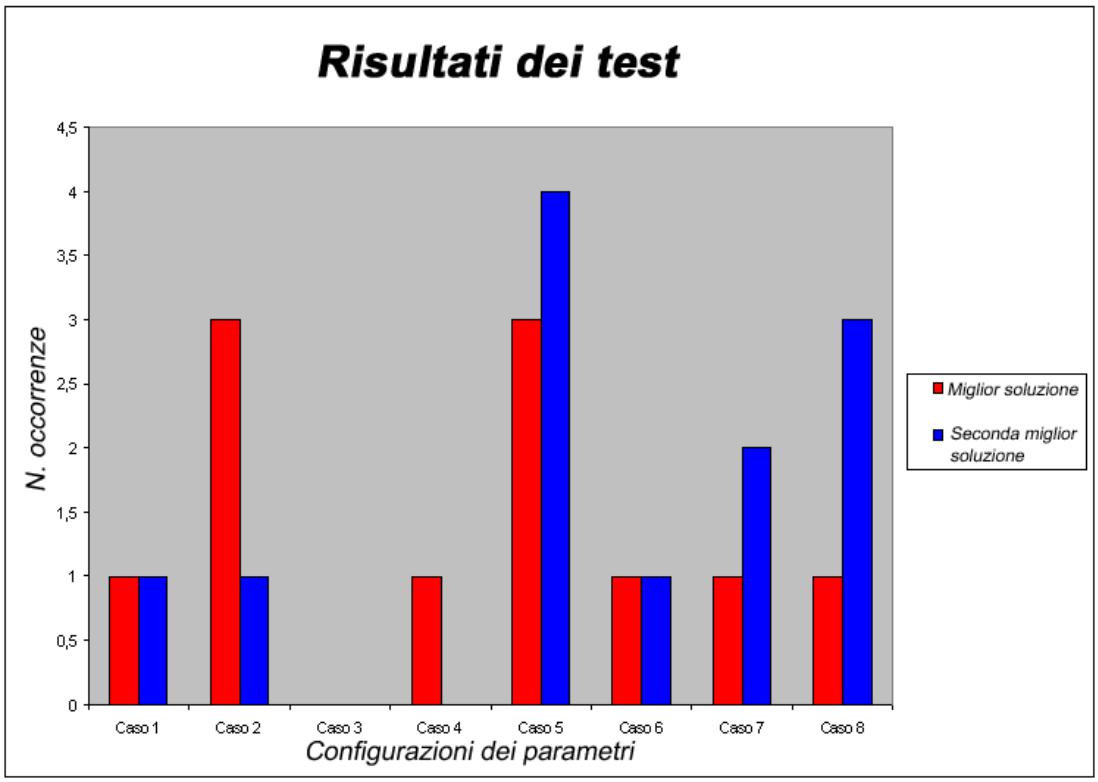

Figura 6.1: Istogramma dei risultati ottenuti dall'esecuzione dei test.

6.2 Le prestazioni dell'algoritmo

Un'analisi accurata delle prestazioni dell'algoritmo progettato si potrà effettuare solo dopo il periodo di sperimentazione a cui verrà sottoposto nei prossimi mesi durante i quali esso verrà applicato a numerose istanze reali per verificarne il suo comportamento. Per ora possiamo solamente constatare che, da un primo confronto sull'istanza reale di cui si disponeva tra la soluzione ottenuta tramite l'applicazione dell'algoritmo e la soluzione pianificata manualmente (cioè con il sistema attualmente in uso), l'algoritmo sembra poter garantire prestazioni equiparabili a quelle ottenibili attraverso una pianificazione eseguita da un personale qualificato.

L'algoritmo è stato applicato all'istanza reale, e nell'intervallo di 2 ore di tempo durante il quale il programma è restato in esecuzione, esso ha alternato per 37 volte la fase di intensificazione e diversificazione della procedura di Tabu Search, compiendo un totale di 16600 mosse (la fase di diversificazione è stata applicata 36 volte). Nel grafico in fig. 6.2 è riportato l'andamento della traiettoria seguita dall'algoritmo durante le prime 2000 mosse compiute.

Nel grafico sono chiaramente distinguibili i punti in cui l'algoritmo si trova in una fase di diversificazione (aree del grafico dove il valore della f.o. cresce rapidamente) e quelli in cui sta applicando una fase di intensificazione (aree del grafico dove il valore della f.o. dopo essere sceso rapidamente da un picco "sembra" rimanere costante). Si può notare anche la differenza tra i due metodi che sono utilizzati per implementare la fase di diversificazione e che vengono continuamente alternati dall'algoritmo.

Per apprezzare maggiormente come avviene l'esplorazione dello spazio delle soluzioni durante le fasi di intensificazione della procedura possiamo fare riferimento alla fig. 6.3. Essa mostra come varia il valore della funzione obbiettivo delle soluzioni esplorate durante la prima fase di

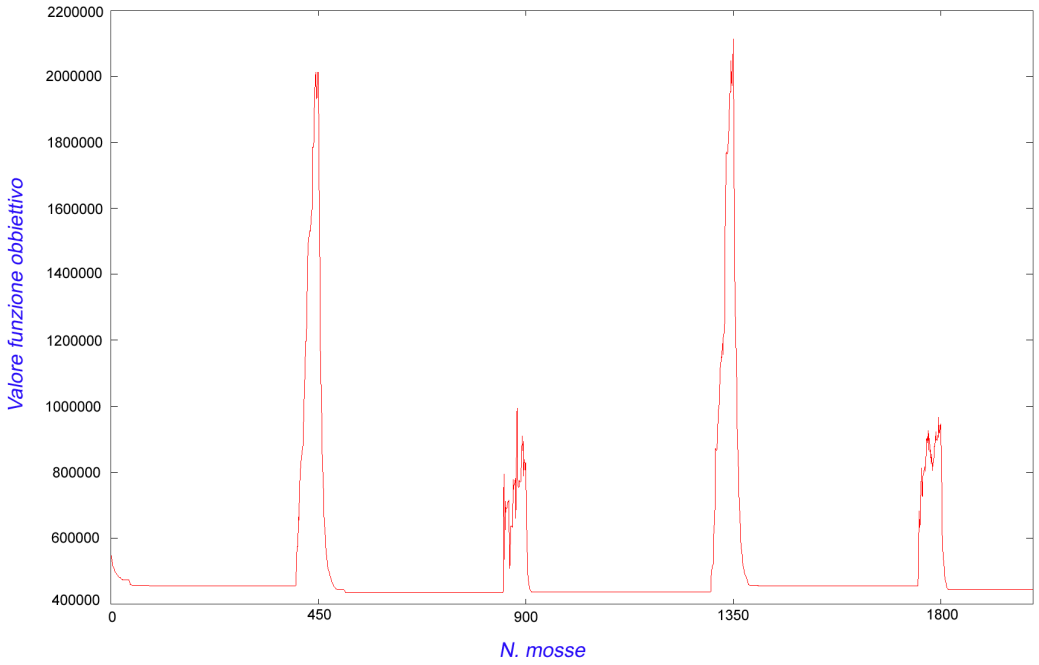

Figura 6.2: Grafico delle soluzioni visitate dall'algoritmo nelle prime 2000 mosse.

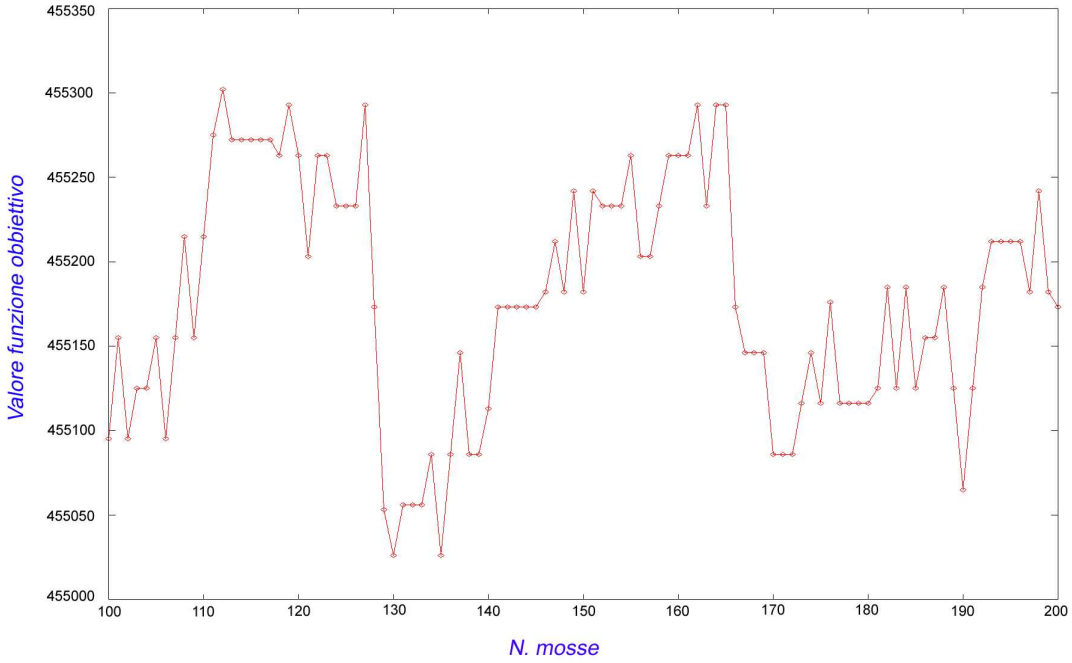

Figura 6.3: Grafico delle soluzioni visitate dall'algoritmo durante la prima fase di intensificazione.

intensificazione e precisamente tra la mossa n. 100 e la mossa n. 200. Si può notare proprio dall'andamento del grafico la caratteristica tipica della tecnica di Tabu Search che, permettendo l'applicazione anche di mosse peggiorative, riesce a non rimanere intrappolata nei punti di minimo "locale" che incontra durante l'esplorazione dello spazio delle soluzioni ammissibili.

Per l'istanza reale disponevamo, come detto, della pianificazione delle commesse sulle presse che era stata applicata dal personale dell'azienda. Abbiamo quindi potuto calcolare i valori di questa pianificazione (che indicheremo come "pianificazione manuale") relativi alla funzione obbiettivo scorporata nelle sue due componenti ($f = \sum w_j T_j + \sum_{i=1}^{m} SU_i$). La seguente tabella riassume il risultato del confronto effettuato tra la "pianificazione manuale" e la pianificazione ottenuta applicando il software sviluppato (che indicheremo come "pianificazione con software") all'istanza reale dei dati:

	Pianificazione manuale	**Pianificazione con software**
$\sum w_j T_j$	379345	376067
$\sum_{i=1}^{m} SU_i$	21180	20670
f.o. $f = \sum w_j T_j + \sum_{i=1}^{m} SU_i$	400525	396737

Come si vede la soluzione ottenuta dall'algoritmo progettato è migliore della pianificazione ottenuta manualmente rispetto ai criteri di ottimalità definiti nell'algoritmo stesso. Va detto tuttavia che la soluzione pianificata manualmente tiene conto anche di alcune informazioni che purtroppo non sono ancora disponibili nei database di sistema e che quindi non possono essere utilizzate dall'algoritmo. In particolare come abbiamo spiegato nella sezione 1.8 nella descrizione delle operazioni di cambio-lavorazione, un parametro importante che può influenzare la sequenza con la quale le commesse sono pianificate su di una pressa, è la temperatura a cui bisogna mantenere il cilindro di plastificazione della pressa durante lo stampaggio. Passare infatti da una resina termoplastica caratterizzata da un'elevata temperatura di fusione, ad una con una temperatura molto inferiore può costare anche qualche ora di attesa durante le operazioni di cambio-stampo ad aspettare che il cilindro della pressa si raffreddi. Tuttavia l'informazione sulla temperatura di fusione caratteristica di ogni materiale non è presente al momento nei database aziendali e non può dunque essere utilizzata dal nostro programma.

Per questo motivo l'algoritmo che calcola i tempi di setup tra due lavorazioni consecutive (descritto in fig. 5.8) è in realtà incompleto; esso tiene conto solamente delle informazioni che sono attualmente a disposizione nei database aziendali. In definitiva dunque la pianificazione ottenuta manualmente è migliore della pianificazione ottenuta con l'utilizzo del nostro programma per quanto concerne i tempi di setup reali, soprattutto grazie alle informazioni aggiuntive che sono a conoscenza del personale e che non sono ancora attualmente registrate nei database.

Per quanto concerne invece il contributo dovuto alla somma del *weighted tardiness* di ogni commessa ($\sum w_j T_j$) al valore finale della funzione obbiettivo, l'algoritmo sviluppato dimostra di poter ottenere degli ottimi risultati, battendo il risultato ottenuto con la pianificazione manuale. Anche in questo caso bisogna tuttavia fare qualche precisazione: un ritardo di qualche ora o persino di un giorno sulla data di consegna prevista per una commessa è del tutto ininfluente ed è preferibile dunque risparmiare un paio di ore sui tempi di setup (rallentando il meno possibile la produzione ed incrementando l'efficienza degli impianti) piuttosto che risparmiare un giorno sui ritardi delle consegne. Questa filosofia porterà sicuramente ad un settaggio diverso dei "pesi" (per ora assunti unitari) dei due obbiettivi che concorrono a formare la nostra funzione obbiettivo reale. Solo l'applicazione ad una vasta gamma di istanze reali dell'algoritmo permetterà di calibrare al meglio anche questi parametri e di ottenere dunque soluzioni di qualità paragonabile a quelle ottenibili con il sistema attuale di pianificazione.

Conclusioni

In questa tesi si è affrontato un problema di scheduling su macchine parallele nel contesto aziendale dello stampaggio ad iniezione di resine termoplastiche. Dopo un'accurata analisi dei componenti coinvolti nel processo di produzione e di pianificazione si è potuto classificare il problema secondo la notazione matematica proposta in [13].

Data la complessità intrinseca del problema, si è scelto di progettare un algoritmo euristico che fosse in grado di fornire "buone" soluzioni al problema in tempi ragionevoli. L'algoritmo si articola in due fasi.

Nella prima fase viene costruita una pianificazione iniziale utilizzando la fase costruttiva della procedura euristica GRASP e applicando come funzione di tipo greedy per l'ordinamento dei job la regola di carico ATCS.

Nella seconda fase viene applicata la tecnica euristica di Tabu Search che, partendo dalla pianificazione iniziale ottenuta dopo la prima fase dell'algoritmo, esplora lo spazio delle soluzioni ammissibili alla ricerca della pianificazione migliore.

L'algoritmo è stato quindi implementato tramite l'utilizzo del linguaggio di programmazione C++; una suite di test effettuata su 11 istanze di dati, di cui una reale e 10 generate in maniera random, ha permesso di calibrare in maniera opportuna alcuni parametri dell'algoritmo. Dal confronto effettuato sull'unica istanza reale di dati a disposizione tra la soluzione ottenuta con l'impiego del software progettato e la pianificazione redatta dal personale specializzato dall'azienda si possono apprezzare i buoni risultati, rispetto ai criteri di ottimalità definiti, ottenuti dall'algoritmo progettato che dimostrano la bontà di quest'ultimo. Come detto un'indicazione più precisa sull'efficacia dell'utilizzo del programma sviluppato si potrà avere solo al termine del periodo di sperimentazione al quale esso verrà sottoposto nei prossimi mesi.

Bibliografia

[1] K. R. Baker, *Introduction to Sequencing and Scheduling*, John Wiley & Sons, New York 1974.

[2] R. Battiti, *Reactive Search: Toward Self-Tuning Heuristics*, Modern Heuristic Search Methods, John Wiley & Sons Ltd., pp. 61–83, Chichester 1996.

[3] R. Battiti, G. Tecchiolli. *The reactive tabu search*, ORSA Journal on Computing, 6(2) pp. 126–140, 1994.

[4] A.Brandolese, A. Pozzetti, A. Sianesi, *Gestione della produzione industriale*, Hoepli, Milano 1991.

[5] R. Durante, *Le Apparecchiature Ausiliarie nella Trasformazione dei Termoplastici,* Piovan Plasticstechnologies, Padova 2001.

[6] T. Feo, M. Resende, *A Probabilistic Heuristic for a Computationally Difficult Set Covering Problem*, Operations Research Letters, vol. 8 pp. 67-71, 1989.

[7] T. Feo, M. Resende, *Greedy randomized adaptive search procedures*, Journal of Global Optimization, 6 pp. 109–133, 1995.

[8] F. Glover, *Tabu search–Part I*, ORSA Journal on Computing, 1 pp. 190–206, 1989.

[9] F. Glover, *Tabu search–Part II*, ORSA Journal on Computing, 2 pp. 4–32, 1990.

[10] L. Lamport, *LATEX: A Document Preparation System (2nd edition)*, Addison-Wesley Reading, Massachusetts 1994.

[11] Y.H. Lee, K. Bhaskaran, M. Pinedo, *A Heuristic to Minimize the Total Weighted Tardiness with Sequence Dependent Setups*, IIE Transactions, 29 pp. 45–52, 1997.

[12] S. Lippman, J. Lajoie, *C++ Corso di programmazione (Terza Edizione)*, Addison Wesley, Milano 2000.

[13] M. Pinedo, *Scheduling: theory, algorithms, and systems (2nd edition)*, Prentice Hall, NJ 2002.

[14] J.L. Viescas, *Microsoft Office Access 2003 Inside Out*, Microsoft Press, Washington 2004.