UNIVERSITÀ DEGLI STUDI DI PADOVA

DIPARTIMENTO DI INGEGNERIA INDUSTRIALE

DIPARTIMENTO DI TECNICA E GESTIONE DEI SISTEMI INDUSTRIALI

CORSO DI LAUREA MAGISTRALE IN INGEGNERIA MECCANICA

Tesi di Laurea Magistrale in Ingegneria Meccanica

MODELLO PER L'OTTIMIZZAZIONE DELLE LOGICHE DI PRELIEVO PER L'ASSERVIMENTO DI UN SISTEMA DI ASSEMBLAGGIO

Relatore

Prof. Maurizio Faccio

Laureando

Andrea Raise

Anno Accademico 2018-2019

Sommario

La seguente tesi si pone lo scopo di generare un modello di riferimento per la risoluzione di un problema di natura duale che contrappone due funzioni obiettivo: minimizzare il tempo medio di prelievo degli articoli dal magazzino e ridurre lo spazio occupato in linea. Un problema di questo tipo non può essere affrontato con metodi convenzionali perché non è possibile calcolare tutte le soluzioni possibili delle combinazioni articoli-UDC per poter determinare quale sia la migliore. Si deve ricorrere agli algoritmi genetici, una categoria di algoritmi che prende spunto dai fenomeni biologici che determinano la variazione dei geni, per poter risolvere il problema in tempi computazionali ragionevoli. L'output dell'algoritmo genetico dovrà essere una curva di *trade-off* che viene denominata frontiera di Pareto. La frontiera mostra il set di soluzioni ottime in grado di rispettare i vincoli assegnati. Con una funzione di selezione è poi possibile decidere quale soluzione possa essere considerata la migliore.

Indice

Introduzione

Nell'ambito di ogni azienda manifatturiera, riveste un ruolo fondamentale il rifornimento delle linee di produzione. L'operazione di asservimento è fortemente legata al sistema di gestione della produzione. Uno dei modelli più influenti a partire dagli anni '80 è stato quello della Produzione Snella (*Lean Production*). Il termine fu coniato da Krafcik proprio per indicare gli interventi aziendali atti a contenere l'utilizzo di risorse (materiali, ore-uomo, ore-macchina, spazi, ecc.), pur mantenendo un'elevata varietà e qualità dei prodotti. I principi cardine sono quattro (De Toni e Panizzolo, 2018):

1) Produzione totalmente sincronica
   Sia l'assemblaggio che la fabbricazione devono avvenire in condizioni ripetitive, con la fabbricazione che deve seguire il ritmo imposto dall'assemblaggio. Questa concezione si oppone alle eccessive scorte intermedie che vengono utilizzate come polmone tra tali due fasi di produzione, evitando di rallentare il flusso dei prodotti. Questo stesso principio può essere applicato anche all'esterno dello stabilimento, in ottica di fornitura. In quest'ultimo caso vengono però richiesti rapporti di corrispettiva fiducia non semplici da ottenere.

2) Logica a trazione (*pull*)
   La produzione, per quanto possibile, deve rispettare le cadenze del mercato. Rispettando l'approccio *Just in Time* (JIT), l'azienda deve produrre quanto richiesto dai clienti senza creare scorte di prodotti finiti per sopperire a mancanze in flessibilità. L'obiettivo primario è quello di ridurre le scorte e i materiali in lavorazione, ovvero il *Work in Progress* (WIP). I tempi di attraversamento (*Lead Time*) vengono dunque ridotti ai minimi termini. Tale condizione richiede una flessibilità dinamica molto elevata, accompagnata da tempi di set-up molto bassi.

3) Miglioramento continuo (*kaizen*)
   La mentalità Lean si basa su uno standard dinamico, caratterizzato da processi e prodotti continuamente perfezionabili. L'obiettivo da perseguire è quello di un miglioramento continuo e incrementale, che si oppone allo standard statico fordistico e al concetto di rivoluzione produttiva: è meglio compiere tanti piccoli passi in avanti, piuttosto che compiere pochi salti innovativi irreversibili.

4) Auto-attivazione
   Secondo tale principio la forza lavoro deve essere vista come parte integrante attiva delle risorse produttive. I lavoratori hanno la facoltà di interrompere il flusso produttivo se riscontrano anomalie e problemi qualitativi, con l'obiettivo di bloccare le non conformità a monte, prima che esse arrivino a valle con costi maggiorati (dovuti all'impiego di risorse in fasi successive).

La *Lean Production* pone la propria attenzione sulla valutazione di tutti i processi che effettivamente danno valore ai prodotti (*value stream*), cercando dunque di limitare quelli che invece rappresentano solamente degli sprechi. Sotto quest'ottica l'operazione di rifornimento delle linee di assemblaggio, per quanto necessaria alla produzione, di fatto non partecipa all'aumento del valore del prodotto, ma incrementa il costo del processo produttivo. L'ottimizzazione dei flussi legati al rifornimento delle linee di produzione è dunque uno studio importante per arrivare ad una soluzione che permetta, da un lato, di ridurre al minimo le inefficienze dovute a mancanti di linea, dall'altro, di ridurre le risorse investite in questa operazione. Tale dunque è l'obiettivo della presente tesi, che si prefigge come scopo quello di definire un modello per l'ottimizzazione delle logiche di prelievo.
La tesi è costituita da quattro capitoli:

- Capitolo 1 – Letteratura: verrà presentata una fotografia della conoscenza scientifica attuale, andando ad individuare un campo applicativo non ancora affrontato.
- Capitolo 2 – Modello: verrà costituito un modello di riferimento per l'ottimizzazione delle logiche di prelievo per l'asservimento di un sistema di assemblaggio.

- Capitolo 3 – Caso di studio: il modello verrà applicato ad un caso reale, nella fattispecie un’azienda del campo manifatturiero.
- Capitolo 4 – Conclusioni: verranno tratte le conclusioni positive e le limitazioni del modello generato.

Capitolo 1

Letteratura

La gestione delle liste di prelievo è un argomento critico per molte realtà aziendali. I costi logistici collegati alla gestione del magazzino (accettazione, stoccaggio, prelievo, ecc.) sono spesso molto elevati. Secondo Rouwenhorst et al. (2000) i costi logistici sono già largamente determinati in fase di design del magazzino, comprensiva di specifiche tecniche, selezione delle risorse e determinazione del layout. Soprattutto negli ultimi anni è stato notato un trend crescente di articoli riguardanti l'analisi di efficienza delle attività di picking nei magazzini, come evidenziato da De Koster et al. (2017), Van Gils et al. (2018) e Davarzani e Norrman (2015). In particolare Van Gils et al. (2018) hanno notato un interesse crescente verso lo studio di più problemi di ottimizzazione in contemporanea, al contrario degli anni precedenti in cui si preferiva analizzare un solo problema alla volta. De Koster et al. (2017) hanno verificato un trend crescente nei confronti di tecnologie automatizzate di stoccaggio, come sistemi AS/R e AVS/R, e nei confronti di problematiche fino a poco tempo fa trascurate, come il fattore umano nelle operazioni di picking. Secondo Davarzani e Norrman (2015) il trend potrebbe essere fortemente legato alle richieste di cambiamenti rapidi ed in tempo reale (ad esempio per ordini di clienti con poco preavviso) e soprattutto all'evoluzione dell'e-commerce.

1.1 Caratterizzazione di un magazzino

La caratterizzazione di un magazzino si articola in tre diverse analisi: processi, risorse e organizzazione (Rouwenhorst et al., 2000). I processi rappresentano il flusso di materiale all'interno del magazzino:

- L'accettazione consiste nel primo processo col quale il materiale entra nel magazzino (ciò può avvenire con fornitura esterna o interna). I materiali devono spesso essere controllati e posti in contenitori diversi da quelli di arrivo.
- Con lo stoccaggio i materiali vengono piazzati nelle unità di stoccaggio. L'area può essere suddivisa in due aree distinte, che sono la *reserve area*, per lo stoccaggio dei materiali in forma di pallet (o di qualunque forma economica), e la *forward area*, che funge da *supermarket* e permette una rapida operazione di picking (i materiali sono già stoccati nella forma più comoda, solitamente su scaffalature, per l'operazione di rifornimento). La reintegrazione del supermarket avviene tramite il trasferimento di materiali dall'area di stoccaggio di massa. Tale operazione è chiamata *Replenishment*.
- Il prelievo avviene tramite degli ordini di prelievo, compilati su richiesta dei processi di valle (rifornimento delle linee produttive, ordini clienti). Gli articoli prelevati possono quindi essere smistati o raggruppati in base alle liste di prelievo.
- La fase di spedizione è l'ultima fase e riguarda tutti gli articoli che devono essere inviati ai clienti. In questa fase solitamente sono necessarie operazioni di controllo qualità, imballaggio, caricamento sui mezzi di trasporto.

Le risorse di un magazzino possono essere catalogate come:

- Unità di stoccaggio, ovvero la modalità o forma con la quale un articolo viene messo a magazzino (pallet, scatole, contenitori di plastica, ecc.).
- Sistema di stoccaggio, che indica la struttura utilizzata (scaffalature, magazzini automatici, ecc.).
- Equipaggiamento, ovvero l'attrezzatura per il picking e per la spedizione, tra cui carrelli elevatori, transpallet, tugger train, sistemi di pallettizzazione, ecc. Nell'equipaggiamento rientrano anche gli ausiliari come i palmari o i lettori bar code.
- Warehouse Managment System (WMS), che consiste nel sistema informatico in appoggio alla gestione del magazzino. Può essere utilizzato non solo per inventario o verifica degli articoli, ma anche per la gestione delle liste di prelievo.

- Il personale, la risorsa più flessibile e dalla quale dipende maggiormente l'efficienza del magazzino.

Infine vi è l'organizzazione del magazzino, scelte gestionali da effettuare in fase di design che determinano il flusso di processo:

- Organizzazione dei processi.
- Assegnazione delle baie, da cui o verso cui viene portato il materiale.
- Politiche di stoccaggio, per scegliere come gestire l'allocazione delle merci nel magazzino. Ne esistono molte, tra cui gestione dedicata, gestione randomica, gestione con clusterizzazione (per esempio classi ABC, famiglie di prodotti, stoccaggio correlato, ecc.). Bisogna stabilire anche la politica di gestione di un eventuale *forward area*.
- Gestione degli addetti al prelievo, assegnazione delle liste di prelievo, generazione delle routes da seguire, raggruppamento degli ordini di prelievo (ordini gestiti singolarmente o raggruppati tramite batch picking). Se gli ordini di prelievo sono raggruppati allora gli articoli devono essere poi smistati opportunamente.
- Gestione dell'equipaggiamento inutilizzato o a riposo.

1.2 Progettazione del magazzino

Rouwenhorst et al. (2000) e Van Gils et al. (2018) suddividono le decisioni relative alla progettazione di un magazzino in tre diverse categorie:

- Decisioni strategiche
- Decisioni tattiche
- Decisioni operative

1.2.1 Decisioni strategiche

Le decisioni strategiche si riferiscono ad un orizzonte temporale ampio e sono legate al flusso di processo, al grado di automazione ed al layout dell'area di stoccaggio. Solitamente sono scelte che determinano un elevato valore di investimento e possono essere studiate in base a due problemi principali, che

sono specifiche tecniche e valutazioni economiche. Con specifiche tecniche si intendono l'unità di stoccaggio, il sistema di stoccaggio e l'equipaggiamento. In input ci sono le caratteristiche dei prodotti e gli ordini. In output vi sono le configurazioni possibili dei sistemi di stoccaggio in grado di soddisfare i vincoli impostati. Una volta ottenuta questa lista di possibili alternative, la migliore va presa sulla base di valutazioni economiche (valore investimenti e costi operativi stimati). Bisogna tenere in considerazione che scelte effettuate a questo livello influiscono moltissimo sulle altre due categorie perché immettono dei vincoli e dei requisiti ai livelli più bassi di analisi.

1.2.2 Decisioni tattiche

Le decisioni tattiche sono relative al medio termine e riguardano principalmente il dimensionamento delle risorse (dimensionamento del sistema di stoccaggio, numero di operatori, allocazione delle merci) sulla base delle scelte effettuate al livello precedente. Le scelte includono il dimensionamento del sistema di stoccaggio (comprese le aree di picking), le logiche di rifornimento e di stoccaggio (random, stoccaggio dedicato, famiglie di articoli), la quantità di equipaggiamenti e asservitori richiesti. In questo caso bisogna considerare non solo vincoli economici, ma anche vincoli legati all'efficienza ed alla capacità del magazzino. Le principali logiche di stoccaggio sono evidenziate da Chackelson et al. (2013). La random storage consiste nell'assegnazione dinamica delle celle di stoccaggio (scelte tra quelle a disposizione) alla merce in entrata; ne consegue che si tende ad usare molto meglio lo spazio a disposizione, ma aumenta il tempo necessario all'identificazione della posizione degli articoli, nonché il tempo necessario per le operazioni di picking. La Closest open location storage consiste nello stoccare l'articolo in entrata nel primo slot libero dalla baia di ingresso: ne consegue che vengono usate di più le celle posizionate vicino all'ingresso del magazzino. La dedicated storage si basa sull'assegnazione statica dei posti pallet e delle unità di carico. Il grande vantaggio di questa logica risiede nella suddivisione degli articoli in posti facilmente memorizzabili, ma aumenta lo spazio richiesto con conseguente peggior utilizzo dell'area del magazzino. Una via di mezzo è rappresentata dalla class-based storage, con la quale gli articoli sono raggruppati in classi di consumo (derivanti dalle analisi ABC); viene ridotto il tempo di picking riunendo in posti adiacenti gli articoli con maggiore frequenza di prelievo. Un'altra alternativa è il family grouping, che considera le

relazioni che possono esistere tra alcuni articoli, andando dunque a suddividerli in famiglie (ad esempio articoli appartenenti ad uno stesso prodotto finito, articoli appartenenti ad una stessa tipologia di materiale di cui sono formati).

1.2.3 Decisioni operative

Le scelte operative devono essere fatte in merito al breve termine. Dati i vincoli già esistenti dettati dai livelli più elevati, le logiche operative sono meno legate tra loro e possono essere affrontate in modo indipendente. Esse principalmente riguardano il controllo e la gestione: assegnazione dei carichi, assegnazione delle merci in entrata, raggruppamento e sequenziamento degli ordini, scelta delle picking routes, scelta delle aree di riposo per l'equipaggiamento non utilizzato. Tra le logiche di prelievo più note vi sono la S-shape, la return strategy, midpoint strategy, largest gap strategy, composite strategy e la optimal routing. La logica S-shape (o anche nota come traversal) è probabilmente la più semplice: l'operatore percorre tutti i corridoi del magazzino in cui deve prelevare almeno un articolo entrando da un lato e uscendo dall'altro. La return consiste nell'entrare ed uscire dallo stesso lato di un corridoio. La midpoint crea un punto centrale in ogni corridoio e l'operatore non può superarlo (per prendere un articolo che sta oltre tale punto di divisione, deve accedere dal lato opposto del corridoio). La largest gap è simile alla midpoint, ma l'operatore accede ad un corridoio scegliendo il lato di ingresso in base alla distanza massima tra gli articoli da prelevare. La composite strategy nasce dall'integrazione tra return e S-shape (Petersen, C. G., 1997). La logica ottimale consiste in una combinazione tra le logiche viste in precedenza con l'obiettivo di minimizzare il tempo di picking (Ratliff, H.D., & A.S. Rosenthal, 1983).
In Tabella 1.1 è possibile visualizzare una panoramica dei vari livelli di progettazione di un magazzino.

Tabella 1.1: *Panoramica dei tre livelli di dettaglio dei problemi che nascono durante la progettazione di un magazzino.*

Categoria di decisioni	Orizzonte temporale	Livello di dettaglio	Principali problemi di progettazione
Livello strategico	Lungo	Basso	Selezione equipaggiamento Livello di automazione Layout dell'area di stoccaggio
Livello tattico	Medio	Medio	Scelta delle zone di picking Assegnazione degli articoli nelle zone Allocazione delle merci Flusso ordini attraverso le picking zones Gestione smistamento articoli degli ordini
Livello operativo	Breve	Alto	Raggruppamento degli ordini Routes di prelievo Gestione operatori Livellamento dei carichi Assegnazione degli ordini

1.3 Indicatori di performance

Gli indicatori di performance vengono suddivisi da Staudt et al. (2015) in "indicatori di performance diretti" (*hard metrics*) ed "indicatori di performance indiretti" (*soft metrics*). Con i primi si intendono quei parametri che possono essere quantificati con espressioni matematiche non troppo complicate; ad esempio possono riferirsi al tempo del ciclo degli ordini o ai costi. I secondi sono invece parametri che rappresentano valutazioni spesso soggettive o comunque qualitative e devono essere analizzati con sofisticati strumenti, come analisi di regressione, logica sfumata, Data Envelopment Analysis (DEA) o Structured Equations Model (SEM); ad esempio possono riferirsi alla soddisfazione dei clienti o alla loro fedeltà.

1.3.1 Indicatori di performance diretti

Gli indicatori di performance diretti sono quattro e sono comunemente impiegati nell'industria. Essi sono:

- Tempo

- Qualità (“Servizio” per Van Gils et al. (2018))
- Costo
- Produttività

Gli indicatori di tempo riguardano principalmente il *lead time* totale dell’ordine (dalla richiesta del cliente fino all’evasione), il tempo di scarico, il tempo di completamento della fase di prelievo dell’ordine (*order picking time*) e il *lead time* di consegna al cliente. Altri indicatori meno rilevanti sono il tempo di stoccaggio, il lead time di attesa, il tempo di spedizione ed il tempo di *downtime* dell’equipaggiamento. In modo più semplicistico Van Gils et al. (2018) considerano solo *order picking time* e precocità/ritardo (*earliness/tardiness*). Quest’ultimo indicatore rappresenta la differenza tra il tempo richiesto per completare le operazioni di picking dell’ordine e il tempo massimo a disposizione per l’evasione dell’ordine (che dipende dalla data comunicata al cliente).
Gli indicatori di qualità riguardano la puntualità, il livello di completamento dell’ordine, la correttezza del picking, gli stock-out e la soddisfazione del cliente. La puntualità è quello più rilevante per Staudt et al. (2015), mentre Van Gils et al. (2018) considerano un più generico “livello di servizio”.
Gli indicatori di costo sono relativi a costi inventariali, costi di emissione dell’ordine, costi del personale ed equipaggiamento, costi di mantenimento (edifici, equipaggiamento). I più importanti riguardano gli *inventory costs*, perché generalmente considerati i più delicati dai managers, dal momento che considerano al loro interno tutti i costi del magazzino.
Gli ultimi sono gli indicatori di produttività, che indicano l’efficienza di utilizzo delle risorse. I più importanti sono il “throughput”, cioè la capacità rotazionale effettiva del magazzino, e la produttività del lavoro, inteso come il rapporto tra il numero di articoli movimentati e le ore necessarie per farlo. Atri indicatori meno importanti riguardano l’efficienza delle spedizioni, l’utilizzo dello spazio, l’efficienza della fase di picking, la capacità di rotazione in forma economica.

1.3.2 Indicatori di performance indiretti

Al contrario degli indicatori diretti, quelli indiretti necessitano di strumenti matematici complessi. Le performance sono ricavate utilizzando informazioni tratte da database.

L'indicatore principale è la performance della forza lavoro, che influisce sul lead time degli ordini e dei prelievi e dunque può compromettere il livello di servizio percepito dal cliente. Altri indicatori sono: le attività *Value Adding Logistic* (VAL), inteso come la quantità di attività che effettivamente comportano un elevato aumento del valore logistico (sterilizzazione, assemblaggio finale, installazione), rispetto a quelle che, invece, comportano un aumento meno evidente (kitting, aggiunta manuali, etichettature); l'*Inventory Management*, correlato sempre di più con il livello di informatizzazione ed automazione del magazzino; il livello di automazione del magazzino, legato all'utilizzo di particolari tecnologie, come *Radio-Frequency Identification* (RFID), lettori barcode e robot; il livello di percezione di soddisfazione del cliente, parametro molto importante per quei magazzini molto orientati alle richieste dei clienti; la manutenzione del magazzino.

1.4 Metodi di risoluzione

Van Gils (2018) identifica tre categorie di metodi per valutare l'interazione o l'integrazione tra diverse logiche di prelievo. Essi sono:

- Modelli analitici
- Simulazioni
- Programmazione matematica

I modelli analitici consistono in un set di equazioni matematiche in grado di approssimare il comportamento di un sistema in base ad alcuni parametri. Le simulazioni sono metodi atti ad emulare il comportamento di un sistema e si basano su prove numeriche. La programmazione matematica porta alla generazione di modelli di riferimento costituiti da espressioni caratterizzate da funzioni obiettivo e vincoli che si adattano al campo di impiego.
Per quanto i modelli analitici siano il metodo migliore per prevedere il comportamento di un sistema e per ridurre al minimo i tempi computazionali di calcolo, essi sono difficili da sviluppare. Al contrario le simulazioni, oltre ad essere applicazioni molto settoriali, necessitano di sviluppare uno scenario adeguato e ripetitive run per ridurre gli effetti stocastici, ma sono più semplici da sviluppare.

1.5 Stato dell'arte

In questo paragrafo viene effettuata un'analisi degli articoli utilizzati per la catalogazione della letteratura odierna. Come evidenziato da Van Gils et al. (2018), risulta di primaria importanza cercare di affrontare più problemi, legati alla progettazione dei sistemi di picking, contemporaneamente. Questo perché cercare di ottimizzare un solo problema alla volta porta alla determinazione di soluzioni subottimali e non necessariamente adatte alla totale attività di stoccaggio. Per aumentare l'efficienza di queste soluzioni è necessario tentare di affrontare più problemi insieme e non individualmente o consequenzialmente. Inoltre i problemi di natura strategica non vengono affrontati perché ritenuti già stabiliti in fase di progettazione del magazzino. Questa è una considerazione che si adatta molto bene alla situazione industriale generica, in cui di solito un'azienda pone vincoli molto precisi sul layout di un magazzino o sull'equipaggiamento a disposizione. Le decisioni di tipo strategico sono quasi sempre già fissate. Al contrario di maggiore rilevanza sono le decisioni di tipo tattico ed operativo perché più adatte a situazioni di medio-breve termine e più semplici da applicare a situazioni reali. Inoltre gli articoli consultati riguardano tutti lo studio di magazzini manuali con layout tradizionali, quindi costituiti da corridoi di picking tra loro paralleli e qualche corridoio trasversale che divide il magazzino in blocchi.

Tutti gli autori degli articoli consultati concordano nel valutare l'efficienza delle operazioni di picking tramite degli indicatori di tempo, soprattutto il tempo di picking dell'ordine ed *earliness/tardiness*. La scelta di utilizzare il tempo come parametro principale non è casuale, ma è figlia di una semplice considerazione basata sul fatto che il tempo perso per inefficienza si può facilmente tradurre in costi tramite dei costi fissi prestabiliti (ovviamente si tratta di costi *time-related*, ovvero dipendenti dal tempo, come ad esempio il costo degli operatori). Inoltre un'analisi dei tempi dettagliata può portare a comprendere più facilmente dove avvengono le perdite di efficienza e come conciliare i tempi effettivi con quelli pianificati. Gli alti costi delle attività di picking discendono principalmente (all'incirca il 50%) dalle attività di movimentazione (De Santis et al., 2018), quindi operazioni non produttive.

Caron et al. (1998) hanno scelto di utilizzare un approccio basato su un modello matematico che considerasse le logiche di routing (*traversal* e *return*) e di allocazione delle merci, tramite il metodo Cube-Per-Order Index (COI). Il COI indica il rapporto tra spazio richiesto di stoccaggio e frequenza di prelievo. Il

tempo è stato suddiviso in tre categorie: travel time, processing time e administrative time. Il travel time è quello più significativo e dipende dalle distanze in gioco. Tra le ipotesi di lavoro di base risultano l'horizontal travel time (non è stato considerato il movimento verticale degli abbassamenti per il picking), un layout rettangolare a due blocchi e corridoi stretti (approssimabili con delle linee dritte). La logica del traversal routing generalmente supera in performance la return routing, con le uniche eccezioni legate ad un numero basso di prelievi per corridoio (inferiore ad 1). Questo perché con più prelievi per corridoio è più probabile che sia inferiore lo spazio rimanente per percorrere la parte conclusiva del corridoio, piuttosto che la distanza necessaria per tornare indietro. In maniera differente Peterson e Schmenner (1999) hanno valutato vantaggiosa la combinazione di stoccaggio perimetrale e largest gap routing, dal momento che tale routing tende a prediligere percorsi che seguono il perimetro. Le logiche di routing citate sono semplici, ma anche le più usate, perché permettono di ridurre il tempo computazionale. In quest'ambito l'algoritmo euristico per routing Lin-Kernigham-Helsgaun si è dimostrato affidabile (Helsgaun, 2000). Van Gils et al. (2018) hanno evidenziato come finora non sia stata posta sufficiente attenzione all'analisi di ulterior vincoli di picking, relativi soprattutto a restrizioni di peso, forma e fragilità. Altri aspetti poco considerati sono la differenziazione del tempo di movimentazione orizzontale (per muoversi lungo e tra i corridoi) da quello verticale (tempo per prelevare gli articoli dallo slot dedicato, che dipende dalla sua posizione rispetto al pavimento). Chan e Chan (2011) hanno provato ad affrontare questo tema effettuando delle simulazioni. L'allocazione è stata fatta per classi: random, dedicata e class-based. Le classificazioni sono state effettuate con analisi COI e logiche di stoccaggio a classi Entry-Item-Quantity (EIQ), quest'ultima basata sulla quantità e frequenza di articoli e numero di codici richiesti da ogni cliente. La performance di sistemi class-based varia con la densità di prelievo.

La combinazione di problemi di allocazione delle merci e order batching sono state affrontate da Ho e Tseng (2006), che hanno verificato come una logica di seed order che puntasse a minimizzare il numero di corridoi funzionasse positivamente con un'assegnazione di stoccaggio basata su un turnover degli articoli da effettuare all'interno degli stessi corridoi. Si sceglie come ordine base (seed) quello che permette di prelevare dal minor numero di corridoi, poi si aggiungono gli ordini in modo da non aumentare eccessivamente il numero di corridoi da visitare. Secondo Van Gils et al. (2018) la citata combinazione risulta essere la migliore al lordo di possibili congestioni nei corridoi.

La gestione della congestione è stata affrontata da Chen et al. (2016) nell'affrontare la combinazione di problemi di workforce level e routing, attraverso l'implementazione dell'algoritmo A-MOP-NPT, che utilizza l'ottimizzazione ACO per la parte di routing e tiene in considerazione il numero di operatori ed il possibile sovraffollamento dei corridoi. Tale metodo risulta essere particolarmente utile per più blocchi, ma non considera il carico di lavoro degli operatori (la loro capacità) e il problema del batching degli ordini.
La combinazione di order batching e routing presenta due problemi appartenenti alla categoria delle decisioni operative e come tali dovrebbero essere analizzate contemporaneamente (Van Gils et al., 2018). Cheng et al. (2015) hanno sviluppato un algoritmo ibrido composto da Particle Swarm Optimization (PSO) e da ACO per un sistema di prelievo manuale. I risultati sono risultati essere migliori dell'alternativa 2-Phase Genetic Algorithm (GA) di Tsai et al. (2008), basato sulla valutazione dell'order batching in una prima fase e solo in seguito la valutazione della distanza minima da percorrere. Il modello si basa su quello usato da Kulak et al. (2012) con l'obiettivo di integrare il PSO, utilizzato per la parte di batching, con l'ACO, impiegato per la parte di routing (minimizzare le distanze). Questi due problemi potrebbero non essere sufficienti con più operatori addetti al prelievo: in tal caso risulta più utile inserire nel problema anche l'assegnazione dei carichi. Quest'ultimo problema è stato affrontato da Chen et al. (2015) tramite un algoritmo meta-euristico integrante l'hybrid-coded genetico, impiegato per cercare la sub-ottima soluzione per batching e sequencing, e l'ACO. L'obiettivo dell'algoritmo è di ridurre il tardiness, utilizzato perché ritenuto più rilevante allo scopo di evitare i ritardi di consegna delle merci ai clienti o in produzione (in tal caso si parla di Internal Customer).
Chackelson et al. (2013) si sono concentrati su tre problemi contemporaneamente: storage location, batching e routing. Tali problemi sono stati affrontati con simulazione. L'allocazione è basata su una suddivisione degli articoli in classi; inoltre considerano una logica pick-by-article per diminuire i tempi di picking: consiste nel dare la precedenza al prelievo di un articolo indipendentemente dall'ordine a cui appartiene; una scelta di questo tipo implicherebbe però un'analisi della fase di sorting (gli articoli poi prelevati dovrebbero essere smistati nei singoli ordini) che non è stata effettuata. La soluzione migliore è quella del Return routing con Pick-by-article batching. Le interazioni tra i problemi tramite analisi della varianza (ANOVA) ha portato a determinare l'esistenza di una forte relazione tra le logiche di batching e la comunanza degli ordini (omogeneità degli ordini di prelievo). Non è stata invece

rilevata una relazione tra allocazione delle merci e routing. A quest'ultima affermazione sono giunti anche Ho et al. (2008) e Ho e Tseng (2006). Differiscono da tale constatazione Manzini et al. (2007) e Shqair et al. (2014).
Di seguito (Tabella 1.2) viene riportata la situazione attuale della letteratura in formato riassuntivo.

Tabella 1.2: *Situazione letteraria attuale in formato matriciale.*

Autore	N° blocchi magaz.		Metodo risoluzione			Indicatore di performance			Problema affrontato								
	2+	1	Prog. matematica	Modello analitico	Simulazione	Tempo	Costo	Produttività	Servizio	Zone location	Zone assignment	Storage location assignment	Workforce level	Workforce allocation	Job assignment	Order batching	Zone picking	Routing	Order consolidation and sorting
Bartholdi et al. (2001)								X							X		X		
Caron et al. (1998)	X			X		X						X						X	
Chackelson et al. (2013)					X	X						X				X		X	
Chan e Chan (2011)					X	X						X						X	
Chen et al. (2010)					X	X		X	X			X				X		X	
Chen et al. (2016)					X	X							X					X	
Chen et al. (2015)				X		X									X	X		X	
Chen et al. (2017)					X	X							X			X			
Cheng et al. (2015)				X		X										X		X	
Dekker et al. (2004)					X	X						X						X	
De Koster et al. (1999)					X	X						X				X		X	
De Koster et al. (2012)				X		X				X		X		X					
Dijkstra et al. (2017)						X						X						X	

	N° blocchi magaz.		Metodo risoluzione			Indicatore di performance				Problema affrontato										
Autore	1	2+	Simulazione	Modello analitico	Prog. matematica	Tempo	Costo	Produttività	Servizio	Zone location	Zone assignment	Storage location assignment	Workforce level	Workforce allocation	Job assignment	Order batching	Zone picking	Routing	Order consolidation and sorting
Ene e Öztürk (2012)					X	X										X		X	
Giannikas et al. (2017)			X			X										X		X	
Gong e De Koster (2008)				X		X			X							X		X	
Henn (2012)			X			X						X				X		X	
Henn (2015)			X			X							X		X	X		X	
Henn e Schmid (2013)			X			X									X	X		X	
Henn e Wäscher (2012)			X			X						X				X		X	
Ho et al. (2008)			X			X						X				X		X	
Ho e Tseng (2006)			X			X						X				X		X	
Hong et al. (2012)			X			X							X		X	X			X
Hong et al. (2015)						X									X		X		
Hong et al. (2016)						X		X							X		X		
Hong e Kim (2017)						X						X				X			X

Autore	N° blocchi magaz.		Metodo risoluzione			Indicatore di performance				Problema affrontato										
	1	2+	Simulazione	Modello analitico	Prog. matematica	Tempo	Costo	Produttività	Servizio	Zone location	Zone assignment	Storage location assignment	Workforce level	Workforce allocation	Job assignment	Order batching	Zone picking	Routing	Order consolidation and sorting
Hsieh e Huang (2011)			X			X		X				X				X		X	
Hsieh e Tsai (2006)			X			X						X				X		X	
Hwang et al. (2004)				X		X						X						X	
Jewkes et al. (2004)						X					X	X							
Koo (2009)						X		X							X		X		
Kulak et al. (2012)					X	X										X		X	
Li et al. (2016)					X	X										X		X	
Lin et al. (2016)					X	X										X		X	
Manzini et al. (2007)			X			X						X						X	
Matthews e Visagie (2013)					X	X									X			X	
Matusiak et al. (2014)					X	X										X		X	
Matusiak et al. (2017)					X	X									X	X			
Öncan (2015)			X			X										X		X	

	N° blocchi magaz.		Metodo risoluzione			Indicatore di performance				Problema affrontato										
Autore	1	2+	Simulazione	Modello analitico	Prog. matematica	Tempo	Costo	Produttività	Servizio	Zone location	Zone assignment	Storage location assignment	Workforce level	Workforce allocation	Job assignment	Order batching	Zone picking	Routing	Order consolidation and sorting
Pan e Wu (2012)						X						X	X					X	
Parikh e Meller (2008)							X									X	X		
Petersen (2000)			X			X			X	X						X			
Petersen e Aase (2004)			X			X						X				X		X	
Petersen et al. (2004)			X			X						X						X	
Petersen Schmenner(1999)			X			X						X						X	
Quader et al. (2016)			X			X		X				X			X		X	X	
Roodbergen et al. (2015)			X			X						X						X	
Ruben e Jacobs (1999)			X			X		X				X	X			X			
Rubrico et al. (2011)					X	X									X	X			
Scholz e Wäscher (2017)			X			X						X				X		X	
Scholz et al. (2017)					X	X							X		X	X		X	
Shqair et al. (2014)			X			X						X						X	

Autore	N° blocchi magaz.		Metodo risoluzione			Indicatore di performance				Problema affrontato							
	2+	1	Prog. matematica	Modello analitico	Simulazione	Tempo	Costo	Produttività	Servizio	Zone location	Zone assignment	Storage location assignment	Workforce level	Workforce allocation	Job assignment	Order batching	Zone picking	Routing	Order consolidation and sorting
Theys et al. (2010)					X	X						X						X	
Tsai et al. (2008)			X				X									X		X	
Valle et al. (2017)						X										X			
Van Nieuwenhuyse (2009)				X		X							X	X		X			X
Won e Olafsson (2005)			X			X										X		X	
Yu e De Koster (2008)						X		X				X		X		X			
Yu e De Koster (2009)						X				X						X			
Zhang et al. (2017)			X			X									X	X			

La tabella mostra come la maggior parte degli articoli revisionati (circa il 50%) affronti il problema tramite simulazioni. Le simulazioni sono effettivamente più semplici e rapide da sviluppare. Gli algoritmi metaeuristici risultano non adatti a valutare combinazioni di problemi di natura tattica e operativa perché relativi a differenti orizzonti di analisi. Per tali combinazioni è più opportuno ricorrere a modelli matematici o simulazioni. L'indicatore di performance più utilizzato risulta essere il tempo con 58 articoli su 60 analizzati. Sono stati invece poco analizzati costo, produttività e servizio. Per quanto riguarda i problemi di pianificazione sono 3 quelli più affrontati: locazione delle merci, raggruppamento ordini e analisi dei percorsi.

Capitolo 2

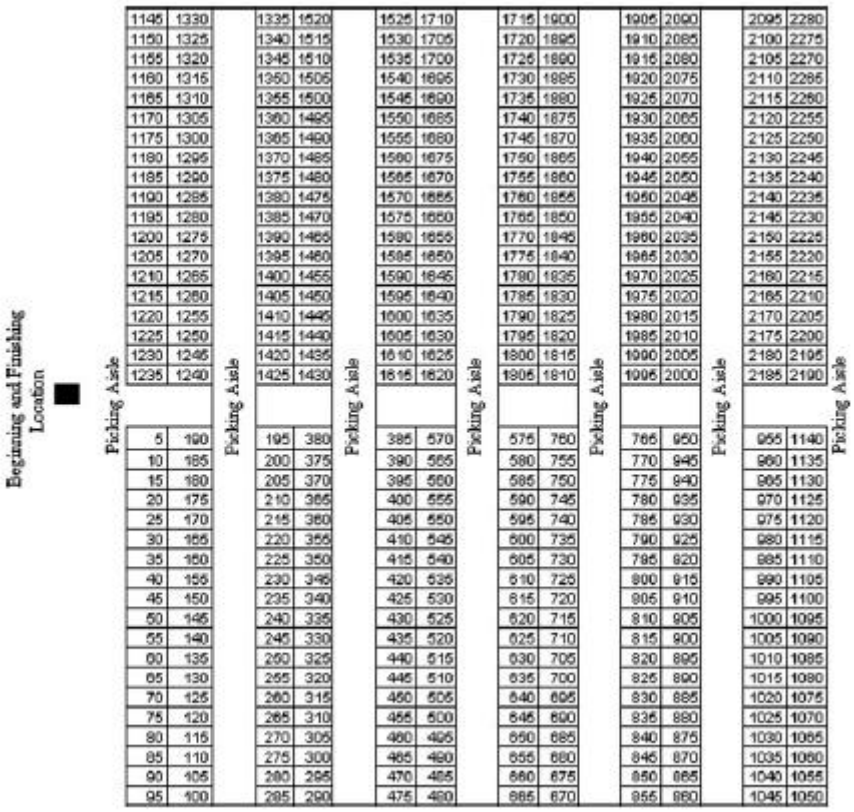
Modello matematico

Il modello matematico si pone come obiettivo quello di affrontare contemporaneamente due problemi separati: l'ottimizzazione della posizione degli articoli a magazzino e la scelta delle UDC (unità di carico) ottimali da utilizzare sulle linee di produzione e nelle stazioni di assemblaggio. Il primo problema è di natura tattica ed è relazionabile con il problema di stoccaggio delle merci a magazzino. Il magazzino considerato nel modello è di tipo manuale ed è costituito da due serie di scaffalature. Il secondo problema è sempre di natura tattica, ovvero con interesse temporale a medio termine. La scelta delle UDC è un fattore determinante perché regola la frequenza di picking degli articoli dal magazzino.

2.1 Ottimizzazione dello stoccaggio degli articoli

La prima parte del modello matematico riguarda lo stoccaggio degli articoli a magazzino. Una possibile configurazione di magazzino è rappresentabile dalla figura 2.1. È attraversato da un corridoio centrale che lo divide in due blocchi costituiti da file di scaffalature. Le scaffalature sono disposte a coppie con corridoi trasversali che permettono l'interazione con tutte le celle di magazzino.

Figura 2.1: *possibile configurazione del magazzino manuale (Ene, S. e Öztürk, N., 2012)*



I parametri principali per la composizione del magazzino sono i seguenti:

- Il numero di coppie di scaffali
- Il numero di ripiani per scaffale
- Il numero di celle di stoccaggio per ogni ripiano di uno scaffale

2.1.1 Calcolo delle distanze

La prima parte del modello si occupa della generazione delle formule per modellizzare il magazzino e le sue locazioni di stoccaggio. La formulazione matematica serve a calcolare la distanza di ogni cella a magazzino da un punto di deposito. Tale punto di deposito rappresenta l'output del magazzino e la posizione in cui gli articoli vengono considerati pronti per essere portati in linea o in stazione di assemblaggio.
La modellizzazione per il calcolo delle distanze e della posizione di stoccaggio si ispira a quella di Ene e Ozturk (2012). Vengono definite due distanze: una orizzontale ed una verticale.

$$D_h = \begin{cases} a + (i-1)\cdot(m+n) + \frac{o}{2} + h \cdot \left| k - \frac{K-1}{2} \right| & j = 0 \\ a + in + (i-1)\cdot m + \frac{o}{2} + h \cdot \left| k - \frac{K-1}{2} \right| & j = 1 \end{cases} \quad (2.1)$$

D_h rappresenta la distanza orizzontale dal punto di order pool alla cella di magazzino. Di seguito vengono elencati i parametri utilizzati:

- i è il numero della coppia di scaffali (va da 0 a I)
- k è il numero di celle per ogni scaffale (da 0 a K), il corridoio centrale che divide i due blocchi di magazzino si presuppone divida equamente le celle della scaffalatura (cioè divide la cella K/2 dalla cella K/2+1)
- m è la distanza tra due coppie di scaffalature
- n è la larghezza di una coppia di scaffalature (due volte la profondità di una scaffalatura)
- o è la larghezza del corridoio trasversale
- a è la distanza tra la prima coppia di scaffali e il punto di order pool
- h è la larghezza di ogni cella

$$D_v = l \cdot e \quad (2.2)$$

L'equazione 2.2 mostra che la distanza verticale è pari al numero del ripiano in cui si trova la cella (l va da 0 a L) moltiplicato per l'altezza di un singolo ripiano e.

2.1.2 Funzione obiettivo

Il tempo per prelevare un articolo da una cella è pari a (2.3).

$$T_{i,j,k,l} = \frac{D_h}{v_h} + \frac{D_v}{v_z} \quad (2.3)$$

Nella 2.3 vh rappresenta la velocità orizzontale media, mentre vz quella verticale. Entrambe dipendono dal mezzo logistico in utilizzo (o dalla velocità dell'operatore). La formula 2.3 restituisce il tempo di uscita di un articolo dal magazzino e rappresenta il tempo del flusso logistico. Non rappresenta il tempo

di picking perché quest'ultimo riguarda le problematiche di routing e order batching, mentre ora viene analizzato un tempo medio del flusso di uscita degli articoli dal magazzino.
Il modello che si occupa dell'ottimizzazione dello stoccaggio degli articoli a magazzino pone come obiettivo la minimizzazione del tempo medio totale del flusso uscente degli articoli, per cui segue la 2.4.

$$min\, T_{TOT} = min \sum_{p}\sum_{i}\sum_{j}\sum_{k}\sum_{l} w_p x_{p,i,j,k,l} T_{i,j,k,l} \qquad (2.4)$$

La funzione obiettivo 2.4 introduce altri due parametri importanti a definire il modello, che sono il peso w_p e la variabile $x_{p,i,j,k,l}$, entrambi dipendenti dall'articolo p. Il peso w_p permette di influenzare il valore di tempo di prelievo di un articolo ed è definito dalla relazione 2.5.

$$w_p = \frac{C_p}{Q_{p,out}} \qquad (2.5)$$

C_p è il consumo giornaliero medio di un dato articolo, mentre $Q_{p,out}$ è la quantità per UDC che viene utilizzata nelle linee di produzione. Tale peso viene espresso in N° UDC/gg.
$x_{p,i,j,k,l}$ è la variabile binaria che stabilisce la posizione di un articolo a magazzino (2.6).

$$x_{p,i,j,k,l} = \begin{cases} 1 & se\ l'articolo\ p\ risiede\ nella\ cella\ i,j,k,l \\ 0 & altrimenti \end{cases} \qquad (2.6)$$

2.1.3 Vincoli

Una volta identificate tutte le variabili si procede con la generazione dei vincoli necessari a garantire una corretta risoluzione del problema. I vincoli sono tre:

1) Ogni articolo può occupare solo una cella di magazzino, cioè

$$\sum_i \sum_j \sum_k \sum_l x_{p,i,j,k,l} = 1 \quad \forall p \tag{2.7}$$

2) Ogni cella al massimo può ospitare un solo articolo

$$\sum_p x_{p,i,j,k,l} \leq 1 \quad \forall (i,j,k,l) \tag{2.8}$$

3) La variabile $x_{p,i,j,k,l}$ è binaria per cui

$$x_{p,i,j,k,l} \in \{0,1\} \tag{2.9}$$

2.1.4 Trasposizione del problema su Matlab

Per affrontare il problema dal punto di vista computazionale è stato utilizzato il software ingegneristico Matlab. Si è deciso di utilizzare la funzione *"intlinprog"* di Matlab, appartenente all'Addon *Optimization Toolbox*. Tale funzione è un risolutore *mixed-integer linear programming (MILP)* e trova il minimo di una funzione obiettivo in base ai seguenti vincoli (2.10, Gamultiobj algorithm).

$$\min_x f^T x \; con \; vincoli \begin{cases} x(intcon) \; integer \\ A \cdot x \leq b \\ Aeq \cdot x = beq \\ lb \leq x \leq ub \end{cases} \tag{2.10}$$

f, x, $intcon$, b, beq, lb, ub sono vettori. A, Aeq sono matrici. Gli output del problema sono il vettore x e i relativi valori della funzione obiettivo f. Il vettore x è costruito in modo tale che gli elementi da $x(1)$ a $x(num_{celle})$ corrispondano al primo articolo preso in esame e dunque, in base ai vincoli impostati, siano tutti "0" tranne un "1" che mi dà la posizione dell'articolo a magazzino. Questo stesso tratto di vettore viene ripetuto per tutti gli altri articoli. Di fatto, partendo dalla matrice con elementi di carattere binario che presenta sulle colonne gli articoli e sulle righe le celle, il vettore x viene costruito mettendo tutte le colonne una sotto l'altra. In modo analogo è stato costruito il vettore riga della funzione obiettivo

andando a trasformare la matrice dei tempi pesati (il peso è il parametro w_p, vedi 2.5), che presenta sulle righe le celle di magazzino e sulle colonne gli articoli.
Il vincolo 1 (2.7) può essere comunicato tramite la matrice Aeq ed il vettore beq. Ogni riga della matrice Aeq rappresenta le possibili celle occupabili da un articolo p. Ad esempio la seconda riga riguarderà il secondo articolo e presenterà tutti "0" tranne un tratto di riga occupato da "1" posti nelle posizioni dalla colonna $num_{celle} + 1$ alla colonna $2 \cdot num_{celle}$. Questo nei termini del problema lineare si traduce nell'affermare che l'espressione $Aeq(2,:) \cdot x = 1$, cioè porre il prodotto matriciale della seconda riga della matrice Aeq per il vettore x uguale a 1 (il vettore b è composto da 1 ripetuti il numero di articoli), è equivalente a vincolare al valore 1 la somma degli elementi da $num_{celle} + 1$ a $2 \cdot num_{celle}$ del vettore x.
Il vincolo 2 (2.8) viene trattato tramite la matrice A e il vettore b. In questo caso ogni riga riguarda invece una singola cella. La prima riga della matrice A sarà ad esempio composta dal tratto di riga composto da un "1" e da $num_{celle} - 1$ "0", ripetuto per il numero di articoli.
Il vincolo 3 viene costruito fornendo i due limiti superiore ed inferiore: lb è un vettore di zeri, mentre ub è un vettore unitario.
La dimensione di x viene impostata tramite il parametro $intcon$, pari in questo problema al numero di articoli per il numero di celle.

2.2 Ottimizzazione della scelta delle UDC

Nella prima parte dell'algoritmo il vettore Qp è stato supposto noto. La scelta delle UDC condiziona la posizione a magazzino perché influenza il parametro wp impostato come peso del problema precedente e dunque merita un approfondimento per capire quali possono essere le logiche che determinano la scelta ottimale delle UDC per ogni articolo. Tale scelta è un problema multi-obiettivo. Da un lato aumentare la dimensione delle UDC e dunque la quantità di pezzi per UDC permette di poter prelevare meno volte l'articolo dal magazzino, risparmiando tempo di picking. Dall'altro diminuire le dimensioni dell'UDC permette di ridurne lo spazio occupato nelle linee di produzione, evitando di accumulare pezzi di articoli non necessari e permettendo di avere la disposizione, in posizioni più comode, di un maggior quantitativo di articoli.

2.2.1 Funzione obiettivo

Le funzioni obiettivo sono due:

- Ridurre il tempo medio di flusso degli articoli in uscita
- Diminuire lo spazio occupato in linea/stazione di assemblaggio

Il tempo del flusso degli articoli in uscita viene determinato in modo analogo alla 2.4, con la differenza che la matrice delle posizioni occupate x in questo caso sarà ritenuta nota (e denominata $u_{p,i,j,k,l}$), mentre la variabile del problema è rappresentata dalla quantità per UDC Qp (2.11).

$$min\ T_{TOT} = min \sum_{UDC} \sum_{p} \sum_{i} \sum_{j} \sum_{k} \sum_{l} w_{p,UDC} u_{p,i,j,k,l} T_{i,j,k,l} \quad (2.11)$$

Con w_p calcolato mediante la 2.12

$$w_{p,UDC} = \frac{C_p}{Q_{p,UDC} \cdot x_{UDC,p}} \quad (2.12)$$

La natura del problema è però discontinua. Infatti per poter ottimizzare l'efficienza temporale delle UDC è necessario massimizzarne il contenuto. Dunque, come ipotesi del problema, non saranno tollerate soluzioni che non prevedano di colmare le UDC, in modo da evitare contenitori mezzi vuoti. Questo si traduce nella scomposizione della quantità per UDC in altri due parametri che sono $x_{UDC,p}$ e $Q_{p,UDC}$. $x_{UDC,p}$ è la variabile del problema ed è rappresentata da una matrice $[x_{UDC,p}]$ con elementi tutti binari che presenta sulle righe le UDC e sulle colonne gli articoli e che determina dunque per ogni l'articolo la UDC selezionata tramite un "1". $[Q_{p,UDC}]$ è una matrice simile alla $[x_{UDC,p}]$, ma presenta sulle righe gli articoli e sulle colonne le UDC; gli elementi di quest'ultima matrice sono le quantità per UDC di ogni articolo in ogni possibile UDC e dunque è una panoramica di tutte le possibili combinazioni tra gli articoli e le UDC considerate.
La seconda funzione obiettivo considera invece un nuovo parametro definito dalla relazione 2.13.

$$m = \frac{M_l}{N_{articoli}} = \frac{\sum_p \sum_{UDC} L_{p,UDC} \cdot x_{UDC,p}}{N_{articoli}} \quad (2.13)$$

Si suppone per ipotesi che tutte le UDC vengano posizionate in linea di assemblaggio con il lato corto rivolto verso l'operatore, in modo da permettere ad un maggiore quantitativo di contenitori di essere disposti frontalmente all'operatore. M_l indica dunque la sommatoria delle larghezze delle UDC selezionate per ogni articolo e viene definito come Metri Lineari occupati. Dividendo per il numero di articoli si ottiene la larghezza mediamente occupata da ogni articolo. Minimizzare questo valore significa complessivamente ridurre lo spazio occupato sulle stazioni di assemblaggio.

2.2.2 Vincoli

I vincoli di questo problema sono 2:

1) Ogni articolo può essere assegnato ad una sola UDC

$$\sum_{UDC} x_{UDC,p} = 1 \quad \forall p \quad (2.14)$$

2) La variabile $x_{UDC,p}$ è binaria, per cui

$$x_{UDC,p} \in \{0,1\} \quad (2.15)$$

2.2.3 NSGA-II Procedure: Introduzione

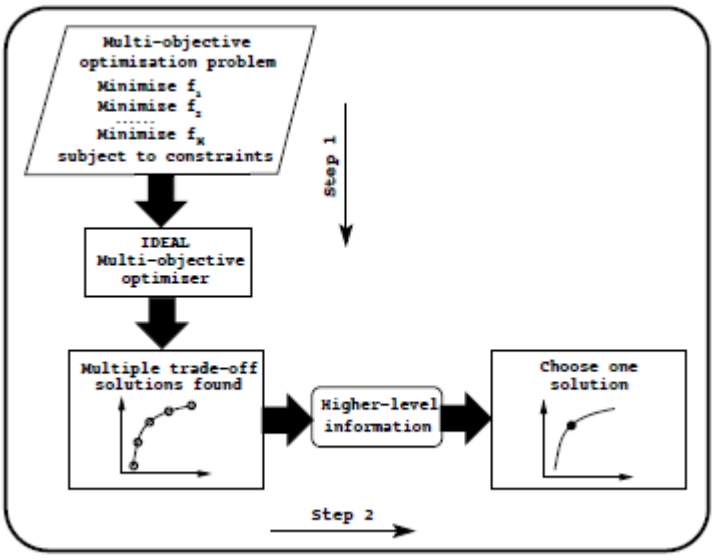
Un problema multi-obiettivo richiede la determinazione di un set di soluzioni in grado di soddisfare contemporaneamente più fitness functions. Quando il problema assume un valore dimensionale elevato tale set di soluzioni si traduce in un insieme che può essere infinito o comunque finito, ma estremamente numeroso. La capacità di risoluzione dell'algoritmo è legata alla potenza del calcolatore usato, ma in ogni caso non è un problema risolvibile andando semplicemente a determinare tutte le soluzioni possibili per effettuare una

selezione alla fine. Il solo calcolo delle fitness functions nei problemi multi-obiettivo non è sufficiente a selezionare gli individui migliori perché gli obiettivi sono multipli ma tale calcolo costituisce un solo criterio. È necessario dunque introdurre un nuovo concetto, che è quello di dominanza, in grado di suddividere la popolazione in gruppi omogenei di individui dominanti e dominati.
L'algoritmo NSGA-II (Non-Dominated Sorting Algorithm II) è uno dei più popolari ed è appartenente al gruppo degli algoritmi EMO (Deb K., 2011). Cerca soluzioni ottimali della frontiera di Pareto tramite le seguenti features:

- Principio di elitarismo
- Meccanismo di salvaguardia della diversità
- Privilegia soluzioni non-dominate

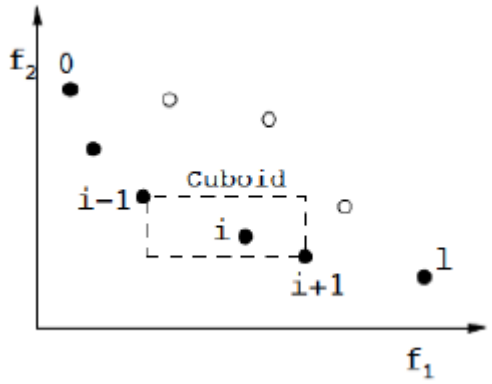
EMO sta per *Evolutionary Multi-Objective Optimization*. L'obiettivo è quello di minimizzare o massimizzare un set di fitness functions soggette a vincoli. La soluzione ottimale può essere raggiunta solo tramite il concetto di dominanza (vd 2.2.4). L'obiettivo di EMO è quello di cercare un set di punti appartenenti alla frontiera di Pareto, mantenendo in contemporanea la diversità della popolazione affinché la frontiera comprenda punti in tutto il suo range. Il funzionamento schematico di un algoritmo genetico può essere espresso dalla figura 2.2.

Figura 2.2: schema rappresentativo di EMO (Deb K. 2011)



L'algoritmo genetico fornisce in output la frontiera di *trade-off* (frontiera di Pareto) con un set di punti ottimali in grado di soddisfare i requisiti del problema. Risiede poi nell'utilizzatore dell'algoritmo la facoltà di scegliere il punto migliore tra gli ottimi consigliati tramite l'utilizzo di una funzione di scelta (che ovviamente deve poter pesare relativamente i vari obiettivi).
Ad ogni generazione NSGA-II genera la popolazione modificata geneticamente e la unisce agli individui selezionati della generazione precedente. La popolazione ottenuta subisce una suddivisione in ranks (vd 2.2.4) con la creazione di frontiere di punti non-dominati. I punti che eccedono la quantità massima di individui permessi vengono eliminati sulla base del valore della loro *crowding distance* (vd 2.2.4). La *crowing distance* è una misura dello spazio attorno ad un individuo "i" che non è occupato da nessun altro individuo (fig. 2.3).

Figura 2.3: Schema rappresentativo del calcolo della crowding distace (Deb K. 2011)



La modifica genetica della popolazione avviene tramite crossover e mutazione. Questi due meccanismi si basano sullo stesso concetto che riguarda i cromosomi e la ricombinazione del patrimonio genetico. Il crossover permette di ottenere soluzioni a partire da coppie di individui genitori selezionati tra i migliori della generazione precedente e di raggiungere la convergenza dell'algoritmo (i figli in qualche modo sono simili ai genitori per cui è molto bassa la probabilità che il figlio sia distante da valori ottimi di fitness functions). La mutazione introduce delle modifiche nel patrimonio genetico di un individuo per generare nuovi cromosomi semplicemente cambiando uno o più componenti del vettore che compone l'individuo. L'idea è che con tali modifiche casuali si possa il più

possibile esplorare parti dello spazio delle soluzioni non ancora osservate, con l'obiettivo di evitare una convergenza locale agli ottimi.

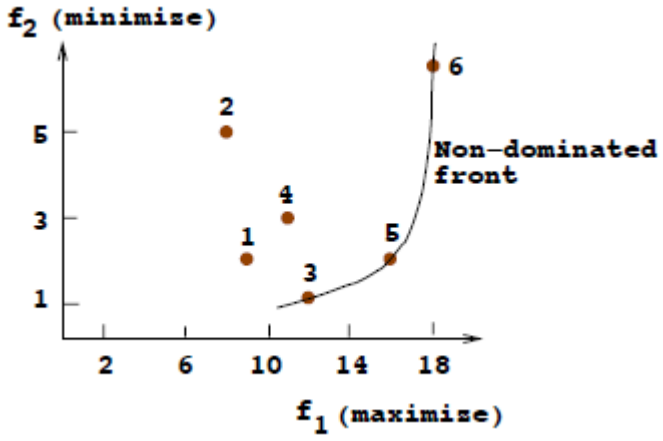
2.2.4 Gamultiobj su Matlab

La variabile di output del problema è una matrice di elementi binari che presenta sulle righe le UDC e sulle colonne gli articoli. Per ragioni analoghe al problema precedente anche in questo caso l'output su Matlab viene trasformato in un unico macro vettore colonna di dimensioni (N,1) con N pari al prodotto tra il numero di articoli e il numero di possibili UDC prese in considerazione. In questo caso, per risolvere il problema, è stata utilizzata la funzione *"gamultiobj"*, appartenente all'Addon *Global Optimization Toolbox* (Gamultiobj algorithm). Tale funzione trova il fronte di Pareto di funzioni obiettivo multiple utilizzando un algoritmo genetico. L'algoritmo genetico usato è una variante di NSGA-II, algoritmo controllato ed elitario (vd 2.2.3). Un algoritmo elitario privilegia individui con valori migliori di fitness functions; il fatto che sia controllato permette di esplorare più aree di scenari possibili aumentando la diversità di popolazione (anche a partire da individui con valori di fitness function peggiori), non permettendo dunque una troppo rapida convergenza: il rischio è di non trovare un fronte di Pareto ottimo globale, ma invece solamente uno locale, ovvero potrebbe capitare che punti non vicini a quelli selezionati dal fronte in realtà abbiano valori migliori di fitness function. L'elitarismo e la diversità vengono controllati tramite due opzioni di gamultiobj: ParetoFraction e DistanceMeasureFcn. La prima controlla il numero massimo di individui che possono esistere sul fronte di Pareto (individui elitari). La seconda aiuta invece a mantenere la diversità della popolazione mantenendo nell'algoritmo individui caratterizzati da valori di funzioni obiettivo relativamente distante dal fronte di Pareto individuato ad una data generazione.
Il concetto di elitarismo si traduce mediante la dominanza (2.16).

$$\begin{cases} f_i(x) \leq f_i(y) & \forall i \\ f_j(x) < f_j(y) & per\ alcuni\ j \end{cases} \qquad (2.16)$$

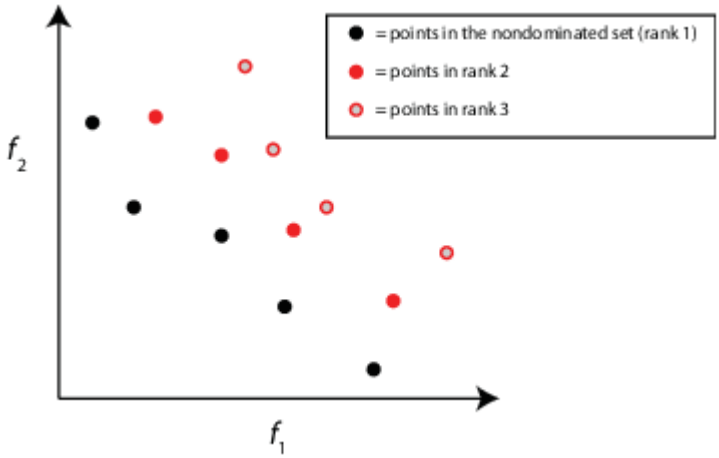
Un punto x domina su un punto y se la 2.16 viene rispettata. L'insieme di punti dominanti compone i Q punti della frontiera di Pareto (vedi 2.17).

Figura 2.4: esempio di dominanza (Deb K. 2011)



Nell'esempio sopra riportato (fig. 2.4) viene evidenziata la frontiera di Pareto che raggruppa i punti non-dominati. Il punto "5" ha la stessa f_2 del punto "1" ma una f_1 maggiore (nell'esempio la f_1 va massimizzata), per cui il punto "5" domina il punto "1". La caratteristica di tale frontiera è che il vantaggio ottenuto in un obiettivo nel passaggio da un punto ad un altro appartenente a questo raggruppamento avviene solo tramite il sacrificio di uno o più degli altri obiettivi. Si è dunque in una condizione di *trade-off* (conflitto tra gli obiettivi).
L'algoritmo ragiona tramite il concetto di *rank*: gli individui appartenenti al rank "k" sono dominati solo dagli individui appartenenti al rank "k-1" o minore, come ad esempio illustrato dalla figura 2.5.

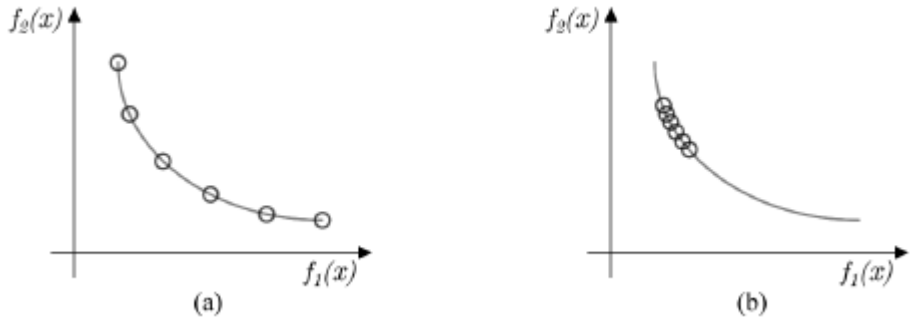
Figura 2.5*: Immagine esempio sul confronto tra i rank della popolazione di individui nell'algoritmo genetico Gamultiobj (gamultiobj algorithm)*



Gli individui del rank "1" sono quelli caratterizzati dai migliori valori delle funzioni obiettivo, per cui dominano su tutti gli altri. Gli individui del rank "2" possono dominare solo quelli del rank "3". Gamultiobj utilizza tale concetto per scegliere gli individui che dovranno essere i genitori della generazione successiva.

Un altro concetto chiave che riguarda il Gamultiobj è quello di *"Crowding Distance"*: è una misura della vicinanza di un individuo ai suoi vicini più stretti, calcolata a partire dagli individui di uno stesso *rank*, rappresenta dunque la densità di soluzioni circostanti un particolare individuo. Tale computazione richiede, dopo aver raggruppato gli individui in ranks, di ordinarli in ordine ascendente di grandezza; per ogni funzione obiettivo (e dunque per ogni dimensione del problema) agli individui con i valori agli estremi (maggiore e minore) viene assegnato il valore di distanza infinita. Agli altri individui viene assegnata una distanza pari alla differenza assoluta normalizzata tra i valori di fitness function delle due soluzioni adiacenti, ovvero è una misura dello spazio attorno ad un individuo non occupato da nessun altro individuo della popolazione corrente. Può essere anche vista come una misura della densità della popolazione: le soluzioni che si trovano in zone ad alta densità (quindi con bassa *Crowding distance*) riceveranno una minore chance di selezione per la successiva generazione: l'obiettivo è quello di evitare affollamenti ed avere una distribuzione più uniforme di punti sul fronte ottimo di Pareto (fig. 2.6).

Figura 2.6*: Esempio di distribuzione delle soluzioni sul fronte ottimo di Pareto*



La *Crowding distance* complessiva di un individuo viene ottenuta sommando le *crowding distances* ottenute nelle varie dimensioni del problema. La *crowding distance* è un parametro fondamentale perché determina il calcolo dello spread (vd 2.17).

L'algoritmo si interrompe (raggiunge convergenza) se lo spread risulta inferiore ad un dato valore impostabile. Lo spread è una misura dello spostamento generazionale della curva di Pareto e viene definito dalla seguente relazione (2.17).

$$spread = \frac{\mu + \sigma}{\mu + Qd} \qquad (2.17)$$

σ è la deviazione standard delle distanze tra i punti che compongono il fronte di Pareto; Q è il numero di punti; d è la distanza media misurata tra questi punti. μ è la somma degli indici di funzione obiettivo "k" della norma della differenza tra l'attuale punto di Pareto a valore minimo per quell'indice e il punto minimo per quello stesso indice nella precedente iterazione. Dunque l'algoritmo si ferma quando sia μ è piccolo, cioè i valori estremi della funzione obiettivo non cambiano tra le iterazioni, sia σ è piccolo, ovvero i punti sul fronte di Pareto sono distribuiti in modo uniforme. Tale condizione di uscita viene solitamente fissata tramite il parametro *options.FunctionTolerance*: quando la media geometrica della variazione relativa dello spread su *n* generazioni (*n* fissato tramite un altro parametro chiamato *options.MaxStallGenerations*) è minore della tolleranza fissata; lo spread finale deve essere minore dello spread medio delle generazioni precedenti. In modo alternativo l'uscita può essere causata da vincoli computazionali, elencati di seguito:

- Numero massimo di generazioni.
- Limite di tempo computazionale.
- Fitness limit: l'algoritmo si ferma se il valore di funzione obiettivo è inferiore ad un valore impostato.

2.2.5 Inizializzazione di Gamultiobj

Il passo iniziale per avviare l'algoritmo è quello di inizializzare la popolazione. La popolazione iniziale può essere fornita tramite alcuni metodi:

- Con l'opzione *InitialPopulationMatrix*. Il numero di individui della popolazione deve essere pari all'opzione *PopulationSize*.
- Con le impostazioni default di Matlab che utilizzano gli algoritmi gacreationuniform o gacreationlinearfeasible. Il primo genera la

popolazione iniziale senza vincoli (considera eventualmente solo gli estremi); la seconda funziona quando ci sono vincoli lineari.
- Impostando manualmente una funzione di creazione della popolazione iniziale, tramite l'opzione optimoptions *"CreationFcn"*.

Si è scelto di impostare manualmente la popolazione perché in questa maniera è possibile non solo rispettare i vincoli lineari e i limiti di validità della variabile, ma anche il vincolo di valori *integer* (ricordando che il problema è di natura discreta). La funzione, denominata *int_pop*, genera individui caratterizzati dalla presenza di elementi binari (0 o 1) fino al raggiungimento del limite di abitanti della popolazione iniziale (PopulationSize) e li filtra attraverso un check in grado di determinare se gli elementi rispettino i vincoli lineari: solo se ritenuti validi vengono conservati altrimenti vengono eliminati. Si procede con iterazioni per ripristinare gli individui mancanti al raggiungimento del limite. L'iterazione è stata costruita con un ciclo *while* (vedere appendici).
Complessivamente in input *gamultiobj* richiede i seguenti argomenti:

- Funzioni obiettivo
- Dimensione della variabile x soluzione del problema
- Vincoli lineari di disuguaglianza (matrice A, vettore b)
- Vincoli lineari di uguaglianza (matrice Aeq, vettore beq)
- Range di accettabilità della soluzione (limiti lb e ub)
- Vincoli non lineari
- Opzioni di ottimizzazione: è possibile modificare i parametri di default dell'algoritmo tramite il comando *optimoptions*.

2.2.6 Iterazioni dell'algoritmo Gamultiobj

Le generazioni successive vengono create tramite i seguenti passaggi:

- La funzione selezione sceglie gli individui dalla popolazione corrente che dovranno essere selezionati per far parte della generazione successiva. La funzione default è il *binary tournament selection*, che preleva i candidati a due a due (è binario) per determinare ogni volta il migliore (in base alle fitness functions).

- L'algoritmo utilizza gli individui della popolazione corrente per generare individui "figli" tramite le funzioni di mutazione e crossover.
- Per ogni figlio vengono calcolati i valori di funzione obiettivo. I figli vengono uniti alla popolazione corrente.
- Tutti gli individui della popolazione corrente vengono suddivisi in ranks e vengono calcolate le crowding distances.
- La popolazione viene riportata alla dimensione massima di individui (*PopulationSize*) mantenendo solo gli individui appropriati per ogni rank.

Come già menzionato, la condizione di uscita principale viene data tramite lo spread massimo accettabile.
L'operazione di selezione naturale è legata al concetto di evoluzione biologica: gli individui che dimostrano migliori valori di fitness functions sono i candidati ad essere selezionati per poter essere inseriti all'interno della generazione successiva dell'algoritmo. Affinché si possa giungere a convergenza globale è necessario che il patrimonio genetico sia sufficientemente vario ed eterogeneo.
Anche per le funzioni di mutazione e crossover si è preferito utilizzare funzioni custom, in modo da garantire la natura binaria del problema.
Il crossover avviene tramite la ricombinazione del patrimonio genetico di altri cromosomi in modo da mantenerne un aspetto simile: tale aspetto è fondamentale per assicurare la convergenza dell'algoritmo. Se i figli non fossero simili ai genitori non sarebbe garantita la convergenza. Il crossover viene descritto come lo scambio di tratti di patrimonio genetico tra una coppia di individui genitori. Esistono più tipologie di crossover binario. Il crossover binario viene chiamato ad un punto ed il figlio di lunghezza di genoma "l" viene ottenuto dagli "n" elementi del primo genitore e dagli "l-n" elementi del secondo genitore. Nel crossover a due punti l'effetto è analogo solo che si prendono "k1" e "l-k2" elementi dal primo genitore e gli elementi da "k1" a "k2" del secondo genitore.
La funzione di crossover scelta, denominata *int_crossover*, permette di generare dei figli a partire da coppie di individui genitori selezionati. La generazione di figli avviene tramite l'unione di pezzi di vettore degli individui genitori ed è realizzato andando ad impostare due valori casuali integer k_1 e k_2 compresi tra 1 e il numero dimensionale del problema (numero articoli per numero UDC), con passo pari al num_UDC. In tale modo ogni pezzo selezionato dal genitore sarà non potrà presentare per uno stesso articolo pezzi di vettore da genitori

diversi, dunque l'individuo figlio sarà conforme con i vincoli lineari impostati all'inizio del problema.
L'operatore di mutazione introduce dei cambiamenti nel patrimonio genetico per generare nuovi cromosomi. Non combina il patrimonio genetico di due genitori come fa il crossover, ma al contrario reintroduce aspetti genetici che erano andati persi a seguito dell'operazione di selezione. La mutazione cambia il valore di uno o più dei componenti della stringa di bit. L'obiettivo della mutazione è quello di modificare pesantemente il patrimonio genetico di alcuni individui, affinché si possa esplorare lo spazio delle soluzioni alla ricerca di aree non valutate. Tale strumento permette dunque di evitare la convergenza rapida ad un set di soluzioni ottime locali. Bisogna però evitare un abuso di questo strumento, altrimenti il rischio è di raggiungere la situazione opposta, ovvero rendere instabile l'algoritmo e non giungere mai a convergenza.
La funzione di mutazione, denominata *int_mutation*, genera degli individui figli a partire dalla popolazione selezionata corrente andando a modificare in maniera casuale la posizione degli articoli all'interno delle UDC di un individuo padre.
La nuova popolazione (generazione successiva) sarà dunque formata tra tre tipologie di figli:

- Gli individui selezionati tramite la *binary tournament selection*.
- Gli individui figli generati con la funzione di crossover.
- Gli individui figli che hanno subito la funzione di mutazione.

2.2.7 Output dell'algoritmo Gamultiobj

Gli output richiedibili all'algoritmo *gamultiobj* sono i seguenti:

- La soluzione x del problema (sono i punti che risiedono sulla frontiera di Pareto), restituita come una matrice m x n, in cui m è il numero di soluzioni trovate e n è la dimensione di ogni soluzione.
- I valori di fitness functions relativi all'insieme di soluzioni ottime trovate.
- L'*exitflag*, che spiega la condizione di uscita utilizzata dall'algoritmo.
- Il set di informazioni sul processo di ottimizzazione: tipo di problema affrontato (informazioni su vincoli applicati), numero totale di generazioni create, numero totale di funzioni obiettivo calcolate, distanza

media calcolata tra le soluzioni (deviazione standard della norma della differenza tra le soluzioni sul fronte di Pareto e la loro media), lo spread finale (relativo alle ultime due iterazioni), numero massimo di violazioni ai vincoli impostati del set di Pareto.

- La popolazione finale (completa) all'ultima iterazione.
- Risultati della popolazione finale (funzioni obiettivo).

2.2.8 Selezione della soluzione migliore

Come anticipato nell'introduzione all'algoritmo NSGA-II (vd 2.2.3), la parte finale dell'applicazione dell'algoritmo richiede l'intervento da parte dell'utilizzatore per la selezione della soluzione migliore. L'algoritmo in output fornisce un set di soluzioni ottime che stanno sulla frontiera di Pareto. La soluzione migliore tra le ottime va scelta in base ad una funzione di selezione che possa valutare la bontà quantitativa di una soluzione in base ai suoi valori di fitness function pesati. Si è scelto di utilizzare una semplice distanza euclidea, come mostrato dalla 2.18.

$$F = \sqrt{\left(\frac{f_1}{N}\right)^2 + \left(f_2 - f_{2,MIN}\right)^2} \qquad (2.18)$$

Dunque le due funzioni obiettivo sono state pesate in modo diverso ed è stata effettuata una predilezione a favore dello spazio in linea. La soluzione migliore del modello fornito verrà scelta in base al valore di F. Nella relazione 2.18 f_1 indica la funzione obiettivo 1, ovvero la minimizzazione del tempo, mentre la f_2 indica la seconda funzione obiettivo, cioè minimizzare lo spazio occupato in linea. N è il numero di articoli e permette dunque di valutare un tempo medio di flusso in uscita dal magazzino per articolo. $f_{2,MIN}$ indica invece la larghezza minima media di UDC e dipende dall'unità di carico più piccola che si sceglie di utilizzare (non posso avere una larghezza media inferiore a questa). La soluzione col valore più basso di F verrà ritenuta quella ottimale alla risoluzione del problema di ottimizzazione delle UDC. Tale valutazione si basa sulla considerazione che il punto virtualmente perfetto per la soluzione del problema è caratterizzato da entrambi i valori delle fitness functions uguali al minimo teoricamente possibile, ovvero tempo zero di flusso in uscita degli articoli dal

magazzino e spazio occupato mediamente in linea dagli articoli pari alla dimensione di lato corto dell'UDC più piccola possibile. Tale soluzione virtuale ottima viene definita punto "Utopia".

Capitolo 3

Caso di studio

In questo capitolo verranno mostrate le applicazioni dei due modelli sviluppati nel capitolo 2 ad un caso di studio reale: il primo sull'ottimizzazione delle posizioni di stoccaggio, il secondo sull'ottimizzazione delle unità di carico (UDC). Ci sarà una presentazione dell'azienda presso cui è stata effettuata l'applicazione, a cui seguirà l'analisi dei risultati riguardanti i due modelli. Mentre il primo modello è stato ricavato dalla letteratura, per cui ha già ottenuto una validazione in test precedenti (Ene e Ozturk, 2012), il secondo modello presenta aspetti innovativi che richiedono maggiori considerazioni.

3.1 Presentazione dell'azienda

La società presso cui è stato effettuato lo stage correlato con lo sviluppo della tesi è la Ceado s.r.l., realtà aziendale situata a Spinea (VE). Il nome dell'azienda risale al soprannome veneziano della famiglia Girardi. Il presidente, Michele Girardi, figlio del fondatore Egidio Girardi, è l'ingegnere inventore dei prodotti firmari Ceado, mentre il fratello Paolo Girardi è a capo del settore commerciale dell'azienda. La famiglia ha saputo avventurarsi nel campo del "Made in Italy" con prodotti che vengono venduti in tutto il mondo. Il motto della Ceado è "Think strong, work better", il quale ben enuncia la devozione verso il continuo miglioramento tramite l'analisi di tutte le minime caratteristiche di ogni prodotto al fine di presentare al cliente attrezzatura altamente affidabile e professionale. Nonostante sia ancora una piccola realtà, Ceado segue una filosofia commerciale ed ingegneristica volta a soddisfare le più attente richieste di barmen e baristi, fondendo la cura artigianale, con cui vengono costruiti i macchinari, con

l'obiettivo industriale di espandere il marchio e diventare un punto di riferimento nel panorama internazionale del Bar Equipment e dei Macinacaffè. Tale aspirazione viene conseguita anche con l'aggiornamento costante del comparto tecnico e logistico. È in questa ottica che deve essere vista l'adesione di Ceado all'industria 4.0. L'industria 4.0 non è solo un progetto atto a migliorare le attrezzature e i macchinari per la ricerca di nuove tecniche produttive o di stoccaggio e gestione dei materiali, ma è anche un'esigenza di cambiamento e aggiornamento della mentalità dell'industria italiana (Beltrametti et al., 2017). Il tessuto economico nazionale è formato principalmente (99%) da piccole imprese, che determinano il 50% del PIL. Nell'ottica di un'azienda connessa con i mercati internazionali, assume grande valore il piano di evoluzione digitale (*Internet of Things*) e di evoluzione della cultura aziendale e manageriale: non è pensabile che un'azienda, che ha sempre lavorato con un approccio "tradizionale", si possa improvvisare "fabbrica connessa". Il corretto modo di pensare al progetto nazionale di industria 4.0 è lo stesso sposato da Ceado: un processo evolutivo a step e non una rivoluzione, una mentalità fortemente condivisa dalla filosofia *Lean* tramite l'ormai noto *Kaizen*.

3.1.1 Lo stabilimento produttivo

La Ceado ha già da tempo iniziato un percorso volto al miglioramento dei processi produttivi tramite l'implementazione della filosofia *Lean Production* e dei *kanban* per la gestione logistica e produttiva interna. Lo stabilimento presenta un'area adibita alla parte produttiva con linee di assemblaggio e stazioni di montaggio ed un'area designata a magazzino. Le linee di assemblaggio sono strutturate a step, con ogni step movibile ed intercambiabile per fornire diverse soluzioni di configurazione della linea, mantenendo al contempo lo spazio destinato alle linee più compatto. Il flusso degli articoli è lineare in modo che gli i prodotti finiti rappresentino la parte a valle dello stabilimento, mentre a monte c'è il magazzino di materie prime. La movimentazione dal magazzino alle linee di assemblaggio avviene tramite un *tugger train*, soluzione logistica molto improntata alla generazione di un flusso di rifornimento gestibile e regolabile in tempi e ritmi. I rimorchi permettono il trasporto sia di articoli su trolley (ad esempio per componenti voluminosi come i corpi dei vari maccinari), sia di articoli disposti in UDC su scaffalature mobili a ripiani (tranquillamente montabile sul *tugger train*).

3.1.2 Classificazione aziendale

La Ceado è una società classificabile principalmente come ATO, ovvero *Assembly To Order*. È una tipologia molto diffusa che pone il proprio focus sull'assemblaggio dei prodotti finiti. Alla ricezione dell'ordine cliente il flusso interno aziendale prevede soprattutto la preparazione di montaggi di semilavorati e l'assemblaggio dei prodotti finiti. Questo significa che la pianificazione deve prevedere e coprirsi con le scorte di materie prime e semilavorati da officina. Il tempo di risposta al cliente è abbastanza rapido dal momento che il Lead Time dipende solo da assemblaggio e preparazione degli articoli per la spedizione. Solo per alcune configurazioni standard di prodotti finiti c'è una scorta a magazzino con conseguente piccola componente di MTS (*Make to Stock*), adibita al soddisfacimento di ordini cliente con basso Lead Time richiesto.

3.2 Generazione degli input del problema

Per la generazione di tutti gli input necessari alla corretta esecuzione degli algoritmi visti nel capitolo precedente è stato necessario effettuare una fase di raccolta dati. Solo i consumi degli articoli e i parametri di definizione del magazzino erano noti e facilmente reperibili.

3.2.1 Consumo medio giornaliero

I consumi degli articoli sono stati prelevati dal software gestionale in uso all'interno dell'azienda e sono stati ricavati effettuando una media giornaliera degli scarichi degli articoli. Tale informazione è stata comunicata all'algoritmo tramite la matrice nx2 in cui ad ogni codice in colonna "1" è associato il consumo giornaliero in colonna "2" (fig. 3.1).

Figura 3.1: *Estratto della tabella utilizzata con i consumi giornalieri medi per ogni articolo*

	A	B
1	Codice articolo	Consumi giornalieri
2	10186	10
3	10191	100
4	10199	15
5	10304	40
6	10310	14
7	10318	75
8	10341	42
9	10360	15
10	10375	20
11	10381	12
12	10694	20
13	10695	20

3.2.2 Definizione del magazzino manuale

I parametri geometrici del magazzino sono stati inseriti all'interno di un vettore chiamato "Param_Mag". Tale vettore comprende i seguenti dati:

- I è il numero di coppie di scaffali
- L è il numero di ripiani per scaffale
- K è il numero di celle per ogni ripiano di ogni scaffale
- m è la distanza tra due coppie di scaffalature
- n è la larghezza di una coppia di scaffalature (di fatto due volte la profondità di una scaffalatura
- o è la larghezza del corridoio trasversale
- a è la distanza tra il blocco magazzino e il punto di output (accumulo articoli uscenti per order batching)
- h è la larghezza di ogni cella
- e è l'altezza di ogni cella
- vh è la velocità orizzontale
- vz è la velocità verticale

Di seguito viene riportata la tabella riassuntiva con i dati di input selezionati per la risoluzione del problema (fig 3.1).

Figura 3.2: *Tabella contenente i parametri geometrici di definizione del magazzino*

I	K	L	m	n	o	a	h	e	vh	vz
-	-	-	m	m	m	m	m	m	m/s	m/s
10	18	3	2,7	1,7	2,7	3	0,9	1,4	0,5	0,3

3.2.3 Creazione e popolazione della matrice Qp

Per la creazione del database si è scelto di appoggiarsi al DBMS (*Database Management System*) Microsoft SQL Server 2017. SQL (*Structured Query Language*) è un linguaggio di programmazione per database per la gestione di informazioni contenute in un database "relazionale". Il termine relazionale indica la natura del DMBS, ovvero basata sulla presenza di relazioni tra database in forma di tabelle, note come "relations". Con il linguaggio SQL è possibile ogni tipo di modifica:

- Creazione ed eliminazione di tabelle
- Selezionare e modificare righe
- Accedere ai dati contenuti nel database relazionale
- Definire i dati di un database
- Creare viste
- Creare stored procedures per la gestione automatica dei dati
- Impostare i livelli di autorizzazione e i permessi di accesso su tabelle, procedure e viste

Il database è stato popolato andando a registrare tutte le possibili combinazioni di ogni articolo e UDC, come richiesto dall'input del problema di ottimizzazione delle UDC (vd 2.2). Di seguito viene riportato un estratto della tabella CAPACITA_UDC, costituita dai campi CODICE_ARTICOLO, CODICE_UDC E QUANTITA_UDC. I primi due campi sono chiavi primarie, per cui non possono esistere più soluzioni con stessa combinazione di codice articolo e UDC. Le Primary Keys permettono infatti di creare un indice cluster univoco che determina l'univocità delle combinazioni di valori di tutte le colonne appartenenti alla chiave primaria.
Tale tabella ha permesso di creare una struttura digitale solida, in grado di essere facilmente aggiornata all'occorrenza. La modifica può avvenire tramite una diretta interazione con la tabella all'interno della sezione Esplora Oggetti di SQL

Server Management Studio: è possibile con semplici comandi entrare nella sezione di modifica della tabella e cambiare manualmente i dati voluti. In alternativa può essere utilizzato un set di comandi Transact all'interno della finestra di scrittura degli script.
Come ipotesi lavorativa (vd 2.2.1) le UDC non possono essere riempite a metà, ma è necessario, per aumentare l'efficienza di trasferimento dal magazzino alle linee di produzione, andare a colmare le UDC di pezzi.

3.3 Applicazione del modello al caso di studio

Ora che gli input sono noti è possibile procedere con l'applicazione dei due algoritmi al caso di studio. Il primo, essendo tratto dalla letteratura, non ha bisogno di conferme, mentre il secondo necessita di un'analisi più approfondita. Dunque i risultati del primo modello verranno accettati come input del secondo algoritmo. L'output principale del primo modello fa riferimento alla disposizione ottimale degli articoli a magazzino e consiste in una matrice di dimensioni "mxn" in cui m rappresenta il numero delle singole locazioni a magazzino, mentre n rappresenta il numero di articoli. In seguito si procederà con un piccolo test per verificare la bontà del modello di ottimizzazione della scelta delle UDC. Il test verrà effettuato su scala minore con valori fittizi e i dati in uscita verranno poi interpretati. Con scala minore si intende un quantitativo ristretto di articoli ed una scelta molto limitata di UDC. Una volta appurata la bontà dell'algoritmo, si continuerà con un problema di natura reale, in questo caso si è scelto di ottimizzare la scelta delle UDC di tutti gli articoli presenti sulle linee di assemblaggio.

3.3.1 Risoluzione del problema di ottimizzazione dello stoccaggio

I dati relativi ai consumi sono già stati presentati. Come anticipato in 2.1.2 la quantità per UDC Qp è un input del problema ed è supposta nota. Per comodità viene posta uguale ai consumi giornalieri medi in modo che il numero di UDC mediamente prelevate dal magazzino in un giorno sia pari ad "1". Ovviamente si tratta di una scelta di partenza dell'algoritmo, in modo da poterlo inizializzare. Il risultato viene fornito mediante due functions:

- Tempi_Mag, la quale utilizza i dati in input per calcolare le distanze dal punto di output del magazzino di ogni cella. In seguito calcola i tempi necessari all'uscita di un qualsiasi articolo da ogni cella di magazzino. Tali tempi vengono disposti in un vettore di output "T" che presenta il tempo necessario per estrarre un articolo ed è formato da "2xIxKxL" elementi. La presenza di quel "2" è dovuta al fatto che "I" indica il numero di coppie di scaffali.
- Posiz_Mag, che, a partire dal vettore risultante T della function Tempi_Mag, costruisce la funzione obiettivo tramite la generazione della matrice complessiva dei MT, costituita da "m" volte il vettore colonna T moltiplicato per "wp", in cui "m" indica il numero di articoli. La funzione poi tramite l'algoritmo *intlinprog* va a determinare la disposizione migliore degli articoli a magazzino.

Di seguito viene riportato il resoconto dei risultati dell'algoritmo.

Figura 3.3: *Tabella contenente i parametri computazionali di output*

```
>> Lancio_Alg
LP:               Optimal objective value is 15836.266667.


Optimal solution found.

Intlinprog stopped at the root node because the objective value is within a gap tolerance of the optimal value,
options.AbsoluteGapTolerance = 0 (the default value). The intcon variables are integer within tolerance,
options.IntegerTolerance = 1e-05 (the default value).
```

In figura 3.3 si nota che il risultato di tempo totale di flusso medio giornaliero degli articoli in uscita dal magazzino è pari a 15836,27 s/gg, ovvero circa 4,4 h/gg. Sembra molto elevato ma va ricordato che non è rappresentativo del tempo di picking. Quest'ultimo può essere calcolato con altri algoritmi andando ad affrontare i problemi di *routing* ed *order batching*. L'uscita dall'algoritmo è stata soddisfacente in quanto è stata raggiunta la convergenza (il *Gap Tolerance* è stato rispettato), inoltre tutti i vincoli sono stati rispettati (è stata rispettata la natura integer binaria del problema).

Figura 3.4: *Tabella contenente il resoconto dei tempi computazionali richiesti*

Profile Summary

Generated 02-Dec-2019 15:14:26 using performance time.

Function Name	Calls	**Total Time**	Self Time*	Total Time Plot (dark band = self time)
Lancio_Alg	1	32.279 s	0.003 s	
Posiz_Mag	1	32.276 s	3.269 s	
intlinprog	1	28.560 s	0.024 s	
...hAndCut>IntlinprogBranchAndCut.run	1	28.346 s	0.013 s	
slbiClient	1	28.156 s	0.072 s	
optim\private\slbiMex (MEX-file)	1	16.687 s	16.687 s	
slbiClient>changeConstraintForm	1	10.146 s	10.102 s	
slbiClient>getOptions	1	1.052 s	0.112 s	
opt...vate\validateIntlinprogUserOptions	1	0.549 s	0.158 s	
blkdiag	1	0.418 s	0.418 s	
...SolverOptions.extractOptionsStructure	1	0.296 s	0.012 s	
slbiClient>getFixedOptions	1	0.295 s	0.003 s	
slbiClient>prepareTempFileName	1	0.292 s	0.006 s	
...linprog.extractCustomOptionsStructure	1	0.284 s	0.166 s	
tempname	1	0.267 s	0.043 s	
slbiClient>prepareOutput	1	0.189 s	0.038 s	
...edLinear.performExitflagAndMsgActions	1	0.177 s	0.009 s	
opaque.char	2	0.166 s	0.132 s	
...dLinear>GeneralizedLinear.checkRun	1	0.122 s	0.004 s	
...gorithm.extractCustomOptionsStructure	1	0.118 s	0.117 s	
...t;IntlinprogBranchAndCut.checkProblem	1	0.118 s	0.007 s	
...on>SlbiCommon.createExitflagAndMsg	1	0.115 s	0.005 s	
...createExitflagAndMsgNormalTermination	1	0.110 s	0.030 s	
slbiClient>formOutput	1	0.108 s	0.103 s	

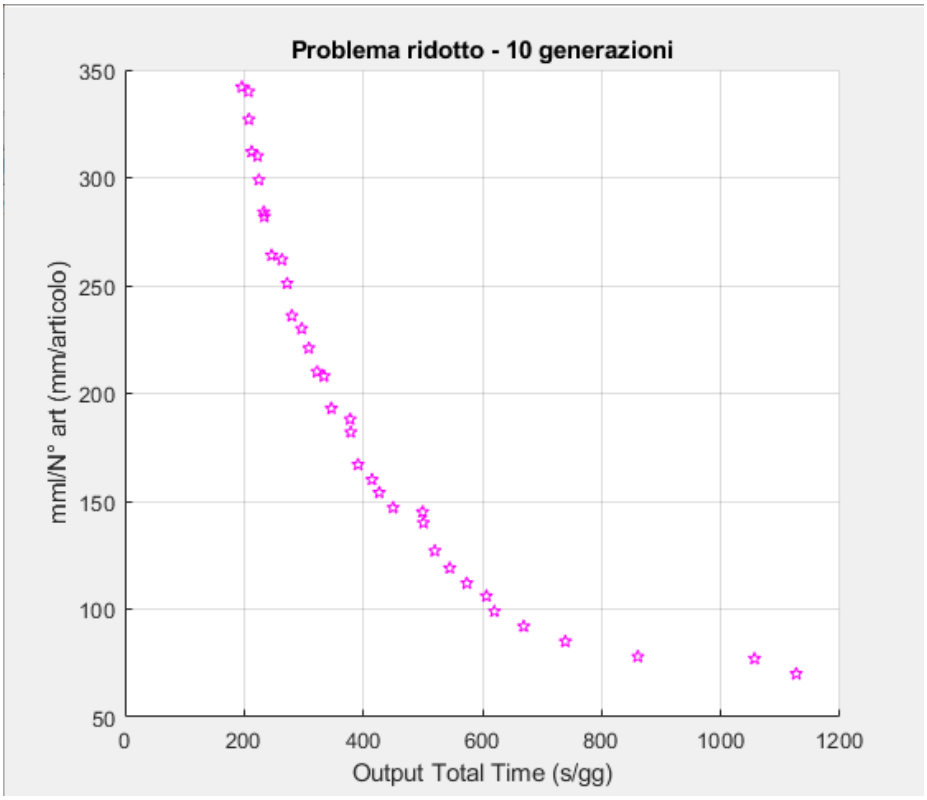
Dalla figura 3.4 si evince che il tempo totale richiesto dall'algoritmo è di poco più di 32s, per la risoluzione di un problema con 438 articoli e 1080 celle di magazzino.

3.3.2 Risoluzione del problema "ridotto"

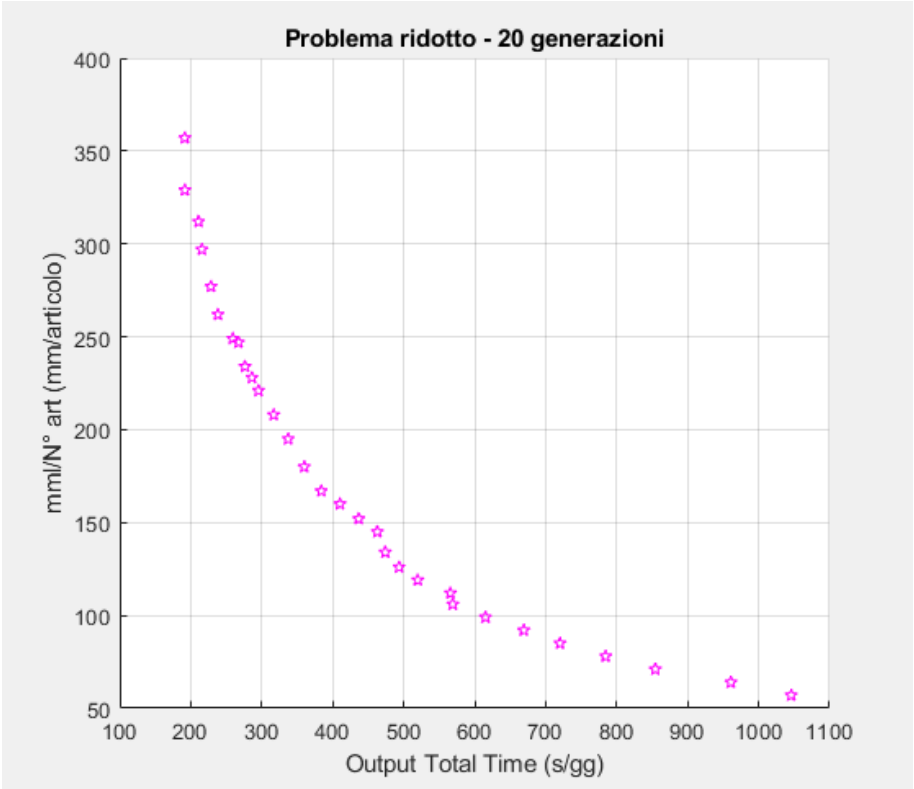
Il problema in natura ridotta è stato affrontato con solamente 10 codici e 4 possibili UDC. Di seguito vengono riportati gli output del problema (fig. 3.5).

Figura 3.5: *Output del problema ridotto al variare del numero di generazioni*

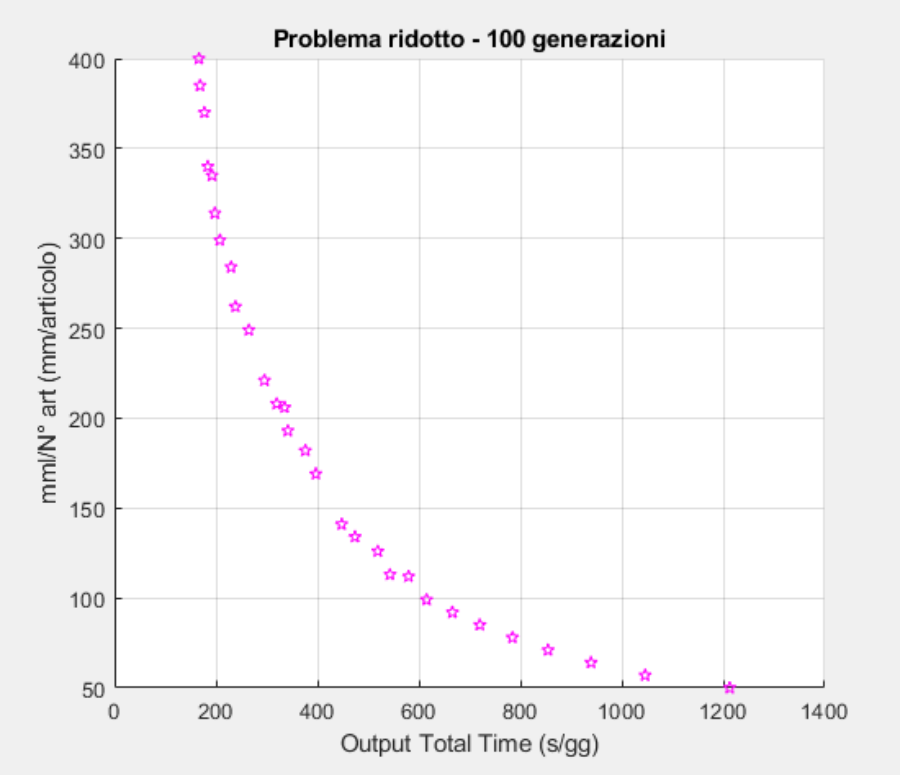
a



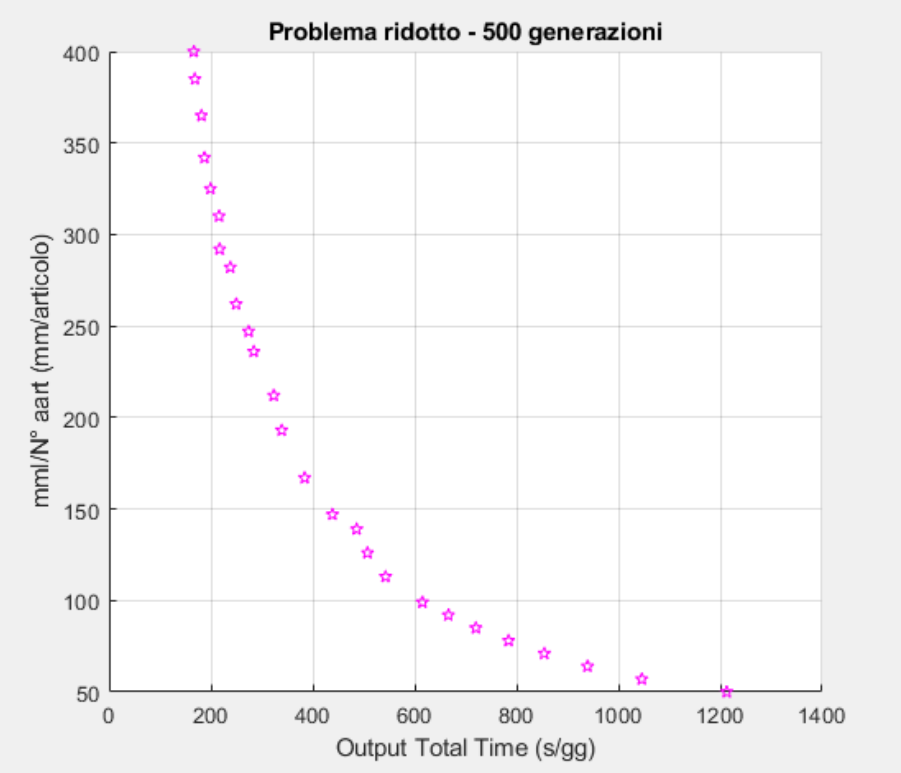
b



c


Problema ridotto - 100 generazioni
mml/N° art (mm/articolo)
Output Total Time (s/gg)
400
350
300
250
200
150
100
50
0
200
400
600
800
1000
1200
1400

d


Problema ridotto - 500 generazioni
mml/N° aart (mm/articolo)
Output Total Time (s/gg)
400
350
300
250
200
150
100
50
0
200
400
600
800
1000
1200
1400

La figura 3.5 mostra la variazione dei risultati al cambiamento del numero di generazioni. È interessante notare come già dal punto di vista esclusivamente visivo il fronte di Pareto ottimale assume una forma sempre più regolare e uniforme all'aumento del numero di generazioni. L'algoritmo è stato fatto girare con popolazione di 100 individui e per 500 generazioni massimo. L'ultima generazione calcolata corrisponde al numero 108, perché viene raggiunta la convergenza dell'algoritmo. I vari step a 10, 20, 100 e 108 mostrano come gradualmente NSGA-II si comporta: la popolazione si sposta gradualmente verso il fronte ottimale di Pareto ad ogni generazione. Il fronte ottimale di Pareto può essere considerato come la curva che meglio approssima la posizione del rank minore di popolazione alla generazione 108. La seguente tabella 3.1 mostra i parametri computazionali di output, interessanti a valutare l'efficienza dell'algoritmo.

Tabella 3.1: *Panoramica dei parametri computazionali per il problema ridotto*

Numero di generazioni	Numero funzioni obiettivo calcolate	Average distance	Spread	Tempo computazionale
10	1450	0.0586	0.3288	1.322
20	2950	0.0316	0.9946	1.745
100	14950	0.0220	0.1132	5.344
108	16150	0.0218	0.1098	5.486

La misura di Average distance cala all'aumentare delle generazioni, il che indica che le soluzioni sulla frontiera di Pareto sono uniformemente distribuite. La average distance è la deviazione standard della norma della differenza tra i membri del fronte di Pareto e la loro media. Lo spread va invece a misurare in modo quantitativo lo spostamento medio del fronte di Pareto e anche in questo caso si nota che lo spread diminuisce all'aumentare del numero di iterazioni. Il tempo computazionale (espresso in secondi) aumenta in modo considerevole col numero di generazioni (da 10 a 100 diventa quasi 4 volte maggiore), ma mantiene un valore accettabile per un problema di dimensioni ridotte.
Complessivamente l'algoritmo ha dato risposte positive per cui si può considerare sufficientemente affidabile per passare ad un problema di dimensioni maggiori.

3.3.3 Ottimizzazione delle UDC di tutte le linee di assemblaggio

Questo paragrafo testerà il modello costruito nel capitolo 2 andando ad utilizzare come input tutti gli articoli presenti sulle linee di assemblaggio. Il numero di UDC tra cui si può scegliere è pari a 5. Gli articoli complessivamente impiegati sulle linee di produzione sono 438. È importante precisare che, dal momento che tutte le linee di assemblaggio sono state considerate contemporaneamente, ogni articolo potrà essere disposto su più linee di assemblaggio (in base alla presenza o meno nella distinta base dei relativi prodotti finiti) ma all'interno della stessa UDC. Il consumo medio giornaliero è quello totale di tutte le linee.

Figura 3.6*: Output del problema completo delle linee di assemblaggio*

```
struct with fields:

      problemtype: 'linearconstraints'
         rngstate: [1×1 struct]
      generations: 142
        funccount: 21250
          message: 'Optimization terminated: average change in the spread of Pareto solutions less than options.FunctionTolerance.'
    maxconstraint: 0
  averagedistance: 0.0174
           spread: 0.1826
```

La figura 3.6 mostra che l'algoritmo ha effettuato 21250 valutazioni di funzioni obiettivo con 142 generazioni maturate. La average distance è più bassa rispetto al caso ridotto (0.0174 contro lo 0.0218), mentre il valore dello spread è più elevato ed è pari a 0.1826. Tale risultato sullo spread è attendibile perché il numero dimensionale del problema è molto più elevato (dunque il numero di soluzioni complessive genera una nuvola molto più grande).
L'uscita è avvenuta tramite il raggiungimento del limite di tolleranza per la variazione dello spread della frontiera di Pareto. Il numero di generazioni massime impostate è pari a 500, mentre il parametro di *StallGenLimit* è stato impostato pari a 50. Quest'ultimo parametro influisce sull'uscita dell'algoritmo in quanto lo spread ad ogni iterazione deve arrivare ad essere minore dello spread medio calcolato in tali generazioni precedenti.
Nella figura 3.7 viene mostrato il profilo di tempo computazionale correlato al problema appena affrontato. Si può notare come l'attività più dispendiosa in termini di tempo sia stata la valutazione di tutte le funzioni obiettivo a partire dalla popolazione di ogni generazione.

Figura 3.7: *Visualizzazione di un estratto del profilo dei tempi computazionali per la risoluzione dell'algoritmo genetico..*

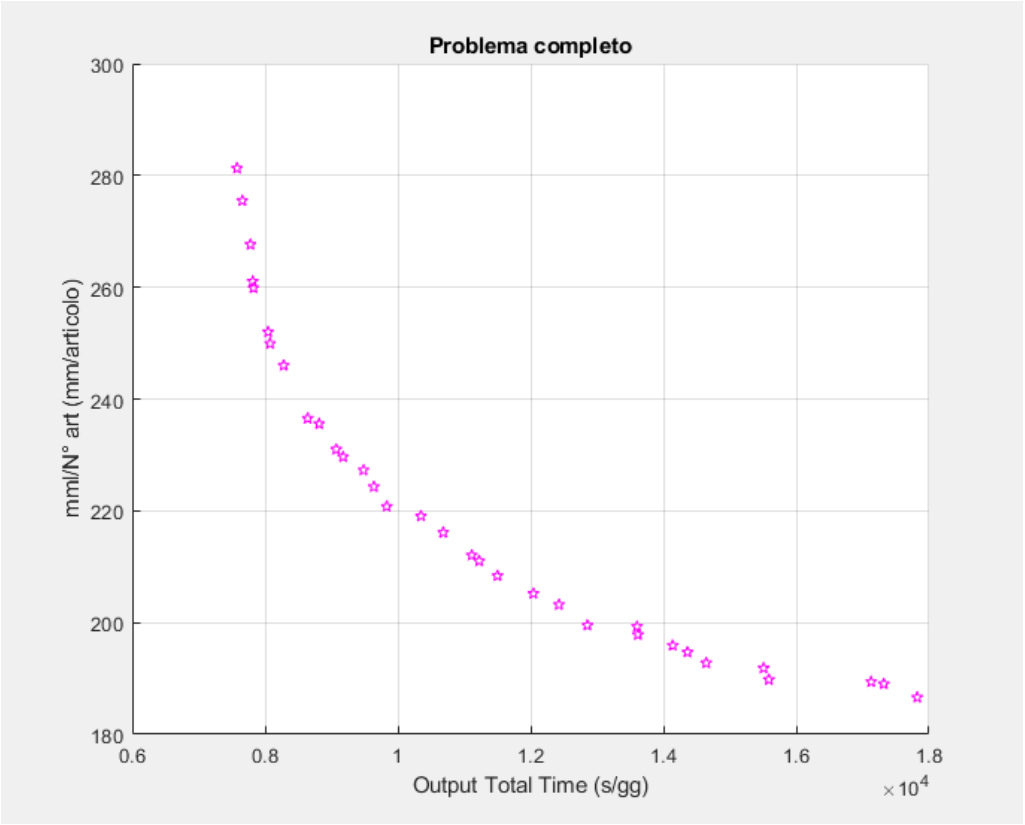
Profile Summary

Generated 02-Dec-2019 21:37:27 using performance time.

Function Name	Calls	**Total Time**	Self Time*	Total Time Plot (dark band = self time)
Lancio_GAMO	1	277.795 s	0.082 s	
Calcolo_Qp_Ott_GAMO	1	277.713 s	0.020 s	
gamultiobj	1	277.531 s	0.026 s	
globaloptim\private\gamultiobjsolve	1	276.388 s	0.030 s	
globaloptim\private\stepgamultiobj	141	270.623 s	1.004 s	
globaloptim\private\objAndConVectorizer	142	241.237 s	1.444 s	
...AnonymousFcn>@(x)fcn(x,FcnArgs{:})	21250	239.812 s	0.918 s	
Calcolo_Qp_Ott_GAMO>objval	21250	238.894 s	238.894 s	
...Score,thisPopulation,num_art,num_UDC)	141	20.996 s	0.009 s	
int_mutation	141	20.988 s	0.749 s	
int_mutation>mutation	141	20.238 s	20.238 s	
globaloptim\private\rankAndDistance	142	4.911 s	0.589 s	
globaloptim\private\gadsplot	143	4.310 s	1.383 s	
globaloptim\private\trimPopulation	141	4.161 s	0.008 s	

Tra le varie funzioni impostate per l'algoritmo genetico, quella che risulta impegnare maggiormente il processore è la *int_mutation*, quindi un possibile miglioramento di tale funzione potrebbe portare alla riduzione del tempo complessivo di calcolo.

Figura 3.7: *Visualizzazione della frontiera di Pareto per il problema di ottimizzazione delle UDC su tutte le linee di assemblaggio.*



La figura 3.7 mostra l'andamento della frontiera di Pareto. La curva ha un andamento sufficientemente parabolico e non presenta evidenti punti di discontinuità tra le soluzioni a rango minimo.
Si può procedere ora con la selezione della soluzione migliore tra le ottime proposte dall'algoritmo. Per effettuare la selezione si ricorre alla relazione 2.18. La larghezza minima media delle UDC è pari a 50 mm. La soluzione selezionata è mostrata nella tabella 3.2.

Tabella 3.2: *Soluzione selezionata per il problema di scelta ottimale delle UDC*

Tempo (s/gg)	mml/N° art (mm/articolo)	Valore funzione selezione
17823	**187**	**142.55**

Capitolo 4

Conclusioni

La tesi si conclude con la selezione della soluzione migliore del problema di ottimizzazione della scelta delle unità di carico applicata ad un sistema di assemblaggio. La curva di Pareto ha fornito un set di soluzioni ottime tra cui è stata scelta la migliore in base ad una funzione di selezione. L'algoritmo ha dunque fornito i risultati voluti andando a generare una curva di *trade-off* che ha messo in luce la natura conflittuale esistente tra il tempo medio giornaliero di uscita degli articoli dal magazzino e lo spazio occupato mediamente dagli articoli in linea.

Sicuramente tra i limiti del modello proposto vi è una mancata validazione reale, dal momento che è ancora in corso tale validazione. Attualmente ci si trova in una situazione di raccolta dati in seguito alla selezione della soluzione ottimale. Un altro limite può essere l'aver considerato un magazzino manuale diviso a due blocchi, quando invece esistono più disposizioni di magazzino e più tipologie. Potrebbe essere interessante studiare il caso con l'aggiunta di un magazzino automatico verticale (in tal caso entrerebbero in gioco ulteriori fattori come il volume e la massa degli articoli).

Appendice A: Function

Tempi_Mag

```
function [T] = Tempi_Mag(I,K,L,m,n,o,a,h,e,vh,vz)

%I è il numero di coppie di scaffali
%K è il numero di celle per ogni scaffale (senza contare corridoio
%trasversale)
%L è il numero di ripiani
%m è la distanza tra due coppie di scaffalature
%n è la larghezza di una coppia di scaffalature (due la volte la profondità
%di una scaffalatura)
%o è la larghezza del corridoio trasversale
%a è la distanza tra il blocco magazzino e il punto di accumulo degli
%articoli prelevati (punto Output)
%h è la larghezza di ogni cella
%e è l'altezza di ogni cella
%vh è la velocità orizzontale
%vz è la velocità verticale

T=[];
for i=1:I
   for j=1:2
      for l=1:L
         for k=1:K
            if j==1
               Dh=(i-1)*(m+n);
            else
```

```
            Dh=(i*n)+(i-1)*m;
          end
        Dh=Dh+a+o/2+h*abs(k-K/2-1/2);
        Add=Dh/vh+((l-1)*e)/vz;
        T=[T;Add];
        end
      end
    end
end
```

Appendice B: Function

Posiz_Mag

```
function [T, Posizione_Mag_Vett] = Posiz_mag(Param_Mag, Consumi, Qp)
%Inizializzo wp

wp=((Consumi(:,2)))'/diag(Qp);

A = cell2mat(Param_Mag);
%scompongo vettore Param_Mag
I = A(1);
K = A(2);
L = A(3);
m = A(4);
n = A(5);
o = A(6);
a = A(7);
h = A(8);
e = A(9);
vh = A(10);
vz = A(11);

%lancio function Tempi_Mag per calcolare tempi in base a Param_Mag
[T] = Tempi_Mag(I,K,L,m,n,o,a,h,e,vh,vz);
MT = (repmat(T,1,size(wp,2)))*diag(wp);
obj = MT(:);

%parametri per vincoli
```

```
num_art = size(wp,2);
num_celle = size(T,1);
num_DIM = num_art * num_celle;

%vincolo 1: ogni articolo può essere assegnato ad una sola cella di
%magazzino
onesvector = ones(1,num_celle);
Ar = repmat(onesvector, 1, num_art);
Ac = mat2cell(Ar, size(onesvector,1), repmat(size(onesvector,2),1,num_art));
Aeq = blkdiag(Ac{:});
beq = ones(num_art,1);

%vincolo 2: ogni cella di magazzino può ospitare solo un articolo al
%massimo
A = repmat(eye(num_celle),1,num_art);
b = ones(num_celle,1);

%definizione funzione lineare obiettivo
%obiettivo è minimizzare obj definita come la matrice dei tempi di prelievo
%pesata con il parametro wp
f = obj;
%le variabili devono essere binarie (0 o 1)
lb = zeros(size(f));
ub = lb + 1;
intvars = 1:length(f);
%lancio la risoluzione del problema tramite la funzione intlinprog
%x sarà la disposizione migliore degli articoli a magazzino
[x,fval,exitflag,output] = intlinprog(f,intvars,A,b,Aeq,beq,lb,ub);

%organizzazione output
Posizione_Mag_Vett = x;
Posizione_Mag = reshape(x,[num_celle,num_art]);

end
```

Appendice C: Function Int_pop

```
function Population = int_pop(GenomeLength, ~, options,num_art,num_UDC)
% INT_POP Function crea una popolazione iniziale in grado di
%soddisfare i vincoli di tipo binario

   A = options.LinearConstr.Aeq;
   b = options.LinearConstr.beq;
   curpop = zeros(0, GenomeLength);
   popsize = options.PopulationSize;


   while size(curpop,1) < popsize
      more_needed = popsize - size(curpop,1);
      for i=1:more_needed
         %deve creare individui fino a raggiungere popsize
         for a=1:num_UDC:GenomeLength
               %deve muoversi su tutta la lunghezza di Genoma
               %con passo il numero di UDC
               beta = randi([0,num_UDC-1]);
               %prendo un valore beta a caso per piazzare
               %l'articolo in una UDC
               x(i,a:a+num_UDC-1) = 0;
               x(i,a+beta) = 1;
         end
      end
      thisbatch = x;
      %effettuo un check per controllare che i vincoli siano
      %rispettati
      passfail = all(A * thisbatch.' == b, 1);
      goodones = thisbatch(passfail,:);
```

```
        curpop = [curpop; goodones];
    end
    Population = curpop;
end
```

Appendice D: Function

Crossover

```
function xoverKids =
int_crossoverarithmetic(parents,options,GenomeLength,...
   FitnessFcn,unused,thisPopulation,num_art,num_UDC)

   %int_crossoverarithmetic è una funzione che genera gli individui figli
   %a partire da coppie di individui genitori

   A = options.LinearConstr.Aeq;
   b = options.LinearConstr.beq;
   curpop = zeros(0, GenomeLength);
   popsize = options.PopulationSize;

   %genero individui figli
   while size(curpop,1) < popsize/2
     more_needed = popsize/2 - size(curpop,1);
     thisbatch =
crossdp(parents,more_needed,thisPopulation,num_art,num_UDC);
     %effettuo un check per determinare l'accettabilità dei figli in base
     %ai vincoli impostati
     passfail = all(A * thisbatch.' == b, 1);
     goodones = thisbatch(passfail,:);
     curpop = [curpop; goodones];
   end

   xoverKids = curpop;
```

```
function [x] = crossdp(parents,more_needed,thisPopulation,num_art,num_UDC)
    % quanti figli produrre
    nKids = more_needed;
    % Preallocazione figli
    x = zeros(nKids,GenomeLength);
    % Per muoversi più rapidamente tra gli individui genitori utilizzo
    % un indice diverso
    index = 1;
    % per ogni figlio
    for i=1:nKids
       % prendo i genitori
       r1 = parents(index);
       index = index + 1;
       r2 = parents(index);
       index = index + 1;
       % i figli sono generati mescolando il patrimonio genetico dei
       % genitori
       % RANDI garantisce che siano integer
       %prendo i due punti di crossover
       alpha1 = randi([1,num_art])*num_UDC;
       alpha2 = alpha1;
       while alpha2 == alpha1
          alpha2 = randi([1,num_art])*num_UDC;
       end
       k1 = min(alpha1,alpha2);
       k2 = max(alpha1,alpha2);
       %ora vado a costruire il figlio usando due pezzi dal primo
       %genitore ed un pezzo dal secondo
       for a=1:k1
          x(i,a) = thisPopulation(r1,a);
       end
       for a=(k1+1):k2
          x(i,a) = thisPopulation(r2,a);
       end
       for a=(k2+1):GenomeLength
```

```
            x(i,a) = thisPopulation(r1,a);
        end

    end
  end

end
```

Appendice E: Function

Mutazione

```
function mutationChildren = int_mutation(parents, options, GenomeLength, ...
    ~, state, ~, thisPopulation,num_art,num_UDC)

% Function che genera i codici figli mutati
% Soddisfa i vincoli lineari impostati

    A = options.LinearConstr.Aeq;
    b = options.LinearConstr.beq;
    curpop = zeros(0, GenomeLength);
    popsize = options.PopulationSize;

    while size(curpop,1) < popsize
      more_needed = popsize - size(curpop,1);
      thisbatch = mutation(parents, more_needed,
thisPopulation,num_art,num_UDC);
      %effettuo un chekc per verificare che tutti i figli soddisfino i
      %vincoli impostati
      passfail = all(A * thisbatch.' == b, 1);
      goodones = thisbatch(passfail,:);
      curpop = [curpop; goodones];
    end

    mutationChildren = curpop;

    %funzione che seleziona e modifica i geni dei genitori
```

```
  function [x] = mutation(parents, more_needed,
thisPopulation,num_art,num_UDC)
      x = zeros(more_needed,GenomeLength);
      index = 1;
      for i=1:min(more_needed,size(thisPopulation,1))
        %index serve per prelevare gli individui genitori dalla
        %popolazione
        index = index + 1;
        for a=1:num_UDC:GenomeLength
          %deve muoversi su tutta la lunghezza di Genoma
          %con passo il numero di UDC, in modo che le modifiche
          %riguardino il tratto di corredo genetico di un solo
          %articolo
          alpha = randi([0,1]);
          if alpha == 0 %rimane invariato tutto il tratto genetico di quell'articolo
            x(i,a:a+num_UDC-1) = thisPopulation(i,a:a+num_UDC-1);
          else %altrimenti deve spostare l'1 in un'altra posizione casuale
            %sempre di quel tratto genetico dell'articolo
            beta = randi([0,num_UDC-1]);
            x(i,a:a+num_UDC-1) = 0;
            x(i,a+beta) = 1;
          end
        end
      end
  end
end
```

Appendice F: Algoritmo Gamultiobj

```
function [x_ga1,fval_ga1,gaoutput1] = Calcolo_Qp_Ott_GAMO(UDC,Q_ALL,Consumi,Posizione_Mag_Vett,T)  %4 tabelle in input
%utilizzo l'algoritmo genetico gamultiobj per minimizzare 2 obiettivi
%contemporaneamente

%parametri per vincoli
num_art = size(Consumi,1);
num_UDC = size(UDC,1);
num_DIM = num_art * num_UDC;

%assegnazione funzioni obiettivo
fun = @objval;

    %function che definisce le funzioni obiettivo
    function F = objval(x)
    %calcolo Tout
    Qeff = diag(diag(Q_ALL*(reshape(x,[num_UDC,num_art]))));
    wp = (Consumi(:,2))'/Qeff;
    MT = (repmat(T,1,size(wp,2)))*diag(wp);
    f1 = MT(:)'*Posizione_Mag_Vett;
    %calcolo del parametro metri lineari/numero articoli
    L = UDC(:,2)';
    L = repmat(L,size(Consumi,1),1);
    Leff = diag(L*(reshape(x,[num_UDC,num_art])));
```

```
   f2 = sum(Leff)/num_art;

   F = [f1;f2];
   end

%vincolo 1: ogni articolo può essere assegnato ad una sola UDC
onesvector = ones(1,num_UDC);
Ar = repmat(onesvector, 1, num_art);
Ac = mat2cell(Ar, size(onesvector,1), repmat(size(onesvector,2),1,num_art));
Aeq = blkdiag(Ac{:});
beq = ones(num_art,1);

%le variabili devono essere binarie (0 o 1)
lb = zeros(num_DIM,1);
ub = lb + 1;
Bound = [lb ub]';
intvars = num_DIM;

%lancio la risoluzione del problema tramite la funzione gamultiobj
%x sarà la disposizione migliore degli articoli nelle UDC

%inizializzazione dei parametri di input dell'algoritmo
populationSize = 100;
stallGenLimit = 50;
generations = 500;
%definizione delle options
opts_ga = optimoptions('gamultiobj','Display','off','PlotFcn','gaplotpareto',...
   'PopulationSize',populationSize,...
   'CreationFcn', @(GenomeLength, FitnessFcn,
options)int_pop(GenomeLength, FitnessFcn, options,...
   num_art,num_UDC),...
   'CrossoverFcn', @(parents,options,GenomeLength,...

FitnessFcn,unused,thisPopulation)int_crossoverarithmetic(parents,options,Gen
omeLength,...
   FitnessFcn,unused,thisPopulation,num_art,num_UDC),...
   'MutationFcn', @(parents, options, GenomeLength, ...
```

```
   FitnessFcn, state, thisScore, thisPopulation)int_mutation(parents, options, GenomeLength, ...
   FitnessFcn, state, thisScore, thisPopulation,num_art,num_UDC),...
   'StallGenLimit', stallGenLimit,...
   'Generations', generations,...
   'PopInitRange', Bound);

%comando per lanciare l'algoritmo
[x_ga1,fval_ga1,~,gaoutput1] = gamultiobj(fun,intvars,[],[],…
        Aeq,beq,lb,ub,opts_ga);

end
```

Bibliografia

Beltrametti, L., N. Guarnacci, N. Intini, C. La Forgia (2017). *La fabbrica connessa*. Guerini, Milano.

Carson, F., G. Marchet e A. Perego (1998). Routing policies and COI-based storage policies in picker-to-part systems. *International Journal of Production Research*, **36**, 713-732.

Chackelson, C., A. Errasti, D. Ciprés e F. Lahoz (2013). Evaluating order picking performance trade-offs by configuring main operating strategies in a retail distributor: A Design of Experiments approach. *International Journal of Production Research*, **51**, 6097-6109.

Chan, F.T.S. e H.K. Chan (2011). Improving the productivity of order picking of a manual-pick and multi-level rack distribution warehouse through the implementation of class-based storage. *Expert Systems with Applications*, **38**, 2686-2700.

Chen, C.M., Y. Gong, R.B.M. de Koster e J.A.E.E. van Nunen (2010). A flexible evaluative framework for order picking systems. *Production and Operation Management*, **19**, 70-82.

Chen, F., H. Wang, Y. Xie e C. Qi (2016). An ACO-based online routing method for multiple order pickers with congestion consideration in warehouse. *Journal of Intelligent Manufacturing*, **27**, 389-408.

Chen, F., Y. Wei e H. Wang (2018). A heuristic based batching and assigning method for online customer orders. *Flexible Services and Manufacturing Journal*, **30**, 640-685.

Chen, T.L., C.Y. Cheng, Y.Y. Chen, L.K.Chan (2015). An efficient hybrid algorithm for integrated order batching, sequencing and routing problem. *International Journal of Production Economics*, **159**, 158-167.

Cheng, C.Y., Y.Y. Chen, T.L. Chen e J.J.W. Yoo (2015). Using a hybrid approach based on the particle swarm optimization and ant colony optimization to solve a joint order batching and picker routing problem. *International Journal of Production Economics*, **170**, 805-814.

Davarzani, H. e A. Norrman (2015). Toward a relevant agenda for warehousing research: literature review and pratictioners' input. *Logistics Research*, **8**, 1-18.

De Koster, R. B. M., A. L. Johnson e D. Roy (2017). Warehouse design and management. *International Journal of Production Research*, **55**, 6327-6330.

De Santis, R., R. Montanari, G. Vignali e E. Bottani (2018). An adapted ant colony optimization algorithm for the minimization of the travel distance of pickers in manual warehouses. *European Journal of Operational Research*, **267**, 120-137.

De Toni, A. F. e R. Panizzolo (2018). *Sistemi di gestione della produzione*. Isedi, Novara, p.28.

Deb, K. (2011). Multi-Objective Optimization using Evolutionary Algorithms: An introduction. *Multi-Objective Evolutionary Optimization for Product Design and Mnaufacturing*. Springer. London, pp 3-34.

Ene, S. e Öztürk, N. (2012). Storage location assignment and order picking optimization in the automotive industry. *The International Journal of Advanced Manufacturing Technology*, **60**, 787–797.

Gamultiobj algorithm, https://www.mathworks.com/help/gads/gamultiobj-algorithm.html, (05/10/2019).

Helsgaun, K. (2000). *An effective im plementation of the Lin–Kernighan traveling salesman heuristic*. European Journal of Operational Research, **126**, 106–130.

Ho, Y.-C., & Tseng, Y.-Y. (2006). *A study on order-batching methods of order-picking in a distribution centre with two cross-aisles*. International Journal of Production Research, **44**, 3391–3417.

Ho, Y.-C., Su, T.-S., & Shi, Z.-B. (2008*). Order-batching methods for an order-picking warehouse with two cross aisles*. Computers & Industrial Engineering, **55**, 321–347.

Kulak, O., Sahin, Y., & Taner, M. E. (2012). *Joint order batching and picker routing in single and multiple-cross-aisle warehouses using cluster-based tabu search algorithms*. Flexible Services and Manufacturing Journal, **24**, 52–80.

Manzini, R., Gamberi, M., Persona, A., & Regattieri, A. (2007). *Design of a class based storage picker to product order picking system*. The International Journal of Advanced Manufacturing Technology, **32**, 811–821.

Pareschi, A., E. Ferrari, A. Persona e A. Regattieri (2014). *Logistica integrata e flessibile per i sistemi produttivi dell'industria e del terziario con*

applicazioni numeriche e progettuali. Società Editricie Esculapio, Bologna, p.312.

Petersen, C. G. (1997). *An evaluation of order picking routing policies*. International Journal of Operations & Production Management, **17**, 1098–1111.

Petersen, C. G., & Schmenner, R. W. (1999). *An evaluation of routing and volume- based storage policies in an order picking operation*. Decision Sciences, **30**, 481–501.

Ratliff, H.D., & A.S. Rosenthal (1983). *Order-picking in a rectangular warehouse: A solvable case of the travelling salesman problem*. Operations Research, **31**, 507-521.

Rouwenhorst, B., B. Reuter, V. Stockrahm, G. J. van Houtum, R. J. Mantel e W. H. M. Zijm (2000). Warehouse design and control: Framework and literature review. *European Journal of Operational Research*, **122**, 515-533.

Shqair, M., Altarazi, S., & Al-Shihabi, S. (2014). *A statistical study employing agent- based modeling to estimate the effects of different warehouse parameters on the distance traveled in warehouses*. Simulation Modelling Practice and Theory, **49**, 122–135.

Staudt, F. H., G. Alpan, M. Di Mascolo e C. M. T. Rodriguez (2015). Warehouse performance measurement: a literature review. *International Journal of Production Research*, **53**, 5524-5544.

Tsai, C. Y., Liou, J. J. H., & Huang, T. M. (2008). *Using a multiple-GA method to solve the batch picking problem: Considering travel distance and order due time*. International Journal of Production Research, **46**, 6533–6555.

Van Gils, T., K. Ramaekers, A. Caris, R. B. M. de Koster (2018). Designing efficient order picking systems by combining planning problems: State-of-the-art classification and review. *European Journal of Operational Research*, **267**, 1-15.