UNIVERSITÀ DI PISA

DIPARTIMENTO DI INGEGNERIA DELL'ENERGIA DEI SISTEMI, DEL TERRITORIO E DELLE COSTRUZIONI

RELAZIONE PER IL CONSEGUIMENTO DELLA LAUREA MAGISTRALE IN INGEGNERIA GESTIONALE

VALIDAZIONE E MIGLIORAMENTO DI UN ALGORITMO GENETICO PER IL BILANCIAMENTO DELLE LINEE DI ASSEMBLAGGIO MANUALI

RELATORI
Prof. Ing. Gino Dini
Dipartimento di Ingegneria Civile e Industriale

Ing. Michela Dalle Mura
Dipartimento di Ingegneria Civile e Industriale

IL CANDIDATO
Adriana Guinicelli
guinicelliadriana@gmail.com

Sessione di Laurea del 27/04/2016
Anno Accademico 2015/2016
Consultazione consentita

Quanto più ci innalziamo, tanto
più piccoli sembriamo a quelli che
non possono volare.

(Friedrich Nietzsche)

Sommario

Il presente lavoro di tesi si focalizza sulla validazione e il miglioramento di un algoritmo genetico, implementato in Matlab, per il bilanciamento delle linee di assemblaggio manuali. Il lavoro si suddivide in due sezioni distinte, la validazione dell'algoritmo per problemi mono-obiettivo e la validazione per problemi multi-obiettivo. La prima fase concerne la definizione della tipologia di validazione utilizzata per testare la bontà dell'algoritmo. Si sono quindi trascritti i dati necessari su Matlab e si è fatto girare l'algoritmo per comprenderne il comportamento. I risultati ottenuti sono stati analizzati e discusse le problematiche scaturite. Sono state proposte alcune modifiche per risolvere i problemi mono-obiettivo e sono stati confrontati i risultati con i valori ottenuti in precedenza. Stessa procedura è stata eseguita per validare l'algoritmo quando utilizzato per raggiungere contemporaneamente più obiettivi. I risultati che si sono riscontrati in entrambe le tipologie di problemi riguardano un miglioramento evidente delle soluzioni che l'algoritmo è in grado di trovare rispetto ai risultati precedenti.

Abstract

The following thesis is focused on the validation and improvement of a genetic algorithm, implemented on Matlab, for the assembly manual line balancing problem. The work is divided into two distinct sections, the validation of the algorithm for mono-objective and validation for multi-objective problems. The first stage concerns the definition of the type of validation used to test the algorithm's goodness. Then, the necessary data are transcribed on Matlab, on which the algorithm has been implemented. The algorithm had been turn in order to understand the behavior. The results obtained were analyzed and the issues arising were discussed. Have been proposed some changes to address the mono-objective problems.The results obtained were compared with those that the algorithm had obtained previously. The same procedure has been done to resolve the difficulties in the multi-objective problems. The results that were found in both types of problems relate a clear improvement of the solutions that the algorithm is able to find compared to previous results.

Indice

Capitolo 1

1 La linea di assemblaggio

1.1 Definizione

Le linee di assemblaggio rappresentano i componenti chiave dei moderni sistemi produttivi; sono sistemi a flusso, tipici della produzione industriale di prodotti standardizzati in grandi quantità; appartengono alle tecniche di produzione dette "di massa" o "a flusso", caratterizzate da una disposizione sequenziale dei macchinari, tale da rispecchiare la sequenza tecnologica delle operazioni necessarie al completamento del prodotto.

In letteratura vengono, per questo motivo, indicati come sistemi produttivi *process-oriented*, in contrapposizioni ai sistemi *job-oriented* dove, invece, i macchinari sono raggruppati per funzione.

Il primo esempio di linea di assemblaggio risale all'idea di William Klann, dipendente di Henry Ford, che partorì l'idea dall'osservazione dei lavoratori. Egli notò come gli operatori si intralciavano a vicenda nell'operare attorno all'auto in costruzione e, parallelamente, fece alcune analisi

Figura 1.1 Linea di assemblaggio Henry Ford

circa l'efficienza che si sarebbe potuta raggiungere attraverso la ripetizione nel tempo delle medesime operazioni.

Non è corretto affermare che egli fu l'inventore della linea di assemblaggio, infatti è appurato che alcuni anni prima delle linee Ford, la linea di assemblaggio fu brevettata da Ransom Eli Olds, fondatore della "Olds Motors Vehicle Company". Il successo del Model T fu, però, di tale portata da permettere all'azienda e a Ford, di fregiarsi del titolo di inventore della linea di assemblaggio.

Insieme alla sua invenzione si presentarono le prime problematiche, tutt'oggi presenti nelle linee di assemblaggio moderne. Fra queste è interessante il caso del collo di bottiglia causato dalla stazione addetta alla verniciatura nella linea Ford. Al tempo, l'unico colore, che era in grado di garantire tempi di asciugatura sufficientemente rapidi, era il nero. Ford, non trovando altre soluzioni tecniche o organizzative, decise dunque la produzione del solo modello nero, da cui la celebre frase riferita al Model T: "disponibile in tutti i colori purché nera".

La comprensione delle linee di assemblaggio richiede una notevole attenzione, poiché sia i criteri che le differenti tipologie, che da essi derivano, sono numerosi; per tale motivo, di seguito, se ne da una breve descrizione, necessaria per affrontare il lavoro di tesi.

1.2 Terminologia

In questa sezione vengono illustrati i concetti generali utilizzati in tutto il lavoro di tesi e necessari per un efficare comprensione e descrizione delle linee di assemblaggio:

- *Operazione (o task)*: è una frazione del lavoro totale necessario ad ottenere il prodotto finito. Le operazioni sono considerate indivisibili, nel senso che non possono essere suddivise in operazioni più elementari;
- *Tempo dell'operazione (task time)*: tempo neccesario per svolgere un'operazione;

- *Stazione di lavoro (workstation)*: è una porzione della linea di assemblaggio nella quale viene svolta una certa quantità di lavoro. I suoi elementi identificativi sono: le attrezzature e i macchinari, nonché il tipo di lavoro assegnato;
- *Rateo produttivo (production rate)*: quantitativo di pezzi da produrre nell'unità di tempo;
- *Efficienza delle linea (line efficienty)*: calcolata come segue $\eta_L = \frac{R_p}{R'_p}$, in cui il numeratore è il rateo di produzione che si avrebbe con le perdite della linea, il denominatore rappresenta il rateo produttivo ideale;
- *Tempo ciclo (cycle time)*: rappresenta la massima quantità di tempo che un pezzo può sostare in una stazione e può essere calcolato $T_c = \frac{60*\eta_L}{R_p}$;
- *Carico di lavoro (workload)*: somma dei task time delle operazioni assegnate ad una workstation;
- *Buffer*: locazioni fisiche usate per lo store temporaneo dei work in process (WIP);
- *Tempo morto (idle time)*: differenza tra il carico di una stazione e il tempo ciclo;
- *Layout*: disposizione delle workstations all'interno della linea di assemblaggio.

1.3 Classificazione

In letteratura generalmente le linee di assemblaggio vengono classificate rispetto alcuni fattori critici che le caratterizzano e che devono essere fissati e/o gestiti durante la loro progettazione. Di seguito la classificazione fornita da Scholl [1] ed integrata con altri elementi ritenuti importanti.

1.3.1 Tempo ciclo

In riferimento al tempo ciclo si crea una prima distinzione tra linee *sincrone* e *asincrone*. Nelle linee sincrone il tempo allocato al completamento di ogni operazione

di assemblaggio è il medesimo per ogni stazione. Ogni stazione disporrà dunque della stessa quantità di tempo per portare a termine l'operazione. Anche la linea, nel suo complesso, avrà un tasso di produzione costante e uscirà un prodotto completato ad ogni intervallo di tempo pari a T_c. Nelle linee asincrone le stazioni possono avere tempi ciclo diversi e, dunque, diversi tassi di produzione. Vanno dunque previsti dei buffer fra stazioni successive per permettere la sosta dei semilavorati in attesa alla stazione successiva.

1.3.2 Numero di prodotti

La seconda dimensione utilizzabile per poter eseguire una classificazione delle linee di assemblaggio è il numero di prodotti assemblati sulla stessa linea. Si possono distinguere tre tipologie di linee:

1) *la linea monoprodotto* (o dedicata) che tratta un solo prodotto lungo la linea;

Figura 1.2 Schema linea single-model

2) *la linea multi-model*, in cui si alterna lungo la linea la produzione di prodotti similari. A causa di significative differenze nel processo produttivo è necessario un setup della linea ad ogni variazione;

Figura 1.3 Schema linea multi-model

3) *la linea mixed-model*, in cui sono assemblate molte versioni di un "prodotto base". I prodotti differiscono solo in alcuni attributi o per caratteristiche opzionali.

Figura 1.4 Schema linea mixed-model

1.3.3 Grado di automazione

A seconda del tipo di attrezzature e del sistema di controllo di produzione, le stazioni possono avere diversi gradi di automazione. Si possono quindi distinguere:

- *stazioni manuali*, in cui le operazioni sono svolte da un operatore umano che utilizza semplici attrezzi o macchine utensili;
- *stazioni semi automatiche*, in cui l'intervento umano è limitato alle operazioni di controllo e rifornimento di materiale per i macchinari, i quali svolgono l'operazione in modo automatico;
- *stazioni totalmente automatizzate*, in cui ogni operazione, compreso il controllo e il rifornimento di materiale, viene svolto totalmente dai macchinari, senza la supervisione di operatori umani.

1.3.4 Layout

Il layout della linea è in gran parte predeterminato dalla sequenza di lavorazioni necessarie al corretto fluire dei semilavorati. Nonostante questo, vi sono delle possibili varianti, illustrate nella seguente tabella, in cui vengono dettagliati i vantaggi e gli svantaggi di ognuno in riferimento al contesto industriale in cui comunemente si ritrovano ad operare.

TIPO DI LAYOUT	VANTAGGI	SVANTAGGI	CONTESTO INDUSTRIALE
Linee seriali	• Semplici da controllare; • Bassa movimentazione manuale; • Alta utilizzazione attrezzature.	• Poco adatto al lavoro di gruppo; • Poca flessibilità; • Scarso contatto visivo tra gli operatori; • Monotonia del lavoro.	Settore meccanico ed elettronico
Linee dritte con stazioni in parallelo	• Aumento della produttività della linea • Più efficienti rispetto alle stazioni in serie in quanto le stazioni sono indipendenti fra loro;	• Maggiori complicazioni nello scheduling; • Richiede maggiori spazi; • Maggiori costi nei sistemi di trasporto dei materiali; • Possibilità di perdere la sequenza dei prodotti all'uscita delle stazioni	Settore meccanico e automotive
Linee ad U	• Miglioramenti nella visibilità e nella comunicazione; • Flessibilità dei volumi; • Percorsi più corti all'interno del sistema;	• Utilizzazione ridotta o impossibile nei tratti a nastro delle due aree d'angolo; • Messa a disposizione dei pezzi di grandi dimensioni nei posti di lavoro più difficoltosa.	Assemblaggio di prodotti meccanici ed elettronici

Figura 1.5 Linee di assemblaggio in funzione del layout

1.4 Problema del Bilanciamento delle linee di assemblaggio

Il bilanciamento delle linee di assemblaggio è uno dei problemi principali che ogni giorno le imprese devono affrontare; una linea bilanciata, infatti, aumenta la produttività in termini di tempi e di costi.

Nel dettaglio, bilanciare una linea di montaggio vuol dire allocare le operazioni, necessarie alla processazione del prodotto, sulle stazioni allo scopo di minimizzare i tempi di attesa, gli sprechi, le risorse umane e fisiche utilizzate.

L'assegnamento delle operazioni non può avvenire indifferentemente all'interno di ogni stazione ma deve rispettare il tempo ciclo a disposizione di ogni workstation e i vincoli di precedenza tra le operazioni. Questi vincoli vengono rappresentati tramite un grafo delle precedenze.

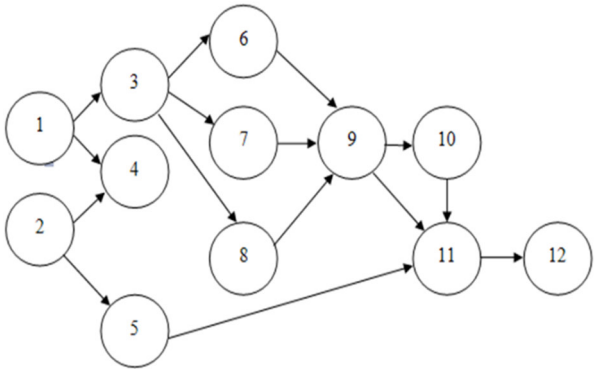

Figura1.6 Grafo delle precedenze

Il grafo contiene un nodo per ogni operazione, un peso per ogni nodo rappresentante il task time di ogni singola operazione ed un arco che definisce le precedenze. Per esempio, in Figura 1.6, le operazioni numero 6 e 7 devono essere completate prima che cominci l'operazione numero 9.

1.4.1 Definizione

Il primo articolo riguardo ALBP (Assembly Line Balanicng Problem) fu pubblicato nel 1955, in cui si proponeva una risoluzione del problema tramite programmazione lineare. Da allora, nella ricerca, si è sviluppato un grande interesse nell'ambito del bilanciamento delle linee di montaggio.

Tuttavia, i problemi ALB appartengono alla classe dei problemi di ottimizzazione di tipo NP-hard[1], per tale motivo si sviluppano costantemente efficienti algoritmi per ottenere soluzioni ottimali.

1.4.2 Classificazione dei problemi di bilanciamento

Esistono differenti tipologie di problemi in ALBP. Quella che segue è una delle possibili classificazioni, in cui si delineano due problemi classici: il semplice (SALBP) e il generale(GALBP).

I problemi SALBP vengono utilizzati per linee single-model; considerando i vincoli di precendenza e le operazioni indivisibili. In questi problemi il tempo di processamento delle operazioni è considerato indipendente dalla tipologia di stazione e dall'ordine in cui vengono processate le operazioni, i task time sono deterministici e conosciuti apriori. Esistono quattro differenti tipologie di SALBP:

- **Tipo-1(SALBP-1):** primo problema di ottimizzazione. Consiste nell'assegnazione delle operazioni alle stazioni con l'unico obiettivo di minimizzare il numero di workstations, dato un tempo ciclo fissato;
- **Tipo-2(SALBP-2):** secondo problema di ottimizzazione. Consiste nell'assegnare le operazioni alle stazioni minimizzando il tempo ciclo, con un numero di stazioni fissato;
- **Tipo-E(SALBP-E)**: terzo problema di ottimizzazione. Consiste nella massimizzazione dell'efficienza della linea, o in altre parole nella minimizzazione simultanea del tempo ciclo e del numero di stazioni;
- **Tipo-F(SALBP-F)**: quarto problema di ottimizzazione. Consiste nel determinare se esiste una soluzione flessibile per una determinata combinazione di un

[1] I problemi di ottimizzazione possono essere classificati in funzione della complessità computazionale rischiesta per risolverli. Un problema appartiene alla classe NP quando può essere risolto da algoritmi di tipo polinomiale non deterministici. I problemi che appartengono alla classe NP-hard sono risolti invece tramite algoritmi che possono essere convertiti in algoritmi per risolvere qualunque problema appartenente alla classe NP.

numero di stazioni pari a *m* e un tempo ciclo *c*, in altre parole, se la linea può operare con *m* stazioni e un tempo ciclo fissato. È un problema di flessibilità che stabilisce se esiste o meno una linea di assemblaggio flessibile per un dato tempo ciclo e un numero di operazioni.

I GALBP sono problemi che non incudono i SALBP. Quando i task time sono variabili, le stazioni sono poste parallelamente o altre differenze rispetto alle tipologie citate sopra. Esistono quattro differenti problemi GALBP:

- **UALBP (U-Shaped Assembly Line Balancing Problem)**: problemi di bilanciamento per linee ad U, in queste linee è possibile assegnare le operazioni di cui i predecessori o successori sono già stati allocati;
- **MALBP (Mixed-model Assembly Line Balancing Problem)**: sono problemi di bilanciamento mixed-model. Si presentano questi problemi quando si devono realizzare diversi modelli di uno stesso prodotto, e si ha un set di operazioni base da processare su tutti i modelli;
- **RALBP (Robot Assembly Line Balancing Problem)**: sono problemi di linee di assemblaggio in cui sono presenti robot per il processamento delle operazioni. In questo tipo di problemi si considera non solo l'allocazione delle operazioni ma anche l'allocazione di un robot ad ogni stazione;
- **MOALBP (Multi-Objective Assembly Line Balancing Problem)**: un problema di bilanciamento multi-obiettivo. Si considerano simultaneamente diversi obiettivi come la minimizzazione dei costi totali e il numero delle stazioni, o la massimizzazione dell'efficienza della linea.

Capitolo 2

2 Metodologie per risolvere ALBP

2.1 Euristiche

Un metodo euristico è una procedura che può trovare una soluzione buona per una data classe di problemi, ma non necessariamente una soluzione ottima.

I metodi euristici sono usati per sviluppare buone soluzioni per i problemi di bilanciamento delle linee di assemblaggio.

Il *Largest-Candidate rule* è l'euristica più semplice da comprendere. Le operazioni da assegnare alle stazioni sono selezionate semplicemente sulla base dell'entità del loro task time: maggiore è la durata maggiore è la possibilità di assegnare l'operazione alla stazione corrente, considerando sempre i vincoli di precedenza. La procedura è la seguente:

1. Creare una lista delle operazioni in ordine decrescente in funzione del task time;
2. Assegnare le operazioni alla prima stazione, iniziando dalla prima della lista. Si procede allocando alla stazione le operazioni che rispettano le precedenze e che non permettono di superare il tempo ciclo;
3. Continuare il processo assegnando gli elementi alla stazione come nello step 2, fino a quando è possibile assegnare operazioni alla stazione;
4. Ripetere gli step 2 e 3 per le altre stazioni nella linea fino a quando tutti gli elementi della lista sono stati assegnati.

L'*Incremental Utilization Heuristic* assegna le operazioni ad una workstation in un determinato ordine di precedenza. Le operazioni sono allocate ad una stazione fino a quando la sua utilizzazione raggiunge il 100%. L'utilizzazione, detta anche efficienza, è la percentuale di tempo in cui la linea produttiva, o stazione, lavora, ed è calcolata dall'equazione seguente:

$$E(\%) = \frac{\sum t}{n*c} * 100 \qquad \text{(eq. 1)}$$

Dove:

n = numero di stazioni;

c = tempo ciclo;

$\sum t$ = tempo totale richiesto per assemblare ogni unità.

Gli step da seguire sono i medesimi del metodo *Largest-Candidate rule,* ciò che è diverso è il criterio di stop nel riempimento delle stazioni, non più il tempo ciclo ma il grado di utilizzazione della stazione.

Un'altra euristica è *Maximum Task Time Heuristic*. Le operazioni sono allocate alla stazione, una alla volta, seguendo l'ordine di precedenza delle operazioni. Se è possibile allocare una o più operazioni ad una stessa stazione, è scelta quella con una durata maggiore. Questa euristica segue tre step:

1. Preso *i*=1, in cui *i* è il numero della stazione che si sta considerando;
2. Fare una lista di tutte le operazioni candidate ad essere assegnate a quella stazione. Per essere in quella lista, l'operazione deve soddisfare le seguenti condizioni: a) non deve essere stata assegnata a precedenti stazioni; b) i predecessori sono stati assegnati ad altre stazioni; c) la durata totale dell'operazione e di tutte le altre stazioni già assegnate alla stazione deve essere minore o uguale alla durata del tempo ciclo. Se non si trova nessuna operazione, si procede con lo step 4;
3. Collocare sulla stazione, l'operazione che nella lista ha la maggiore durata;

4. Non assegnare altre operazioni alla stazione. Questo può essere compiuto in due modi. Se non ci sono candidati nella lista della stazione, ma ci sono altre operazioni da assegnare, imposta $i=i+1$ e ritorna allo step 2. Se non ci sono altre operazioni da assegnare, la procedura è conclusa.

Slack, Chambers e Johnston hanno delineato una procedura che segue gli stessi passi (1,2 e 4) del Maximum Task Time Heuristic, tuttavia, nello step 3, la scelta dell'operazione da assegnare avviene calcolando il numero di operazioni che possono essere assegnate alla stazione successivamente a quella considerata. La procedura infatti prende il nome di *Number of Followers Heuristic*.

Farners e Pereira usano il *Positional Weight Heuristic* in cui la scelta dell'operazione si basa sulla valutazione di un indice che non è altro che la somma dei tempi delle operazioni successive a quella da assegnare. Le operazioni sono allocate in ordine decrescente in base a tale valore. Gli step dell'euristica sono i seguenti:

1. Per ogni operazione si calcola il suo indice;
2. Si crea una lista delle operazioni ordinandole in maniera decrescente secondo l'indice;
3. Si assegnano gli elementi alle stazioni secondo il loro indice, rispettando le precedenze e il tempo ciclo.

COMSOAL è un'euristica sviluppata da Chrysler. Questo metodo assegna random le operazioni alle stazioni e ad ogni iterazione, compara la soluzione corrente alla precedente e prende la migliore. L'euristica è costituita di 6 steps:

1. Per ogni operazione, vengono individuati i successori secondo le precedenze tecnologiche;
2. Creare una lista-A per ogni operazione in riferimento alle operazioni che la precedono e da questa seleziona la prima e assegnala alla stazione;
3. Creare una lista-B composta da tutte le operazioni che non hanno predecessori. Se tutte le operazioni sono state assegnate alla stazione allora fermati;

4. Dalla lista-B crea una lista-C composta da tutte le operazioni i cui tempi non superano il tempo ciclo a disposizione della stazione. Se la lista C è vuota, prendi un'altra stazione e ripeti lo step 4;
5. Seleziona random una operazione dalla lista-C e assegbala alla stazione;
6. La lista-B deve riflettere il tempo rimasto libero della stazione. Se la lista-B è vuota, aggiorna la lista-A e torna allo step 3.

2.2 Metodi meta-euristici

Secondo Sanches, il maggiore problema dei metodi euristici è la possiblità di bloccarsi in regioni di ottimi locali, senza esplorare quelle che possiedono soluzioni efficienti, o ottime. I metodi meta-euristici sono stati sviluppati per risolvere questo problema. La logica è migliorare le procedure euristice per evitare stagnazioni in ottimi locali.

Le tecniche meta-euristiche sono suddivise in due categorie: la prima è la *local search*, che comincia da una soluzione iniziale ed esplora l'intorno delle soluzioni. Tabu Search e Simulated Annealing sono esempi di queste tecniche. La seconda categoria è il *population search*, che inizia con un set di soluzioni iniziali, detto popolazione iniziale. In questa classe di modelli, sono utilizzati operatori per generare nuovi e migliori individui per la popolazione. Ant Colony optimization e Genetic Algorithm appartengono a questa categoria.

Le attuali tecniche meta-euristiche tendono ad essere versatili perché sono metodi generali di risoluzione e possono essere applicate a differenti tipologie di problemi.

2.2.1 Tabu search

Il *Tabu Search* (TS) è un algoritmo per risolvere problemi di ottimizzazione prettamente discreti, proposto da Glover nel 1986. L'algoritmo parte da una soluzione iniziale x_0 ed esegue una serie di mosse che portano ad una nuova soluzione all'interno del "vicinato" (o insieme di adiacenza N(x)) della soluzione corrente x, nella quale la

funzione obiettivo f assume un valore minore della nuova soluzione. Al fine di migliorare l'efficienza del processo di esplorazione, si vogliono perseguire due obiettivi:

- Non rimanere ancorati ad un minimo locale della funzione obiettivo;
- Non esplorare regioni dello spazio (soluzioni x) già esaminate.

Per perseguire il primo obiettivo, ad ogni iterazione del Tabu Search non si assume come nuova soluzione un elemento $x_{t+1} = y \in N(x_t): f(y) < f(x_t)$, cosa che ci costringerebbe in un minimo locale; ma si assume $x_{t+1} = y \in N(x_t): f(y) < f(z), \forall z \in N(x_t)$.

In sostanza ad ogni iterazione si sceglie il "migliore" tra i "vicini", accettando anche soluzioni peggiorative rispetto alla soluzione attuale. Così facendo, si rischia di ricadere subito dopo nel minimo locale, a meno che non si impediscano in qualche modo le mosse recentemente eseguite. Il concetto fondamentale della Tabu Search consiste appunto nel rendere "proibite" (tabu), le ultime k mosse eseguite nel cammino di ricerca, in modo che l'algoritmo non possa tornare sui suoi passi e ricadere nel minimo locale. Inoltre, rendendo proibite le ultime mosse effettuate, si abbassa drasticamente la probabilità di esaminare soluzioni già trovate in precedenza, aumentando così il numero di soluzioni diverse esaminate.

2.2.2 Simulated Annealing

È una metodologia di ricerca altamente adatta per qualunque problema di ottimizzazione che fonda le sue basi nella statistica meccanica. È stata sviluppata originariamente da Kirkpatrick per risolvere problemi di ottimizzazione combinatoriale e discreta. Il SA è nato come metodo di simulazione della tempra (annealing) dei solidi. L'annealing è il processo con il quale un solido, portato allo stato fluido, mediante riscaldamento ad alte temperature, viene riportato poi di nuovo allo stato solido o cristallino, a temperature basse, controllando e riducendo gradualmente la temperatura. Ad alte temperature, gli atomi del sistema si trovano in uno stato

altamente disordinato e quindi l'energia del sistema è elevata. Per portare tali atomi in una configurazione cristallina altamente ordinata (statisticamente), deve essere abbassata la temperatura del sistema. Riduzioni veloci della temperatura possono causare difettosità nel reticolo cristallino con conseguente metastabilità, con fessurazioni e fratture del reticolo stesso (stress termico). L'annealing evita questo fenomeno procedendo ad un graduale raffreddamento del sistema, portandolo ad una struttura globalmente ottima stabile. Il sistema si dice essere in equilibrio termico alla temperatura T se la probabilità $P(E_i)$ di uno stato avente energia E_i è governata dalla distribuzione di Boltzmann:

$$P(E_i) = \frac{\exp\left(-\frac{E_i}{k_B T}\right)}{\sum_j \exp\left(-\frac{E_j}{k_B T}\right)} \qquad \text{(eq. 2)}$$

Dove k_B è la costante di Boltzmann.

L'algoritmo SA per un problema di ottimizzazione può essere riassunto nel seguente generico algoritmo:

1. Sia data una configurazione iniziale o soluzione x_0 con energia o valore della funzione obiettivo E_0. Selezionare un valore iniziale T_0 per la temperatura;
2. Per ogni stadio della temperatura effettuare i seguenti passi :
    A. Generare una configurazione candidata ammissibile tramite una piccola perturbazione casuale della configurazione corrente. Valutare la differenza di energia ΔE fra le due configurazioni;
    B. Se ΔE ≤ 0, la configurazione candidato ha un valore della funzione obiettivo inferiore rispetto a quello della configurazione corrente. Accettare la nuova soluzione e sostituirla a quella corrente. Se ΔE > 0, la configurazione candidato ha un valore della funzione obiettivo peggiore rispetto quello della configurazione corrente. Accettare tale soluzione con una probabilità P(E) e aggiornare la configurazione corrente se necessario;
    C. Se non è raggiunto l'equilibrio termico, torna a Step 2.A
    altrimenti vai a Step 3.

3. Se il processo annealing è incompleto, ridurre la temperatura e ritornare a 2.

La determinazione di un appropriato criterio di interruzione è un fattore critico per il successo del SA. Il processo di annealing converge ad un ottimo globale se la temperatura è sufficientemente ridotta in modo graduale. Al fine di evitare eccessivo tempo di calcolo, ciò che viene usualmente fatto è di interrompere il processo quando il numero di soluzioni accettate ad un certo stadio è inferiore ad un valore fissato. In tal caso ci si può aspettare una soluzione prossima all'ottimo globale. Eventualmente, per affinare o verificare la bontà della soluzione ottenuta, è possibile utilizzare dei metodi di ricerca locale.

2.2.3 Ant Colony Optimization

L'Ant Colony Optimization (ACO) è un algoritmo che formalizza il comportamento delle formiche ed è stato introdotto nel 1992 da Marco Dorigo del Politecnico di Milano. Si tratta di un algoritmo di ricerca del percorso ottimale tra un nodo di partenza (colonia di formiche) e un nodo obiettivo (cibo) basato sull'utilizzo di tecniche probabilistiche e sull'ottimizzazione metaeuristica.

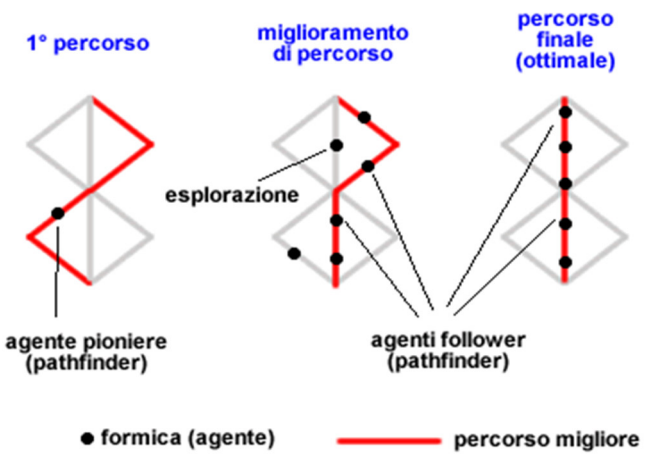

Figura 2.1 Ant Colony Optimization

In natura le formiche si muovono a caso alla ricerca del cibo. Quando lo trovano, tornano a ritroso fino alla colonia depositando sul terreno dei segnali chimici (feromoni) al fine di tracciare un sentiero tra cibo e colonia per le altre formiche (comportamento cooperativo), come illustrato in figura 2.1. Quando altre formiche incontrano il sentiero tracciato dai feromoni, smettono di muoversi a caso e iniziano a percorrerlo anche loro. Col passare del tempo

i feromoni evaporano e le formiche sono portate a compiere piccole variazioni di percorso. Il segnale dei feromoni tende a mantenersi più a lungo nei percorsi più brevi poiché sono anche quelli più frequentati dalle formiche. In conclusione, pur essendo ogni formica dotata di poca memoria e di scarsa intelligenza, tutte insieme riescono a risolvere un problema complesso senza bisogno di un'organizzazione centralizzata. Elementi fondamentali dell'algoritmo:

- Scopo "ideale": trovare il massimo della funzione f : S → R;
- Gli elementi di S sono assimilabili a percorsi in un grafo (N, A);
- N è l'insieme degli stati;
- A è l'insieme delle transizioni ammissibili (opzionale) crs è il costo della transizione r → s;
- Ω è l'insieme dei vincoli del problema (restrizioni sui possibili percorsi).

L'algoritmo ACO è un algoritmo parallelo cooperativo e si distingue dagli algoritmi paralleli competitivi. In entrambi i casi gli agenti lavorano parallelamente e in modo autonomo alla soluzione dello stesso problema. Tuttavia, nel caso degli algoritmi competitivi ciò avviene senza alcuna cooperazione reciproca tra gli agenti. Al termine dell'elaborazione degli algoritmi competitivi viene selezionata soltanto la migliore soluzione finale. Tutte le altre soluzioni inferiori, elaborate dagli altri agenti, sono cestinate. Nel caso degli algoritmi ACO, invece, gli agenti condividono dinamicamente i risultati parziali nel corso dell'elaborazione. Pur restando autonomi, gli agenti ACO comunicano (output) agli altri agenti l'eventuale scoperta di un miglioramento di cammino (efficienza) e sono pronti ad accettare (input) le informazioni e i miglioramenti di cammino scoperti dagli altri agenti anche se non sono stati loro a trovarli. In conclusione, la cooperazione tra gli agenti consente di ottimizzare l'impiego delle risorse e di aumentare la velocità computazionale.

2.2.4 Genetic Algorithm

Gli *algoritmi genetici* si basano sulla teoria evoluzionistica di Darwin.

La teoria evoluzionistica dei sistemi biologici lega la possibilità di sopravvivenza di un individuo principalmente alla sua capacità di adattamento (fitness) all'ambiente. I mezzi che, in ciascun individuo, caratterizzano tale capacità risiedono nel suo patrimonio genetico, cioè in un insieme di informazioni ereditate anzitutto da padre e madre e successivamente assoggettate parzialmente a un processo di cambiamento casuale per far sì che ognuno abbia una identità propria, distinta da quella dei genitori. Gli individui più deboli, meno idonei a far fronte all'ambiente, muoiono, in genere, prima degli altri e, perciò, si riproducono di meno; quelli più forti sopravvivono generalmente più a lungo e si riproducono maggiormente. L'effetto di questo processo è una più diffusa trasmissione delle caratteristiche migliori che, su tempi lunghi, porta automaticamente all'evoluzione della specie e all'esistenza di generazioni in possesso di capacità di adattamento all'ambiente sempre maggiori.

Gli AG risolvono un determinato problema ricorrendo a una popolazione di soluzioni che, inizialmente casuali e quindi con fitness bassa, vengono poi fatte evolvere per un certo numero di generazioni successive, sino all'apparizione di almeno una soluzione con fitness elevata. Per poter applicare l'algoritmo genetico, occorre anzitutto codificare numericamente le soluzioni e individuare una opportuna funzione di fitness. Ogni individuo della popolazione è codificato da un cromosoma, una stringa di lunghezza costante formata da geni. Generalmente i geni sono binari, con valori 0 o 1. Se abbiamo N geni a disposizione si possono realizzare 2^N stringhe diverse.

La funzione di fitness è la misura numerica della "bontà" di una soluzione, generalmente normalizzata tra 0 e 1, e fortemente dipendente dal problema che occorre risolvere. La popolazione che evolve ha un numero M di individui (M << 2N), che viene mantenuto costante da una generazione all'altra. Ad ogni generazione, vengono eseguite opportune operazioni genetiche che producono nuovi individui. Per mantenere costante M, occorre copiare (riprodurre) nella generazione successiva solo M individui ed essi vengono selezionati con criteri probabilistici, premiando tendenzialmente quelli dotati di maggiore fitness. Le operazioni genetiche più importanti sono il cross-over (operatore genetico che prevede la generazione di una

nuova soluzione che si ottiene con la ricombinazione di soluzioni già esistenti) e la mutazione (operatore genetico che prevede l"alterazione casuale di una soluzione), di cui esistono differenti versioni in letteratura.

L'algoritmo genetico di base, di cui esistono molte varianti, è il seguente:

1. Inizializzazione casuale di una popolazione di N cromosomi e calcolo delle fitness;
2. Tramite operatore di selezione, vengono selezionati a coppie gli individui migliori;
3. Le coppie selezionate, vengono sottoposte a cross-over e da questa attività nascono altri due cromosomi, detti figli;
4. Viene utilizzato l'operatore di mutazione, necessario per ricombinare i cromosomi;
5. Calcolo della funzione di fitness e scelta degli N individui migliori;
6. Ritorno al punto 3 fino al soddisfacimento di un criterio di stop che può essere il raggiungimento di numero di iterazioni o di un valore della funzione di fitness.

2.2.5 Genetic Algorithm "Genial"

L'algoritmo Genial è un particolare esempio di algoritmo genetico per la risoluzione di problemi di bilanciamento multi-obiettivo per una linea di assemblaggio manuale, implementato in ambiente Matlab.

Gli obiettivi che vuole ottimizzare l'algoritmo sono cinque:

- Minimizzazione del numero di stazioni;
- Minimizzazione delle abilità degli operatori tra le stazioni. Ogni operazione può richiedere un livello di abilità tra zero (nessuna abilità), uno (bassa abilità) e due (elevata abilità). Tale obiettivo ha lo scopo di ottimizzare l'allocazione delle operazioni alle stazioni in modo da raggruppare nella stessa stazione quelle che richiedono lo stesso livello di abilità;

- Minimizzazione degli equipment fra le stazioni. Ogni operazione può essere compiuta con un massimo di tre differenti equipments, scelti tra 10 diversi strumenti. Ottimizzare tale funzione vuol dire allocare le operazioni sulla linea in modo che siano minimizzati il numero di equipment richiesti ovvero raggruppare le operazioni che richiedono lo stesso equipment nella medesima stazione;
- Minimizzazione dei cambi di orientamento dei componenti in una certa sequenza. Ogni operazione viene svolta con un determinato orientamento nello spazio che può essere $\pm x$, $\pm y$ e $\pm z$. Ottimizzare questa funzione si traduce nel collocare le operazioni sulle stazioni in modo da minimizzare i cambi di orientamento tra un'operazione e l'altra cercando di raggruppare in una medesima stazione le operazioni con lo stesso orientamento;
- Minimizzazione della Workload Variance. Ogni stazione ha a disposizione un determinato tempo ciclo per svolgere le operazioni che gli vengono assegnate. Ottimizzare tale obiettivo si traduce nell'allocare le operazioni alle stazioni in modo da raggiungere tale valore per tutte le stazioni.

Le caratteristiche dell'algoritmo Genial sono:

1. Il cromosoma è costruito secondo la task-oriented rapresentation, ovvero la sequenza dei suoi geni riflette l'assegnamento delle operazioni sulle singole stazioni. Ad esempio, come illustrato nelle seguente figura, le operazioni all'interno del cromosoma raffigurano il loro posizionamento così come apparirà nelle stazioni.

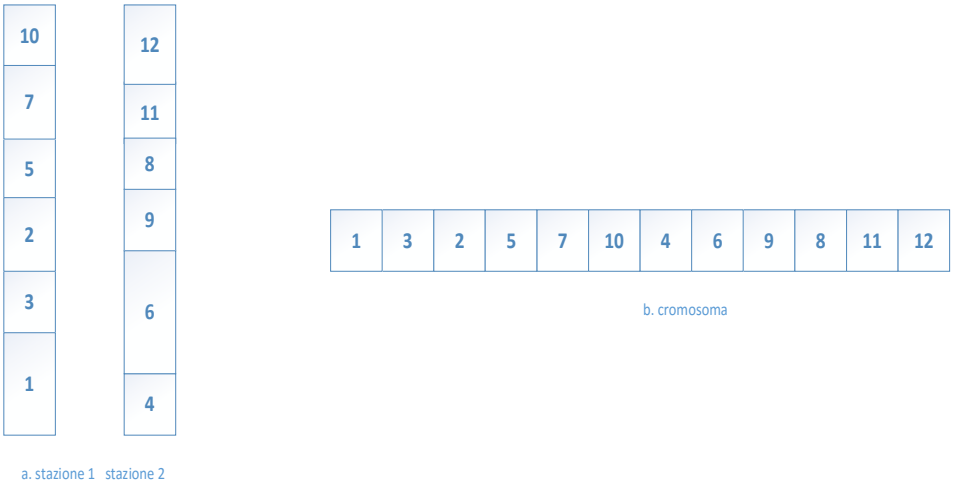

Figura 2.2 Task-oriented representation

2. Per assicurare che il cromosoma sia fattibile ovvero rispetti le precedenza tecnologiche, si utilizza la strategia del *sequenze planner* che consta di 5 step:
   - Creazione di una lista costituita dalle operazioni prive di prcedenze tecnologiche;
   - Se la lista è vuota allora la procedura si ferma, altrimenti prosegue allo step successivo;
   - Si sceglie random una tra queste operazioni e si inserisce nella prima posizione disponibile del cromosoma e si elimina dalla lista;
   - Si inseriscono nella lista le operazioni che seguono l'operazione appena eliminata, se non vincolate da altre operazioni;
   - Si riparte dallo step 2.
3. La bontà di una soluzione ovvero la sua funzione di fitness viene calcolata come somma pesata dei cinque obiettivi:

$$F = \eta_1 F_1 + \eta_2 F_2 + \eta_3 F_3 + \eta_4 F_4 + \eta_5 F_5 \qquad \text{(eq. 3)}$$

   In cui

   F_i = sono i valori normalizzati delle funzioni obiettivo;

   η_i = pesi, $\sum_i \eta_i = 1$;

   $F_i = 1 + \frac{(LB-x)}{(UB-LB)}$

   in cui x rappresenta la variabile da normalizzare, LB e UB sono rispettivamente il limite inferiore e superiore di ciascun obiettivo.

4. La selezione utilizzata è la *Roulette wheel selection*, tecnica con la quale la probabilità che un cromosoma ha di essere selezionato è proporzionale alla sua fitness. Maggiore è la fitness del cromosoma maggiore sarà la sua probabilità di essere selezionato. Nella figura 2.3, la freccia ruota tante volte quanti sono gli individui da generare per la successiva popolazione, la freccia si ferma sugli individui migliori, ovvero coloro che occupano una maggiore porzione di ruota;

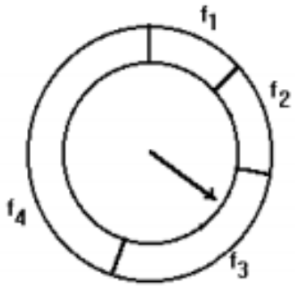

Figura 2.3 Roulette wheel selection

5. L'operatore di crossover, *Uniform order based crossover,* prende in considerazione due soluzioni ammissibili e taglia i vettori di codifica in due punti predefiniti o casuali al fine di ottenere una testa, un corpo e una coda da entrambe le soluzioni. Come indicato in figura 2.4, la prima nuova soluzione sarà data dalla testa e della coda della prima soluzione e dalla parte centrale formata dai geni mancanti così come appaiono nella seconda soluzione. La seconda nuova soluzione sarà data dalla testa e dalla coda della seconda soluzione e dalla parte centrale dai geni mancanti così come appaiono nella prima soluzione. Tale operatore crea automaticamente soluzioni fattibili;

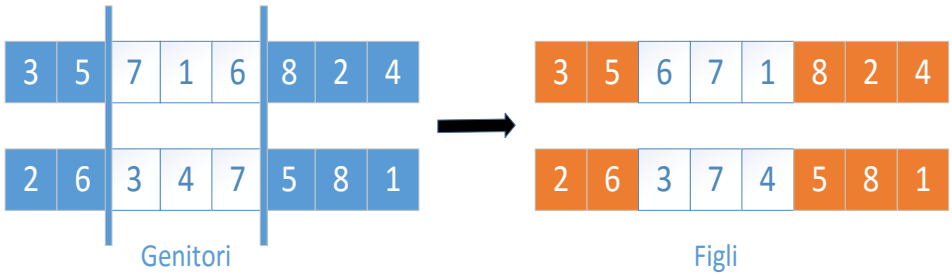

Figura 2.4 Uniform order based crossover

6. La mutazione denominata *swap mutation* è una semplice ricombinazione casuale dei geni di uno stesso cromosoma. Si estraggono random due numeri, e alle corrispondenti posizioni all'interno del cromosoma i geni vengono invertiti fra di loro;

7. Riparazione: tra gli individui creati tramite la mutazione, si possono individuare soluzioni infattibili che necessitano di riparazione per essere reintrodotti nella popolazione;
8. Selezione e procedura finale: dalla popolazione ottenuta dalle precedenti operazioni, vengono selezionati i cromosomi con i più alti valori di fitness, i quali andranno a comporre la nuova generazione. Ad ogni iterazione dell‟algoritmo la numerosità della popolazione rimane invariata. Per evitare la possibilità di perdere i migliori cromosomi come risultato della selezione della probabilità, le soluzioni ottenute nella presente generazione vengono confrontate con quelle della generazione precedente.
9. Criterio di stop: l'algoritmo si arresta dopo aver raggiunto un determinato numero di iterazioni.

2.3 Scopi e articolazione della tesi

Lo scopo di questa tesi è validare e migliorare l'algoritmo genetico Genial, utilizzato per la risoluzione di problemi di bilanciamento per linee di montaggio manuali.

Nel terzo capitolo, viene illustrata la metodologia scelta e utilizzata per testare, validare e migliorare l'algoritmo genetico. La validazione viene suddivisa in due categorie, la prima interessa la validazione dell'algoritmo genetico per la risoluzione di problemi mono-obiettivo a partire dal SALBP-1, la seconda riguarda la validazione per problemi multi-obiettivo. Per valutare la bontà delle soluzioni venogno introdotti alcuni indici di perfomrance. Gli aspetti considerati critici e problematici vengono risolti nell'ultimo capitolo.

Nel quarto capitolo, si illustrano le metodologie proposte per risolvere le problematiche inerenti l'algoritmo. Tali soluzioni vengono trascritte in linguaggio di programmazione e riportate nei file .m su Matlab. Viene fatto girare l'algoritmo in modo da confrontare le soluzioni con quelle ottenute precedentemente e infine si compiono le considerazioni a riguardo.

Capitolo 3

3. Validazione dell'algoritmo genetico GenIAL

In questo capitolo viene affrontata la validazione dell'algoritmo genetico GenIAL.

Tale algoritmo viene utilizzato per il bilanciamento delle linee di assemblaggio manuali allo scopo di ottimizzare cinque obiettivi precedentemente delineati.

La validazione viene suddivisa in due sottocategorie:

1. Comprendere la bontà dell'algoritmo nel raggiungere soluzioni ottimali per problemi mono-obiettivo. Attualmente, GenIAL valuta le soluzioni in bae al valore della loro funzione di fitness calcolata come indicato nell'equazione 4:

   $$F = \eta_1 F_1 + \eta_2 F_2 + \eta_3 F_3 + \eta_4 F_4 + \eta_5 F_5 \qquad \text{(eq. 4)}$$

   in cui

   η_i = peso assegnato all'obiettivo i-esimo;

   F_i = funzione normalizzata tra 0 e 1 dell'obiettivo i-esimo.

   Per rendere mono-obiettivo lo strumento, viene posto pari a uno il peso dell'obiettivo che si vuole raggiungere e pari a zero tutti gli altri.
2. Comprendere la bontà dell'algoritmo di trovare soluzioni a problemi multi-obiettivo ponendo i pesi degli obiettivi come di seguito proposto:

   η_1 = 0.2;

   η_2 = 0.1;

   η_3 = 0.2;

   η_4 = 0.2;

η_5 = 0.3.

3.1 Metodo di validazione

In letteratura, per comparare le metodologie di risoluzione per problemi SALBP-1, SALBP-2 e SALBP-F, viene utilizzato un insieme di dati elaborato da Scholl. Si tratta di una combinazione ed evoluzione dei dati sviluppati da Talbot e da Hoffmann. I risultati di tali problemi sono in continua evoluzione poiché, nel tempo, nascono algoritmi sempre più sofisticati per la loro risoluzione.

Il set di dati contiene 269 istanze, generate e realizzate tramite studi empirici, basati su 25 grafi. Ogni grafo, a sua volta, può avere fino ad un massimo di 27 versioni, che si distinguono le une dalle altre per il differente valore del tempo ciclo.

Nella tabella 3.1, vengono specificati i grafi utilizzati per le differenti versioni di SALBP. Buona parte di essi è basata su problemi realizzati "in laboratorio" e non rappresentano nessuna realtà effettiva, altri derivano da reali casi industriali, e altri ancora da modifiche e ricombinazione dei casi reali.

Nome del grafo	Riferimento	N° di operazioni	t_{min} [s]	t_{max} [s]	t_{sum} [s]	OS
Arcus1	Arcus	83	233	3691	75707	59.09
Arcus2	Arcus	111	10	5689	150399	40.38
Barthold	Barthold	148	3	383	5634	25.80
Barthold2	Barthold,modificato	148	1	83	4234	25.80
Bowman	Bowman	8	3	17	75	75.00
Buxey	Buxey	29	1	25	324	50.74
Gunther	Gunther	35	1	40	483	59.50
Hahn	Hahn	53	40	1775	14026	83.82
Heskiaoff	Heskiaoff	28	1	108	1024	22.49
Jackson	Jackson	11	1	7	46	58.18
Jaeschke	Jaeschke	9	1	6	37	83.33
Kilbridge	Kilbridge	45	3	55	552	44.55
Lutz1	Lutz	32	100	1400	14140	83.47
Lutz2	Lutz2	89	1	10	485	77.55
Lutz3	Lutz2,modificato3	89	1	74	1644	77.55
Mansoor	Mansoor	11	2	45	185	60.00
Mertens	Mertens	7	1	6	29	52.38
Mitchell	Mitchell	21	1	13	105	70.95

Mukherje	Mukherje	94	8	171	4208	44.80
Roszieg	Roszieg	25	1	13	125	71.67
Sawyer	Sawyer	30	1	25	324	44.83
Scholl	Scholl	297	5	1386	69655	58.16
Tonge	Tonge	70	1	156	3510	59.42
Warnecke	Warnecke	58	7	53	1548	59.10
Wee-Mag	Wee-Mag	75	2	27	1499	22.67

Tabella 3.1 Casi Scholl

In tabella, sono indicate le referenze originali, il numero di operazioni, il minimo e il massimo task time necessari per calcolare il tempo ciclo e il numero minimo di stazioni, come verrà spiegato di seguito, e il tempo di produzione. Un ulteriore classificazione dei grafi si basa su un indice che ne definisce la complessità, l'order strength (OS), calcolato come il rapporto tra il numero di archi, presenti nel grafo, e che formano cicli e $n * \frac{(n-1)}{2}$, ovvero il massimo numero di archi che può contenere un grafo aciclico[2] con n nodi. Tale indice indica il numero relativo di relazioni di precedenza del grafo considerato. Problemi con un order strength elevato saranno più complessi di problemi con indice inferiore.

Di seguito vengono proposte alcune istanze di Scholl utilizzate per i problemi di tipo SALBP-1, il cui obiettivo è minimizzare il numero di stazioni necessarie a processare tutte le operazioni. I tempi ciclo sono determinati secondo il metodo di Hoffman:

$$m_{max} = \left\lfloor \frac{t_{sum}}{t_{max}} \right\rfloor$$

$$c(m) = \left\lceil \frac{t_{sum}}{m} \right\rceil \quad \text{per m}=\left\lfloor \frac{m_{max}}{2} \right\rfloor, \dots, m_{max}$$

Dove:

m_{max} = numero massimo di stazioni che può avere la linea di montaggio;

$c(m)$ = tempo ciclo ottenuto con un numero m di stazioni;

$t_{sum} = \sum_{i=1}^{n} tek_i$, n è il numero delle operazioni;

t_{max} = operazione con la durata maggiore.

[2] È un grafo senza cicli, ovvero senza percorsi chiusi in cui il nodo di partenza coincide con il nodo di arrivo.

Scholl, per calcolare le soluzioni ottime dei problemi SALBP-1 ha utilizzato l'algoritmo SALOME-1, un metodo branch and bound bidirezionale[3]. Ci sono problemi in cui la soluzione attualmente nota differisce da quella ottima, data dal rapporto del tempo di produzione e il tempo ciclo, in quel caso la differenza è specificata dalla presenza di segni "+". Per validare l'algoritmo GenIAL, si è considerato necessario e opportuno scegliere i 6 casi di studio che rappresentano prodotti esistenti tratti dalla realtà industriale.

Nella tabella 3.2, sono indicati i 6 casi di studio utilizzati per la validazione dell'algoritmo GenIAL. Per ogni caso sono stabilite le istanze date da diversi tempi ciclo (c) e per ognuno di essi è specificato il numero di stazioni (m*) ottenuto da SALOME-1.

Grafo	c	m*	c	m*	c	m*	c	m*	c	m*
Barthold	403	14	434	13	470	12	513	11	564	10
	626	9	705	8	805	7				
Gunther	41	14++	44	12+	49	11+	54	9	61	9+
	69	8+	81	7+						
Hahn	2004	8	2338	7	2806	6	3507	5	4676	4
Lutz1	1414	11+	1572	10+	1768	9+	2020	8+	2357	7+
	2828	6+								
Tonge	160	23+	168	22+	176	21+	185	20+	195	19+
	207	18+	220	17+	234	16+	251	14	270	14+
	293	13+	320	11						
Warnecke	54	31++	56	29+	58	29++	60	27+	62	27++
	65	25+	68	24+	71	23+	74	22+	78	21+
	82	20+	86	19+	92	17	97	17+	104	15
	111	14								

Tabella 3.2 Soluzioni ottime casi Scholl

[3] Il branch e bound è una tecnica generale per la risoluzione di rpoblemi di ottimizzazione combinatoria che si basa sulla scomposizione del problema originale in sottoproblemi più semplici da risolvere. Il SALOME-1 viene detto bidirezionale perché l'attività di branch si usa un grafo con archi bidirezionati.

3.1.1 Caso Barthold

Il caso Barthold interessa l'assemblaggio di una piccola utilitaria. Le operazioni sono 148, e i task time oscillano da un minimo di 3 [s] ad un massimo di 383 [s]. L'order strength di tale problema è 25.80.

Esistono 8 istanze del caso Barthold date dai differenti valori del tempo ciclo. Nel grafo delle precedenze, illustrato in figura 3.1, il numero all'interno dei nodi indica l'operazione, mentre il valore sui nodi indica il task time, in secondi.

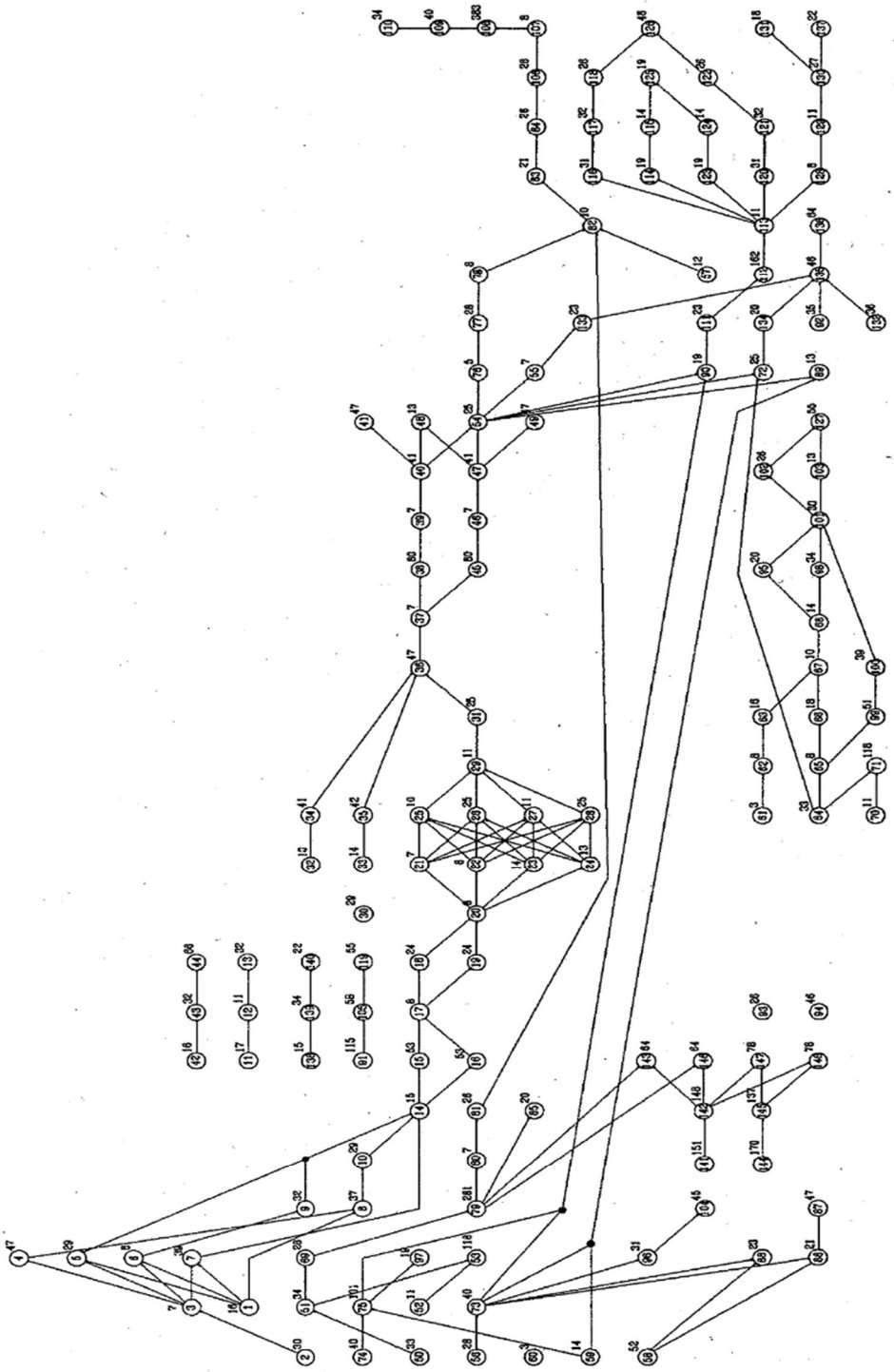

Figura 3.1 Grafo caso Barthold

3.1.2 Caso Gunther

Il caso Gunther descrive l'assemblaggio della "culla" per l'alloggiamento del motore nelle autovetture, in figura 3.2. Le operazioni sono 35, e i task time oscillano da un minimo di 1 [s] ad un massimo di 40 [s]. L'order strength di tale problema è 59.05.

Figura 3.2 Engine auto cradle

Esistono 7 istanze del caso Gunther, il cui grafo è mostrato nella figura 3.3.

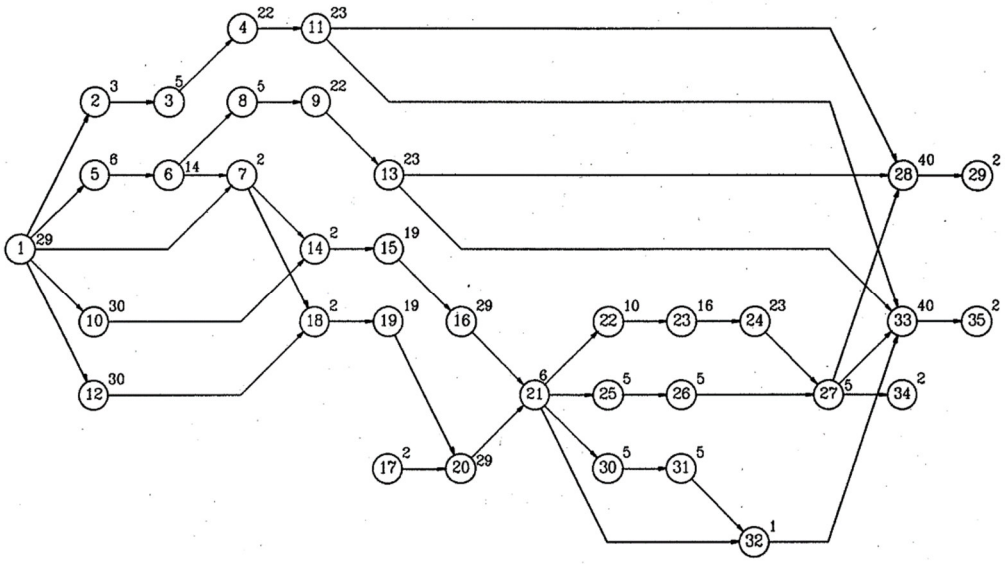

Figura 3.3 Grafo caso Gunther

3.1.3 Caso Hahn

Il caso Hahn riguarda il montaggio di un serbatoio di acqua calda, illustrato di seguito in figura 3.4. Le operazioni sono 53, e i task time oscillano da un minimo di 40 [s] ad un massimo di 1775 [s]. L'order strength di tale problema è 83.40.

Figura 3.4 Hot water tank

Esistono 5 istanze del caso Hahn, di cui si mostra il grafo nella figura sottostante.

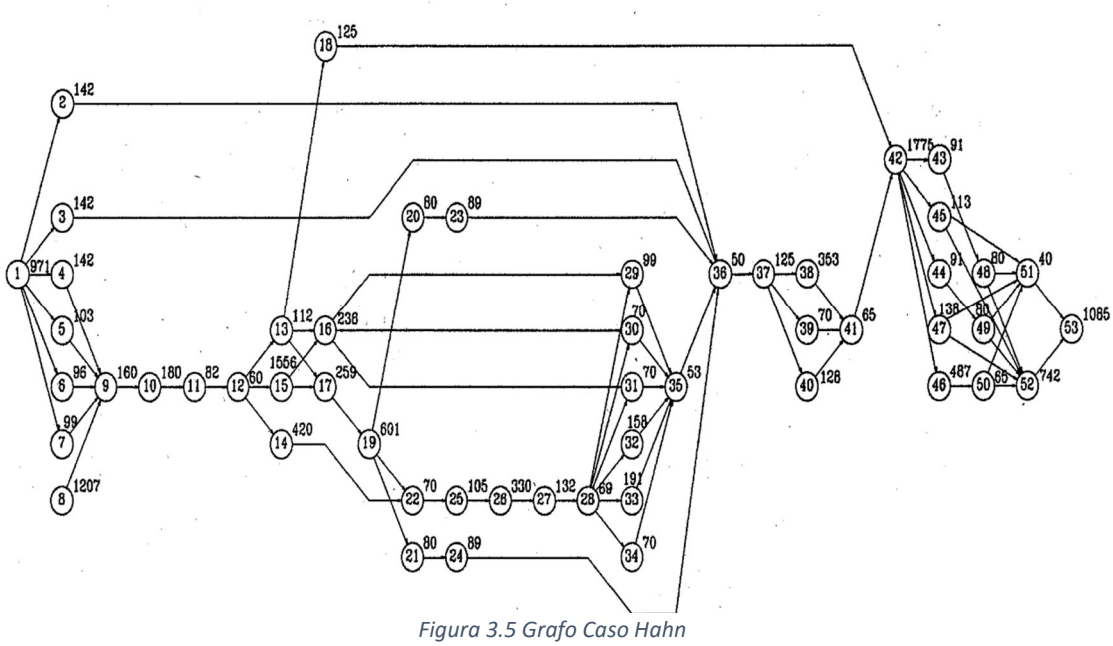

Figura 3.5 Grafo Caso Hahn

3.1.4 Caso Lutz1

Il caso Lutz1 tratta il montaggio di un dispositivo per la riduzione della pressione, di cui segue l'immagine in figura 3.6.

Le operazioni sono 32, e i task time oscillano da un minimo di 100 [s] ad un massimo di 1400 [s]. L'order strength di tale problema è 83.50.

Figura 3.6 Pressure reducing device

Esistono 6 istanze del caso Lutz1, il cui grafo è rappresentato di seguito in figura 3.7.

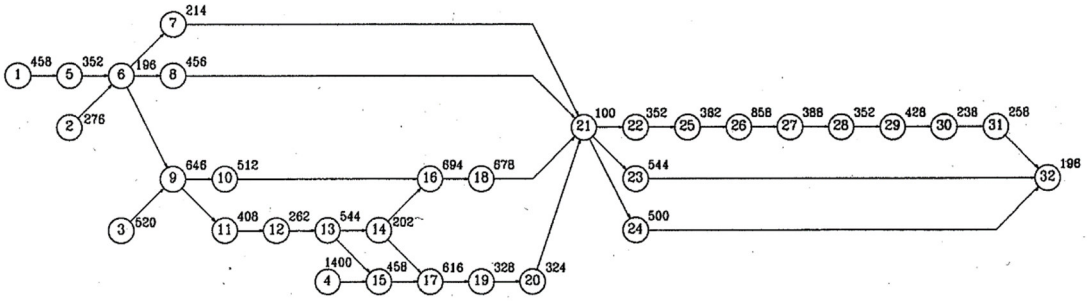

Figura 3.7 Grafo caso Lutz1

3.1.5 Caso Tonge

Il caso Tonge, il cui grafo è rappresentato in figura 3.8, interessa l'assemblaggio di un dispositivo dell'industria elettrica di cui non è specificata la tipologia. Le operazioni sono 70, e i task time oscillano da un minimo di 156 [s] ad un massimo di 3510 [s]. L'order strength di tale problema è 59.40. Esistono 16 istanze del caso Tonge date dai differenti valori del tempo ciclo.

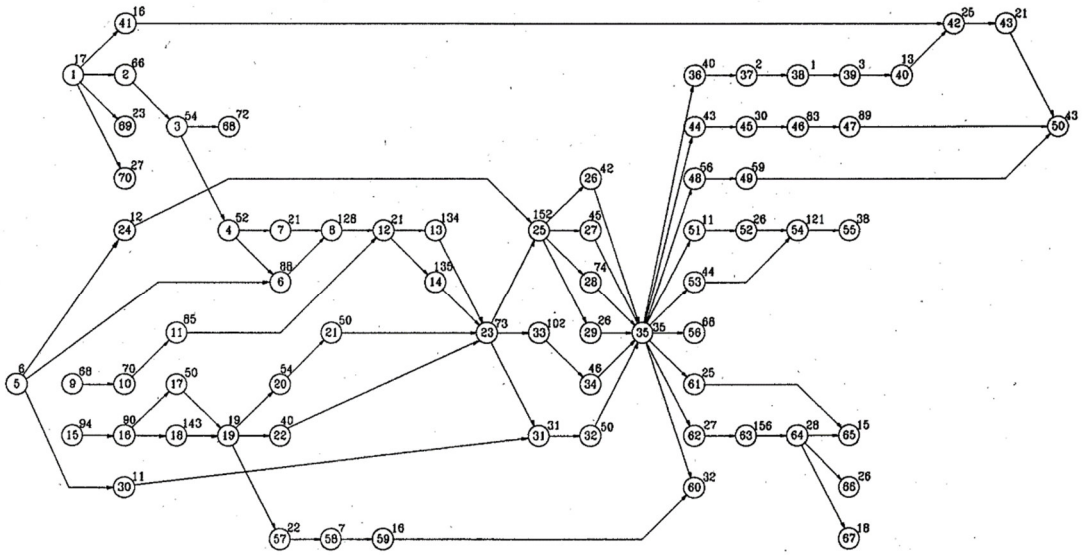

Figura 3.8 Grafo caso Tonge

3.1.6 Caso Warnecke

Il caso Warnecke interessa l'assemblaggio di una microcamera, come illustrata in figura 3.9.

Le operazioni sono 58, e i task time oscillano da un minimo di 7 [s] ad un massimo di 53 [s]. L'order strength di tale problema è 59.10. Esistono 16 istanze del caso Warnecke date dai differenti valori del tempo ciclo ed il grafo del caso è illustrato in figura 3.10.

Figura 3.9 Miniature camera

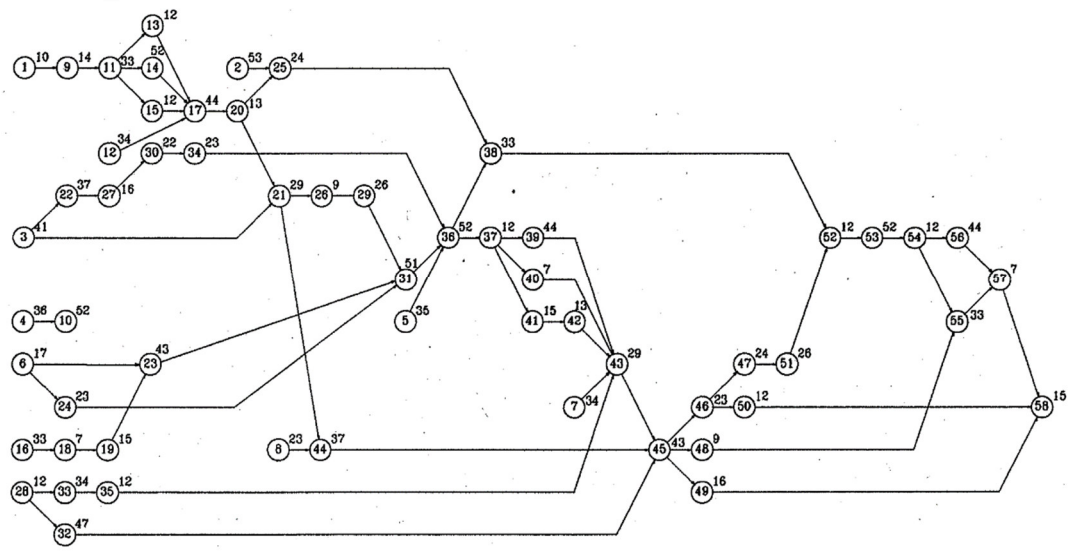

Figura 3.10 Grafo caso Warnecke

3.2 Misurazione delle performance

Prima di compiere la validazione dell'algoritmo genetico GenIAL, è indispensabile definire gli indici di performance sui cui verrà valutata la bontà delle soluzioni individuate.

3.2.1 Indici di performance per problemi mono-obiettivo

L'algoritmo genetico GenIAL è utilizzato per ottimizzare cinque obiettivi contemporaneamente. Il primo passo per la validazione è testare la sua capacità di trovare buone soluzioni ottimizzando un obiettivo alla volta.

Il primo obiettivo è quello di minimizzare il numero di stazioni necessarie a processare il prodotto all'interno della linea di assemblaggio, dato un tempo ciclo; si tratta di problemi del primo tipo SALBP-1. Gli indici su sui si valuterà l'algoritmo sono:

- Numero di stazioni trovate dall'algoritmo comparato con il numero di stazioni ottenuto dall'algoritmo SALOME-1;
- Valore della funzione di fitenss per la soluzione trovata.

Gli altri quattro obiettivi riguardano la minimizzazione delle skill richieste tra le stazioni, la minimizzazione dei cambi di direzione, la minimizzaione degli equipment richiesti e la minimizzazione della workload variance.

Per testare l'algoritmo su tali obiettivi, non si hanno soluzioni ottime a cui riferirsi in letteratura a differenza del numero di stazioni. Per ognuno di essi, l'unica valutazione che può essere compiuta riguarda la comparazione della soluzione trovata con il valore migliore, ovvero il lower bound, che si può ottenere nel caso di studio testato.

L'indice utilizzato per analizzare le performance è il medesimo:

- Numero di skill richieste trovato dall'algoritmo confrontato con il valore del lower bound;

- Numero di equipment richiesti e trovato dall'algoritmo confrontato con il valore del lower bound;
- Numero di cambi di direzioni trovato dall'algoritmo confrontato con il valore del lower bound;
- Valore della workload variance trovato dall'algoritmo e confrontato con il valore del lower bound.

3.2.2 Indici di perfomance per problemi multi-obiettivo

Nell'utilizzo dello strumento per la risoluzione di problemi multi-obiettivo, l'algoritmo ottimizza simultaneamente i cinque obiettivi sopra descritti.

Si ricorda che ogni soluzione, rappresentata dal singolo cromosoma, viene valutata in base al valore della sua funzione di fitness, calcolata come somma pesata dei valori normalizzati delle cinque funzioni obiettivo. È interessante analizzare quanto l'algoritmo si discosta dalle soluzioni trovate nel caso dei problemi mono-obiettivo. Infatti l'indice di perfomance utilizzato in questo caso è il seguente:

- Valori dei singoli obiettivi della soluzione ottima confrontati con quelli trovati testando singolarmente ogni singolo obiettivo.

3.3 Analisi dei risultati

Per far girare l'algoritmo è necessario dare in ingresso all'algoritmo genetico i dati in input che riguardano i casi di studio Scholl scelti per compiere la validazione, sono: il numero di operazioni, il tempo di processamento di ogni operazione, i vincoli di precedenza e il rateo produttivo.

3.3.1 Problemi mono-obiettivo

Per testare l'algoritmo genetico GenIAL su un unico obiettivo, è indispensabile settare il peso dell'obiettivo che interessa pari a uno mentre gli altri pesi devono essere posti pari a zero.

Si ricorda che l'algoritmo valuta i cromosomi in funzione della loro funzione di fintess calcolata come somma pesata di tutti e cinque gli obiettivi definiti precedentemente e indicata nell' equazione 1.

Il primo passo da compiere per testare l'algoritmo su problemi SALBP-1 è quindi, settare il peso η_1 pari a uno e tutti gli altri pari a zero.

Si ricorda che i problemi SALBP-1 consistono nell'assegnazione delle operazioni alle stazioni con l'unico obiettivo di minimizzare il **numero di stazioni di lavoro**, dato un tempo ciclo prefissato.

Nel grafico che segue, si illustrano i risultati ottenuti dall'algoritmo genetico, nelle otto istanze del **caso Barthold**, confrontati con il valore indicato da Scholl e con il valore del lower bound.

In figura 3.11, il valore indicato accanto al nome del caso fa riferimento al tempo ciclo di quella istanza valutato in secondi.

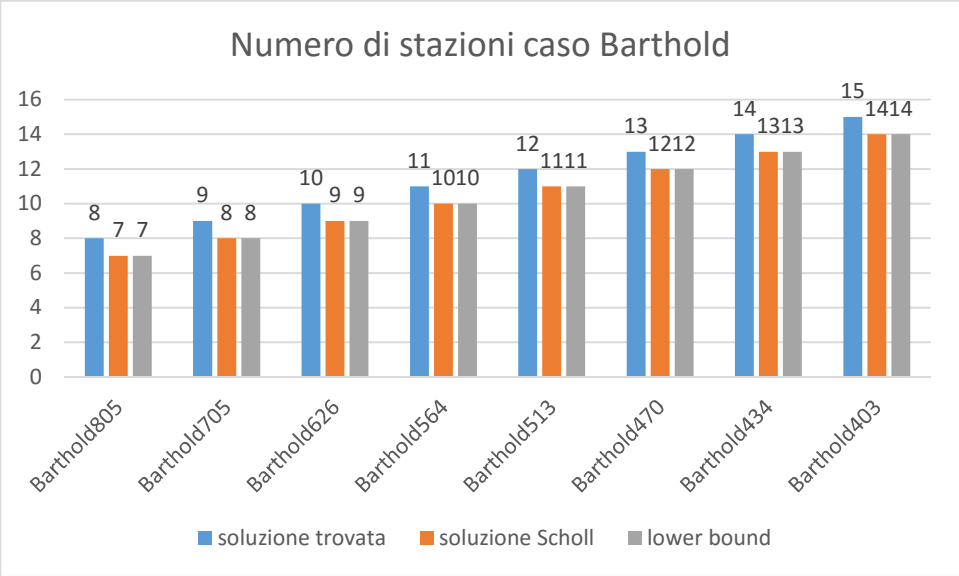

Figura 3.11 Numero di stazioni caso Barthold

Si deduce che l'algoritmo GenIAL è in grado di raggiungere soluzioni mediamente inferiori di una unità rispetto a quelle ottime, sia del lower bound sia di Scholl.

Il valore della funzione di fitness rimane costante e pari a 0.67.

Per comprendere perché ciò avvenga, bisogna ricordare come viene calcolato il valore normalizzato dei singoli obiettivi:

$$F_i = 1 + \frac{(LB-x)}{(UB-LB)} \qquad \text{(eq. 5)}$$

in cui

LB = caso migliore che si possa presentare;

x = valore trovato dall'algoritmo;

UB = caso peggiore che è possibile ottenere.

Il numeratore di tale funzione è pari alla differenza tra il LB e x che nel caso Bathold rimane costante a meno uno (-1). Il denominatore invece quantifica la differenza tra l'upper bound e il lower bound.

Si ricorda che l'upper bound concernente la minimizzazione del numero di stazioni, nell'algoritmo GenIAL, è funzione del lower bound, infatti viene calcolato come percentuale di quest'ultimo, nel dettaglio si hanno tre diversi casi:

- UB_N= ⌈LB_N+0,3*LB_N⌉ se LB_N è minore o uguale alle dieci unità;
- UB_N= ⌈LB_N+0,2*LB_N⌉ se LB_N è compreso tra le 10 e le 30 unità;
- UB_N= ⌈LB_N+0,1*LB_N⌉ in tutti gli altri casi.

Nel dettaglio del caso Barthold, il lower bound assume rispettivamente 10,11,12,13,14,15,16,17 numero di stazioni, che si discostano sempre tre unità dal lower bound, quindi il denominatore assume sempre il valore 3. Da ciò deriva che $F_1 = 1 + \frac{-1}{3} = 0.67$.

Di seguito, in figura 3.12, si mostrano i risultati per il **caso Gunther**, che ha un order strength di 59.05.

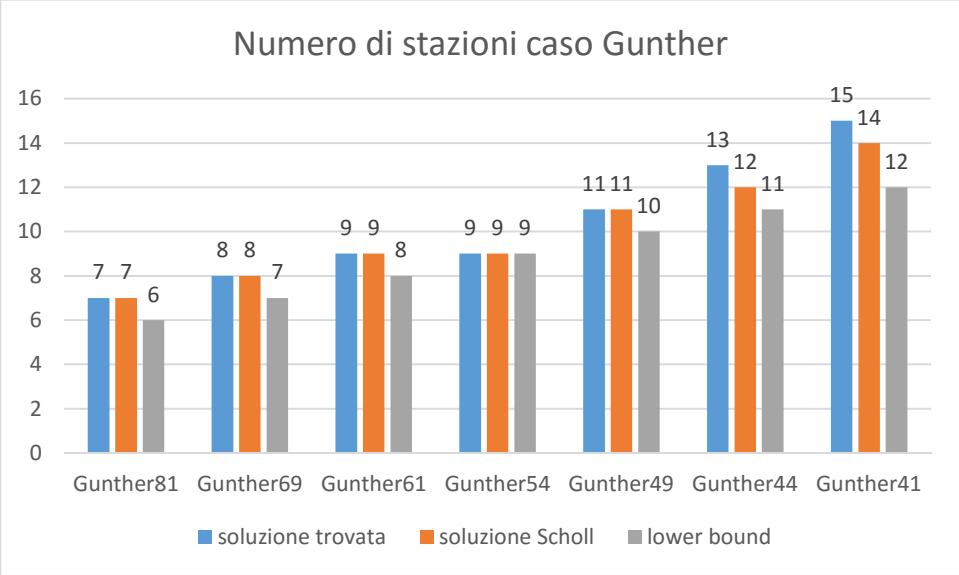

Figura 3.12 Numero di stazioni caso Gunther

In quattro istanze,il numero di stazioni trovate coincide con le soluzioni di Scholl quindi si può affermare che l'algoritmo GenIAL ha una convergenza all'ottimo nonostante si discosti di una unità dal caso migliore definito dal lower bound. Si afferma questo perché, come accennato nel capitolo precedente, i dati Scholl si riferiscono alle soluzioni che è possibile trovare attualmente con gli algoritmi a disposizione dello stato dell'arte.

Il valore della fitness nel caso Gunther è illustrato nella figura 3.13.

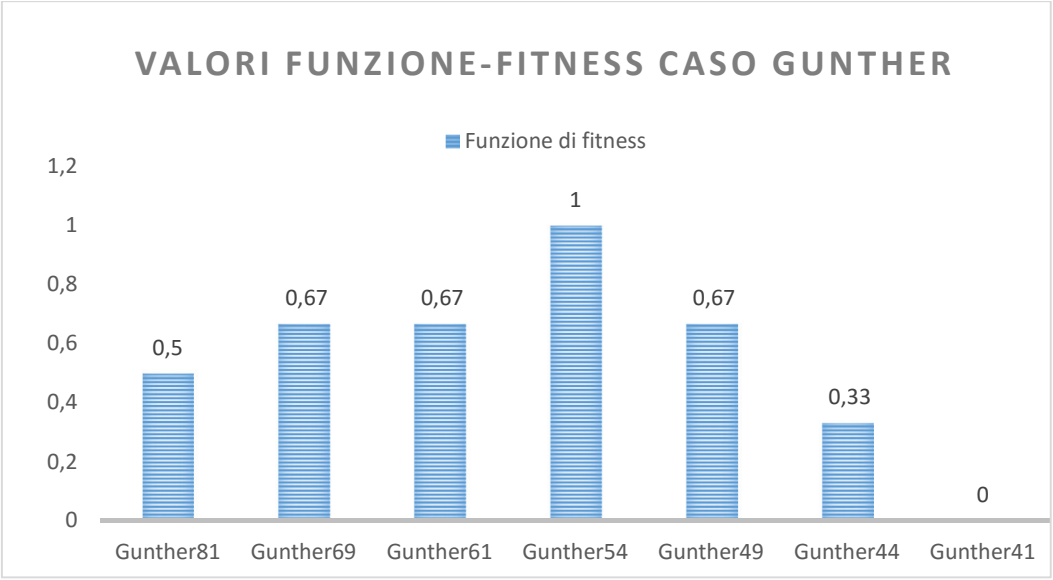

Figura 3.13 Valore funzione-fitness caso Gunther

Nel caso Gunther con tempo ciclo 54[s] il valore della fitness è pari ad uno ovvero ottima, mentre nell'ultima istanza si ottiene fitness nulla perché il valore del lower bound coincide con il valore della soluzione trovata, 15 stazioni, si ha quindi $F_1 = 1 + \frac{12-15}{15-12} = 1 - 1 = 0$.

Il terzo caso su cui si valida l'algoritmo è **Hahn**, uno dei problemi col minor numero di istanze, 5.

I risultati per quanto riguarda la minimizzazione del numero di stazioni sono rappresentati in figura 3.14.

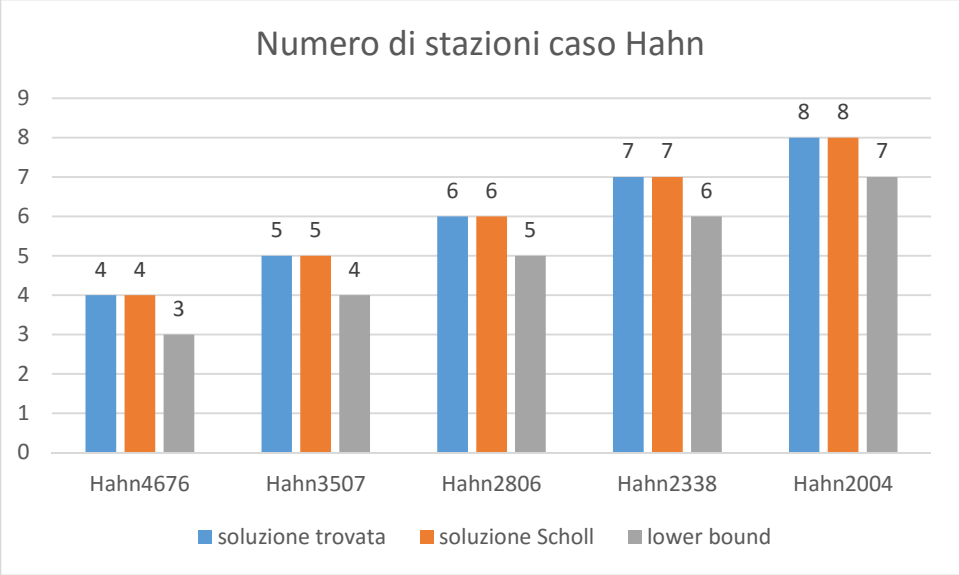

Figura 3.14 Numero di stazioni caso Hahn

In Hahn, il numero di stazioni trovate coincide con le soluzioni di Scholl in tutte le istanze, quindi si può affermare che l'algoritmo GenIAL ha un comportamento ottimo per gli stessi motivi di cui sopra.

Analizzando nel dettaglio i valori delle funzioni di fitness, indicati in figura 3.15, per il caso Hahn, si nota che, con tempo ciclo pari a 4676 [s], la funzione di fitness assume un valore nullo. Accade perché l'upper bound coindice con il numero di stazioni trovato dall'algoritmo.

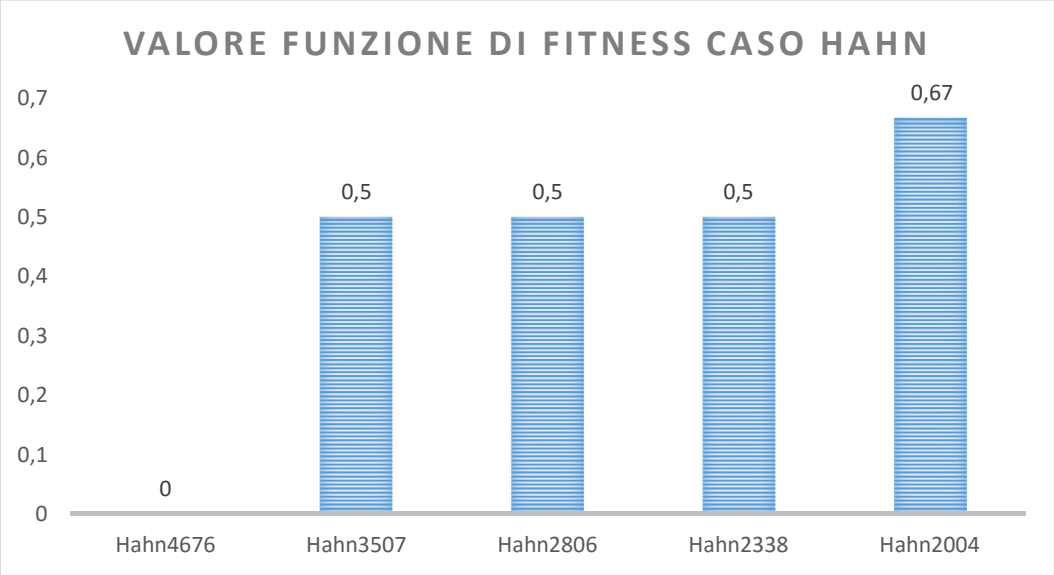

Figura 3.15 Valore funzione-fitness caso Hahn

Il **caso Lutz1** ha un order strength tra i più alti, 83.5. Il numero di stazioni che GenIAL riesce ad ottenere è indicato nel grafico in figura 3.16.

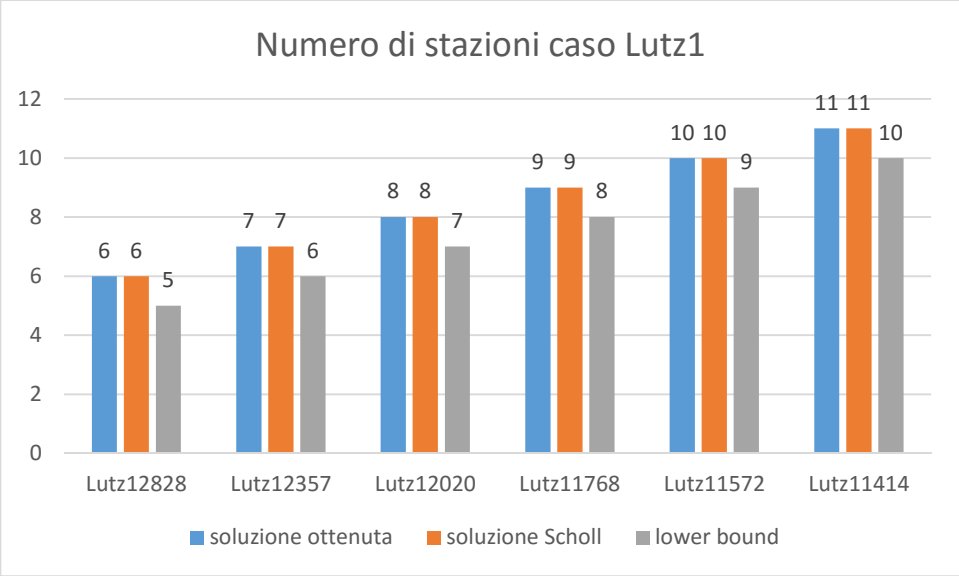

Figura 3.16 Numero di stazioni caso Lutz1

Ugualmente al caso Hahn, si ha coincidenza tra i valori Scholl e le soluzioni ottenute dall'algoritmo genetico. E in particolar modo non si hanno rilevanti problemi nei valori della fitness, tuttavia si nota che la discrepanza tra il lower bound e la soluzione di

GenIAL è pari ad uno in tutte le istanze del caso Lutz1 i cui risultati della funzione di fitness sono indicati in figura 3.17.

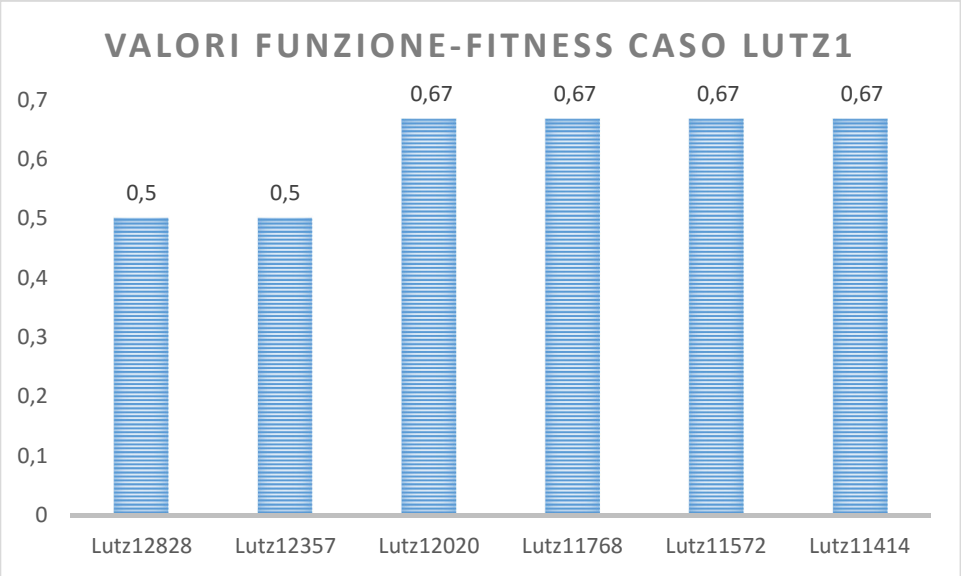

Figura 3.17 Valore funzione-fitness caso Lutz1

Il valore della funzione di fitness dovrebbe essere sempre prossimo all'uno ma non avviene poiché variano i valori dell'upper bound (7,8,10,11,12,13), che nelle prime due istanze si discostano di due unità mentre nelle rimanenti si discostano di tre unità.

Il **caso Tonge** presenta il quantitativo più alto di istanze, 16. L'algoritmo GenIAL ottiene i risultati seguenti:

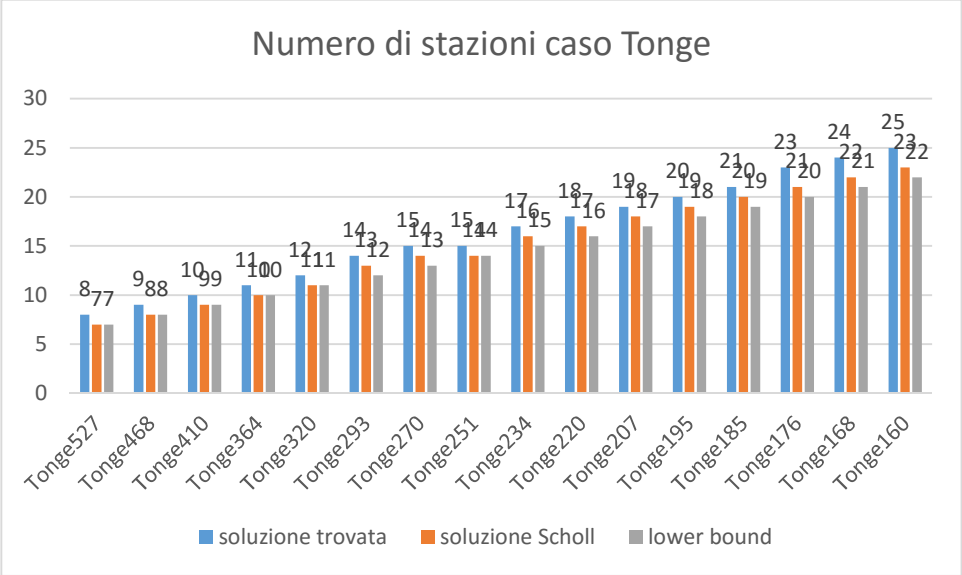

Figura 3.18 Numero di stazioni caso Tonge

In tutte le istanze, l'algoritmo si discosta sia da Scholl che dal lower bound, mentre in sei istanze le soluzioni di Scholl riescono a coincidere con il valore migliore che è possibile individuare. In alcune istanze, come le ultime tre, il cui il tempo ciclo si restringe, le soluzioni dell'algoritmo si allontanano da Scholl di due unità. Tali esiti si manifestano nella funzione di fitness nel seguente modo:

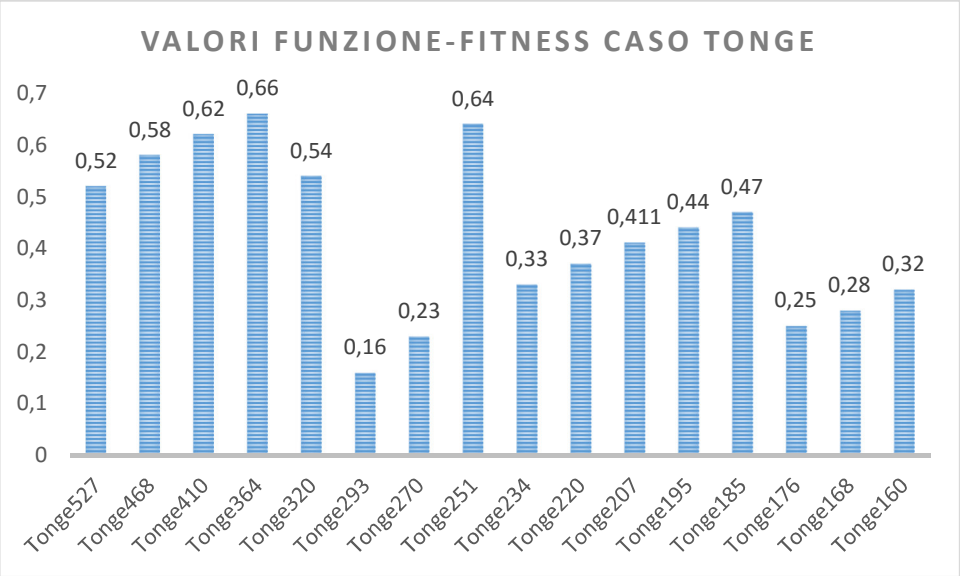

Figura 3.19 Valore funzione-fitness caso Tonge

La fitness è più bassa nei casi in cui GenIAL si distanzia più di una unità dal lower bound. Tuttavia si può affermare che i valori nelle prime due istanze sottostimano la soluzione individuata che differisce di un punteggio dal lower bound.

L'ultimo caso su cui si testa l'algoritmo per la minimizzazione del numero di stazioni è **Warnecke**, anch'esso suddiviso in 16 istanze di problema. I risultati sono presentati nel grafico di figura 3.20:

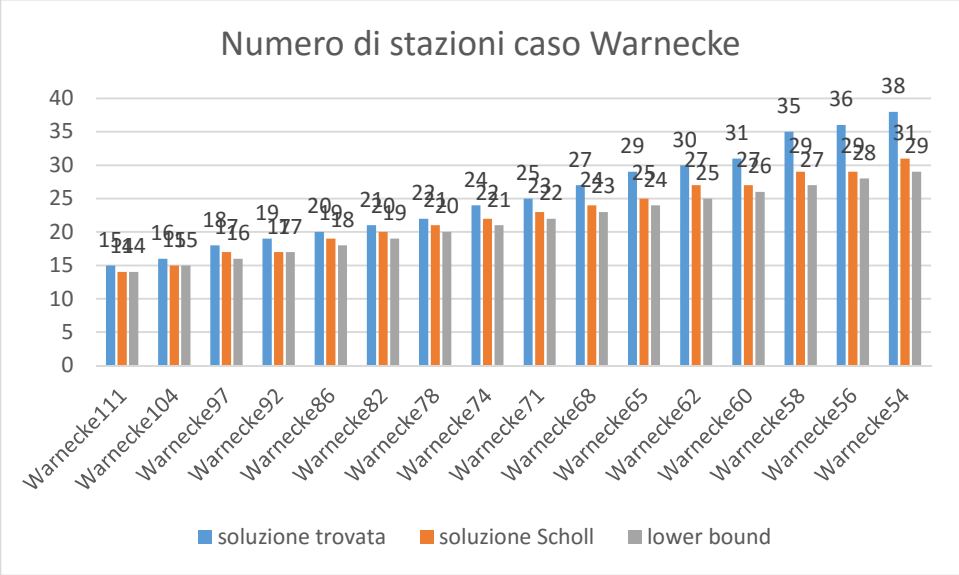

Figura 3.20 Numero di stazione caso Warnecke

In Warnecke si individuano le peggiori soluzioni, che si allontanano da quelle ottime anche di 6 unità. La funzione di fintess presenta un andamento diverso da quelli fin'ora risontrati, com'è possibile notare da figura 3.21.

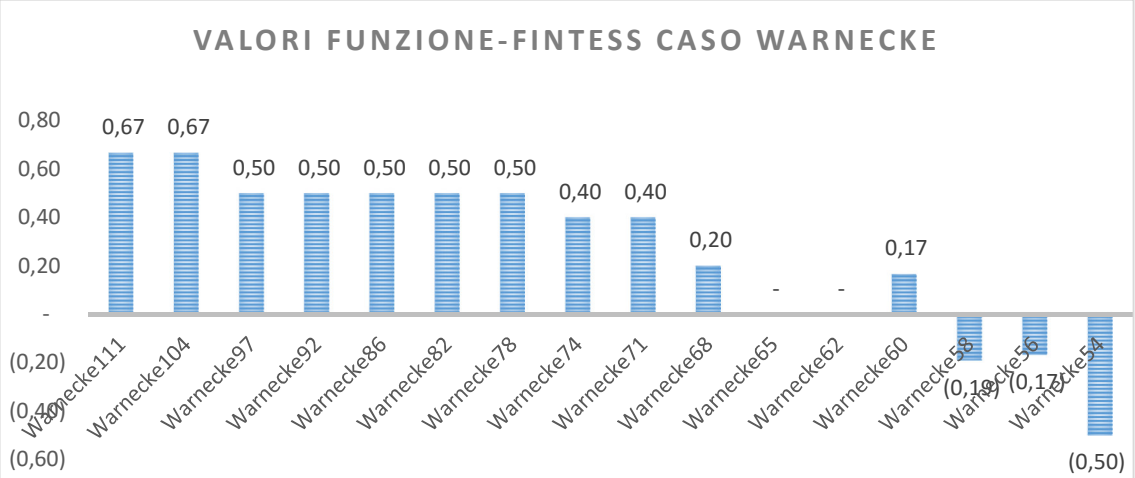

Figura 3.21 Valore funzione-fitness caso Warnecke

La funzione ottiene valori negativi negli ultimi 3 casi. Per la prima volta, l'algoritmo genetico sfora l'upper bound, trovando un numero di stazioni che è addirittura peggiore del caso meno vantaggioso. Nel dettaglio, l'upper bound è uguale a 33,34 e 35 mentre le soluzioni trovate da GenIAL sono 35,36 e 38.

Validato l'algoritmo sulla capacità di raggiungere soluzioni ottime per i problemi di tipo SALBP-1, si procede con la validazione dell'obiettivo inerente la minimizzazione delle **skill richieste**.

Si ricorda che ogni operazione può richiedere un differente livello di abilità per essere compiuta:

- Nessun livello di abilità richiesto, codificato con zero;
- Basso livello di abilità richiesto, codificato con uno;
- Elevato livello di abilità richiesto, codificato con due.

Minimizzare le skill richieste tra le stazioni significa allocare le operazioni sulle stazioni in modod tale che quelle che richiedono la stessa skill siano allocate alla medesima stazione.

Il valore delle skill è stato assegnato alle operazioni seguendo differenti metodologie, necessarie per comprendere quale andamento assumessero i valori del lower bound e upper bound dell'obiettivo:

- Tutte le skill pari ad uno;
- Tutte le skill pari a due;
- Skill assegnate random in modo che siano presenti tutte e tre le tipologie di abilità.

Le soluzioni trovate dall'algoritmo per le 58 istanze sono presentate in tabella 3.3.

Caso di studio	Skill richieste
Barthold403	27
Barthold434	25
Barthold470	25
Barthold513	22
Barthold564	19
Barthold626	19
Barthold705	17
Barthold805	14
Gunther41	16
Gunther44	16
Gunther49	14
Gunther54	13
Gunther61	12
Gunther69	12
Gunther81	10
Hahn2004	15
Hahn2338	14
Hahn2806	10
Hahn3507	9
Hahn4676	7
Lutz11414	15
Lutz11572	14
Lutz11768	14
Luzt12020	11
Lutz12357	11
Lutz12828	9
Tonge160	32
Tonge168	35
Tonge176	33
Tonge185	34
Tonge195	35
Tonge207	30
Tonge220	29
Tonge234	30
Tonge251	25

Tonge270	22
Tonge293	24
Tonge320	22
Tonge364	19
Tonge410	17
Tonge468	13
Tonge527	14
Warnecke54	41
Warnecke56	40
Warnecke58	37
Warnecke60	39
Warnecke62	33
Warnecke65	36
Warnecke68	36
Warnecke71	34
Warnecke74	32
Warnecke78	33
Warnecke82	33
Warnecke86	30
Warnecke92	29
Warnecke97	26
Warnecke104	24
Warnecke111	24

Tabella 3.3 Risultati per le skills richieste

È molto semplice notare che i valori delle skill assumono valori eccessivamente elevati rispetto a quelli del lower bound.

Nel dettaglio si riporta l'esempio del caso Hahn con tempo ciclo 4676 [s] formato da 53 operazioni. Si sono svolti alcuni test con le configurazioni riportate in tabella 3.4.

La configurazione con skill tutte pari a zero non è stata utilizzata perché in quel caso non esiste l'ottimizzazione dell'allocazione delle operazioni poiché tutte richiedono abilità nulla. Per quanto concerne il *casoA* e *casoB*, le cui configurazioni sono presenti nella tabella sottostante, si utilizzano per spiegare l'andamento del lower bound e dell'upper bound nonostante, in pratica, la minimizzazione delle skill non ha alcuna funzionalità. Le altre tre configurazioni sono più reali da un punto di vista industriale poiché la distribuzione del livello di abilità richiesto è meno omogenea.

N° operazione	Task time [s]	Skill richiesta				
		Caso A	*Caso B*	*Caso C*	*Caso D*	*Caso E*
1	970,98	1	2	1	0	1
2	141,96	1	2	2	1	2
3	141,96	1	2	1	2	2
4	141,96	1	2	2	0	0
5	102,96	1	2	1	1	2
6	96	1	2	2	2	1
7	99	1	2	1	0	1
8	1206,96	1	2	2	1	0
9	159,96	1	2	1	2	0
10	180	1	2	2	0	0
11	81,96	1	2	1	1	2
12	60	1	2	2	2	2
13	111,96	1	2	1	0	1
14	420	1	2	2	1	1
15	1555,98	1	2	1	2	0
16	237,96	1	2	2	0	1
17	258,96	1	2	1	1	2
18	124,98	1	2	2	2	1
19	600,96	1	2	1	0	1
20	79,98	1	2	2	1	1
21	79,98	1	2	1	2	0
22	69,96	1	2	2	0	0
23	88,98	1	2	1	1	2
24	88,98	1	2	2	2	2
25	105	1	2	1	0	2
26	330	1	2	2	1	1
27	132	1	2	1	2	1
28	69	1	2	2	0	0
29	99	1	2	1	1	0
30	69,96	1	2	2	2	2
31	69,96	1	2	1	0	2
32	157,98	1	2	2	1	1
33	190,98	1	2	1	2	1
34	69,96	1	2	2	0	1
35	52,98	1	2	1	1	2
36	49,98	1	2	2	2	2
37	124,98	1	2	1	0	0
38	352,98	1	2	2	1	0
39	69,96	1	2	1	2	1
40	127,98	1	2	2	0	1
41	64,98	1	2	1	1	2

42	1774,98	1	2	2	2	2
43	90,96	1	2	1	0	0
44	90,96	1	2	2	1	0
45	112,98	1	2	1	2	2
46	486,96	1	2	2	0	1
47	138	1	2	1	1	1
48	79,98	1	2	2	2	1
49	79,98	1	2	1	0	1
50	64,98	1	2	2	1	0
51	39,96	1	2	1	2	0
52	741,96	1	2	2	0	0
53	1084,98	1	2	1	1	2

Tabella 3.4 Configurazioni per testare i bound delle skill richieste

La soluzione che si è ottenuta con l'attuale metodologia di calcolo dei bound delle skill è illustrata nella tabella 3.5. Il LB_S e l'UB_S sono rispettivamente il lower bound e l'upper bound delle skill richieste, l'UB_N è il limite superiore dell'insieme del numero di stazioni, le skill rappresentano il valore delle abilità richieste dal cromosoma migliore, e infine la funzione di fitness è il valore della soluzione individuata.

Caso	**LB_S**	**UB_S**	**UB_N**	**Skill**	**Funzione-fitness**
A	1	53	4	4	0.9423
B	2	106	4	8	0.9434
C	2	30	4	7	0.8214
D	2	21	4	7	0.7386
E	2	25	4	8	0.7391

Tabella 3.5 Risultati ottenuti dall'algoritmo con le modalità precedenti per i bound delle skill

Nel dettaglio, il lower bound delle skill richieste assume solo tre valori:

- Zero, quando tutte le skill sono pari a zero;
- Uno, quando si hanno skill pari a uno e/o zero;
- Due, quando le skill assumono valori 0, 1 e 2.

Attualmente il lower bound delle skill viene calcolato come somma degli equipment necessari per il montaggio.

Per quanto concerne l'upper bound delle skill, esso viene collegato al valore dell'upper bound del numero di stazioni e indica la sommatoria delle massime abilità tra le stazioni considerando come numero di stazioni l'upper bound di N. Nel caso peggiore ogni stazione richiede la skill più elevata possibile. Ad esempio, nei casi A,B,C,D ed E si dovrebbe ottenere un valore dell'UB_S pari a 8 essendo il valore peggiore del numero di stazioni uguale a 4 ovvero la massima skill (2) moltiplicata per il numero di stazioni, UB_N.

Un ulteriore obiettivo su cui testare l'algoritmo è l'ottimizzazione degli **equipment richiesti** tra le stazioni.

Si ricorda che l'algoritmo suppone che all'interno dell'impianto produttivo si possano utilizzare fino a dieci tipologie differenti di equipment, quindi ogni stazione al massimo potrà richiederne 10. Ad ogni operazione viene assunto un valore dell'equipment che è compreso nell'intervallo di valori interi [1,10].

I risultati ottenuti per le 58 istanze sono mostrati in tabella 3.6.

Caso di studio	Equipment richiesti
Barthold403	87
Barthold434	84
Barthold470	79
Barthold513	82
Barthold564	65
Barthold626	54
Barthold705	33
Barthold805	15
Gunther41	26
Gunther44	22
Gunther49	21
Gunther54	22
Gunther61	19
Gunther69	20
Gunther81	17
Hahn2004	45
Hahn2338	42

Hahn2806	38
Hahn3507	32
Hahn4676	16
Lutz11414	19
Lutz11572	22
Lutz11768	18
Luzt12020	17
Lutz12357	15
Lutz12828	23
Tonge160	54
Tonge168	49
Tonge176	55
Tonge185	47
Tonge195	53
Tonge207	57
Tonge220	49
Tonge234	41
Tonge251	38
Tonge270	56
Tonge293	45
Tonge320	48
Tonge364	36
Tonge410	40
Tonge468	43
Tonge527	70
Warnecke54	52
Warnecke56	49
Warnecke58	47
Warnecke60	44
Warnecke62	50
Warnecke65	48
Warnecke68	36
Warnecke71	39
Warnecke74	44
Warnecke78	47
Warnecke82	50
Warnecke86	52
Warnecke92	45
Warnecke97	48
Warnecke104	54
Warnecke111	58

Tabella 3.6 Risultati per gli equipment

Anche qui si ha una visibilissima divergenza tra i valori ottenuti e i lower bound.

Il lower bound assume solo dieci valori, dall'uno al dieci. Nel caso migliore ogni stazione richiederà un solo tool, tuttavia le stazioni possono essere più di dieci quindi anche supponendo che l'algoritmo riesca ad allocare le operazioni alle stazioni riuscendo ad ottenere la richiesta di un tool per ogni workstation, il valore deve superare le 10 unità e ciò non avviene nei casi testati. Il lower bound degli equipment è calcolato come sommatoria degli equipment necessari per il montaggio.

Per quanto concerne l'upper bound della minimizzazione degli equipment, anche in questo caso, viene connesso all'upper bound del numero di stazioni e si inseriscono gli equipment in modo tale che non si ripetano all'interno delle stazioni; anche per l'upper bound degli equipment si presenta lo stesso problema riscontrato per l'upper bound delle skills.

Si prenda ad esempio il caso Lutz1 con tempo ciclo pari a 2828 [s] e numero di operazioni 32. Per i test sono state utilizzate diverse configurazioni, presentate in tabella 3.7. Sono dieci configurazioni, dalla A alla L, dalla richiesta di un solo equipment, uguale per tutte le stazioni, fino a 10 equipment differenti per il caso L.

N° operazione	Task time [s]	Caso A	Caso B	Caso C	Caso D	Caso E	Caso F	Caso G	Caso H	Caso I	Caso L
1	457,98	1	1	1	1	1	1	1	1	1	1
2	276	1	2	2	2	2	2	2	2	2	2
3	519,96	1	1	3	3	3	3	3	3	3	3
4	1399,98	1	2	1	4	4	4	4	4	4	4
5	351,96	1	1	2	1	5	5	5	5	5	5
6	195,96	1	2	3	2	1	6	6	6	6	6
7	213,96	1	1	1	3	2	1	7	7	7	7
8	456	1	2	2	4	3	2	1	8	8	8
9	645,96	1	1	3	1	4	3	2	1	9	9
10	511,98	1	2	1	2	5	4	3	2	1	10
11	408	1	1	2	3	1	5	4	3	2	1
12	261,96	1	2	3	4	2	6	5	4	3	2
13	543,96	1	1	1	1	3	1	6	5	4	3
14	201,96	1	2	2	2	4	2	7	6	5	4
15	457,98	1	1	3	3	5	3	1	7	6	5
16	693,96	1	2	1	4	1	4	2	8	7	6
17	615,96	1	1	2	1	2	5	3	1	8	7
18	678	1	2	3	2	3	6	4	2	9	8
19	327,96	1	1	1	3	4	1	5	3	1	9
20	324	1	2	2	4	5	2	6	4	2	10
21	99,96	1	1	3	1	1	3	7	5	3	1
22	351,96	1	2	1	2	2	4	1	6	4	2
23	543,96	1	1	2	3	3	5	2	7	5	3
24	499,98	1	2	3	4	4	6	3	8	6	4
25	381,96	1	1	1	1	5	1	4	1	7	5
26	858	1	2	2	2	1	2	5	2	8	6
27	387,96	1	1	3	3	2	3	6	3	9	7
28	351,96	1	2	1	4	3	4	7	4	1	8
29	426	1	1	2	1	4	5	1	5	2	9
30	237,96	1	2	3	2	5	6	2	6	3	10
31	258	1	1	1	3	1	1	3	7	4	1
32	195,96	1	2	2	4	2	2	4	8	5	2

Tabella 3.7 Configurazioni caso Lutz1 per il calcolo dei bound degli equipment

I risultati, ottenuti con l'attuale modaità di calcolo dei bound, sono presentati in tabella 3.8, con riferimenti ai valori del lower bound, dell'upper bound e degli equipment utilizzati dalla soluzione considerata migliore.

Caso	LB_E	UB_E	Equipment
A	1	32	6
B	2	32	10
C	3	32	14
D	4	32	18
E	5	32	20
F	6	32	23
G	7	32	24
H	8	32	25
I	9	32	25
L	10	32	30

Tabella 3.8 Risultati ottenuti per gli equipment con la modalità di calcolo attuale

Il lower bound degli equipment assume, qualsiasi configurazione si utilizzi, un valore da 1 a 10 poiché accorpa tra loro le operazioni che richiedono lo stesso tool senza far riferimento, anche in questo caso, al tempo ciclo a disposizione delle stazioni.

Mentre l'upper bound rimane fisso sul valore 32, tante quante sono le operazioni di cui è costituito il caso Lutz1.

Passando all'ottimizzazione dei **cambi di direzione** è utile chiarire il significato di tale obiettivo.

Ogni operazione può essere svolta su una direzione ±x, ±y e ±z, che viene settate su Matlab rispettivamente con ±1, ±2 e ±3. Ottimizzare tale obiettivo significa rendere sequenziali le operazioni svolte sulla medesima direzione.

I risultati per i cambi di direzione sono illustrati nella tabella 3.8.

Casi di studio	Cambi di direzione
Barthold403	109
Barthold434	114
Barthold470	117
Barthold513	111
Barthold564	106
Barthold626	103
Barthold705	109
Barthold805	108
Gunther41	20
Gunther44	21
Gunther49	23
Gunther54	22
Gunther61	22
Gunther69	20
Gunther81	18
Hahn2004	16
Hahn2338	18
Hahn2806	16
Hahn3507	14
Hahn4676	18
Lutz11414	25
Lutz11572	25
Lutz11768	28
Luzt12020	25
Lutz12357	22
Lutz12828	25
Tonge160	51
Tonge168	54
Tonge176	53
Tonge185	50
Tonge195	54
Tonge207	55
Tonge220	50
Tonge234	48
Tonge251	48
Tonge270	49
Tonge293	47
Tonge320	51
Tonge364	54
Tonge410	53
Tonge468	47
Tonge527	50

Warnecke54	43
Warnecke56	42
Warnecke58	39
Warnecke60	44
Warnecke62	46
Warnecke65	48
Warnecke68	41
Warnecke71	40
Warnecke74	38
Warnecke78	40
Warnecke82	42
Warnecke86	43
Warnecke92	38
Warnecke97	43
Warnecke104	46
Warnecke111	43

Tabella 3.8 Risultati per i cambi di direzione

Nel dettaglio, in tabella 3.9, si presentano alcune delle configurazioni utilizzate per testare il caso Warnecke con tempo ciclo 111 [s]. Sono sei configurazioni che vanno da un massimo di 6 direzioni richieste ad un minimo di 1 direzione uguale per tutte le operazioni.

N°operazione	Task time [s]	Caso A	Caso B	Caso C	Caso D	Caso E	Caso F
1	9,96	-1	-1	-1	2	1	1
2	52,98	1	-1	-1	2	1	1
3	40,98	-2	-1	-1	1	1	1
4	36	2	-1	-1	2	2	1
5	34,98	-3	-1	-1	-2	2	1
6	16,98	3	2	2	-2	1	1
7	33,96	-1	2	2	-2	1	1
8	22,98	1	2	2	-2	1	1
9	13,98	-2	2	2	1	1	1
10	51,96	2	2	2	1	1	1
11	33	-3	-1	-1	1	1	1
12	33,96	3	2	2	1	1	1
13	12	-1	-3	-3	1	2	1
14	51,96	1	1	-3	2	2	1
15	12	-2	1	-3	2	2	1
16	33	2	1	3	2	2	1
17	43,98	-3	1	3	2	2	1
18	6,96	3	1	3	1	1	1
19	15	-1	3	3	1	1	1

20	12,96	1	3	3	1	1	1
21	28,98	-2	-3	-3	1	1	1
22	36,96	2	-3	-3	-2	1	1
23	42,96	-3	-3	-3	-2	1	1
24	22,98	3	-3	-3	-2	1	1
25	24	-1	2	2	-2	1	1
26	9	1	2	2	-2	1	1
27	15,96	-2	2	2	-2	1	1
28	12	2	2	2	-2	2	1
29	25,98	-3	2	2	2	2	1
30	21,96	3	-1	-1	2	2	1
31	51	-1	-1	-1	2	2	1
32	46,98	1	-1	-1	2	2	1
33	33,96	-2	-1	-1	2	2	1
34	22,98	2	-1	-1	2	2	1
35	12	-3	1	2	2	1	1
36	51,96	3	2	2	1	1	1
37	12	-1	1	2	1	1	1
38	33	1	2	2	1	1	1
39	43,98	-2	2	2	1	1	1
40	6,96	2	3	3	1	1	1
41	15	-3	3	3	1	1	1
42	12,96	3	2	2	1	1	1
43	28,98	-1	3	3	1	1	1
44	36,96	1	1	3	1	1	1
45	42,96	-2	-3	-3	-2	1	1
46	22,98	2	1	-3	-2	1	1
47	24	-3	1	-3	-2	1	1
48	9	3	-3	-3	-2	1	1
49	15,96	-1	1	-3	-2	1	1
50	12	1	2	2	-2	1	1
51	25,98	-2	2	2	2	1	1
52	12	2	2	2	2	1	1
53	51,96	-3	2	2	1	1	1
54	12	3	2	2	1	1	1
55	33	-1	-1	-1	1	1	1
56	43,98	1	1	-1	1	1	1
57	6,96	-2	-1	-1	1	1	1
58	15	2	1	-3	1	1	1

Tabella 3.9 Configurazioni caso Warnecke per il calcolo dei bound dei cambi di direzione

I risultati, ottenuti con l'attuale modaità di calcolo dei bound, sono presentati in tabella 3.10, con riferimento ai valori del lower bound, dell'upper bound e dei cambi di direzione, compiuti dalla soluzione considerata migliore.

Caso	LB_D	UB_D	Cambi di direzione
A	5	57	43
B	4	49	32
C	3	48	29
D	2	42	25
E	1	28	8
F	0	0	0

Tabella 2.10 Risultati ottenuti per i cambi di direzione con la modalità di calcolo attuale

Il lower bound assume sempre valori 0,1,2,3,4,5. Si consideri l'esempio di figura 3.22, e si supponga di avere 8 operazioni che in totale richiedono 3 direzioni; il lower bound le sequenzia in modo da accorpare le operazioni svolte nella stessa direzione, mentre l'upper bound le sequenzia in modo che si alternino le differenti direzioni.

1	-1	-1	-1	2	1	2	1	Operazioni
1	1	1	-1	-1	-1	2	2	LB = 2
1	-1	2	1	-1	1	2	-1	UB = 7

Tabella 3.22 Lower bound e upper bound dei cambi di direzione con 3 direzioni

Non sono state notate particolari difficoltà sulle modalità di valutazione dei bound inerenti la minimizzazione dei cambi di direzione.

E nel dettaglio, il lower bound dei cambi di direzione viene calcolato come indicato nell'equazione 6:

$$LB_D = \sum D - 1 \qquad \text{(Eq. 6)}$$

Dove

D = numero di direzioni richiesto dalle operazioni.

Mentre per l'upper bound dei cambi di direzione, viene calcolato tramite la seguente procedura:

1. Calcolare il lower bound di cambi di orientamento;
2. Se LB_D= 0 allora UB_D =0
3. Se LB_D=1 allora UB = 2x, dove x è il numero di volte in cui compare la direzione meno presente
4. Se LB_D assume valori tra 2 e 5

   Se $n_i > \frac{k}{2}$ dove i = ±x, ±y e ± z allora UB_D = 2x dove x = k - n_i ed n_i è uguale al numero di volte in cui compare la direzione i-esima.

   Altrimenti UB_D = k – 1 dove k = numero di operazioni.

L'ultimo obiettivo su cui si testa l'algoritmo è la minimizzazione della **workload variance**, il cui lower bound è sempre pari a zero. Infatti, nella migliore delle ipotesi, tutte le stazioni hanno un tempo di processamento delle operazioni assegnategli pari al tempo ciclo.

Una problematica che si è potuta riscontrare è la differenza che in media si ha tra la workload variance trovata dall'algoritmo e l'upper bound, indicata in tabella 3.11.

Caso di studio	Workload-variance	UB_WV
Barthold403	0,0608	6,5144
Barthold434	0,1157	7,9193
Barthold470	0,1331	9,7584
Barthold513	0,1438	12,2217
Barthold564	0,1999	15,5878
Barthold626	0,0055	20,3207
Barthold705	0,4464	27,2093
Barthold805	0,0051	37,6198

Gunther41	0,0156	0,0734
Gunther44	0,0047	0,0899
Gunther49	0,0034	0,1158
Gunther54	0,1498	5,64
Gunther61	0,0017	0,2009
Gunther69	0,0037	0,2768
Gunther81	0,0083	0,3368
Hahn2004	14,826	234,226
Hahn2338	31,806	284,6511
Hahn2806	1,3215	446,1499
Hahn3507	8,8904	759,0666
Hahn4676	1,138	10,2347
Lutz11414	1,3737	98,551
Lutz11572	1,4703	128,5126
Lutz11768	3,4525	172,0562
Luzt12020	0,4654	237,9275
Lutz12357	0,9517	289,1494
Lutz12828	1,5665	453,2004
Tonge160	0,0741	0,7473
Tonge168	0,0482	0,8522
Tonge176	0,0507	1,1884
Tonge185	0,0612	1,3608
Tonge195	0,0578	1,5693
Tonge207	0,0754	1,8252
Tonge220	0,0601	2,1381
Tonge234	0,0299	2,1098
Tonge251	0,0601	2,5348
Tonge270	0,0362	3,081
Tonge293	0,0383	3,7988
Tonge320	0,0345	4,7585
Tonge364	0,0163	6,4038
Tonge410	0,0157	8,5968
Tonge468	0,0203	11,8275
Tonge527	0,0201	15,3894
Warnecke54	0,0087	0,0702
Warnecke56	0,0106	0,078
Warnecke58	0,01	0,0853
Warnecke60	0,0106	0,0928
Warnecke62	0,0114	0,1018
Warnecke65	0,0113	0,1157
Warnecke68	0,009	0,1313
Warnecke71	0,0124	0,1475
Warnecke74	0,0159	0,1654
Warnecke78	0,0173	0,2327
Warnecke82	0,017	0,2659

Warnecke86	0,0118	0,3052
Warnecke92	0,011	0,3583
Warnecke97	0,018	0,4158
Warnecke104	0,0078	0,4128
Warnecke111	0,0072	0,4936

Tabella 3.11 Valori Workload variance confrontati con UB_WV

Tale divergenza tende a sovrastimare la funzione di fintess essendo il denominatore del valore normalizzato sempre molto elevato rispetto al numeratore, infatti la funzione può essere scritta $F_5 = 1 + \frac{-x}{UB\_WV}$, poiché il lower bound è sempre pari a zero.

Mentre l'upper bound viene calcolato prendendo a riferimento l'upper bound del numero di stazioni, si riempono le stazioni fino a quando il tempo che impiega la stazione a compiere tutte le operazioni non è il massimo possibile, vicino al tempo ciclo, e alle rimanenti stazioni, senza nessuna operazione allocata, viene assegnato workstation time pari a zero. L'upper bound viene calcolato secondo la seguente formulazione:

$$UB_{WV} = \sum_i \frac{(T_{a_i} - T_{am})^2}{N} \qquad \text{(eq.7)}$$

Dove:

- T_{a_i} = tempo di attesa della stazione i-esima;
- T_{am} = tempi di attesa medi lungo la linea;
- N = numero di stazioni ovvero valore dell'upper bound.

Il problema che si riscontra nella minimizzazione della workload variance è valido anche per gli altri problemi mono-obiettivo.

È usuale, infatti, che l'algoritmo sovrastimi i valori dell'upper bound e sottostimi il lower bound in tutti gli obiettivi proposti tranne nella minimizzazione dei cambi di direzione. Avere valori troppo elevati per il limite superiore dell'insieme e invece valori troppo bassi per il limite inferiore dell'insieme causa una errata valutazione della soluzione raggiunta dall'algoritmo.

Confrontando i grafici proposti in questo paragrafo, si è compreso che soluzioni molto buone, a volte, sono sottovalutate dalla funzione di fitness provocando un'errata analisi dell'utente che percepisce la soluzione non ottimale. In altri casi invece, una soluzione mediocre viene valutata molto buona. Esistono quindi divergenze tra la reale bontà della soluzione e il valore della funzione di fitness che dovrebbe permettere all'utente di percepire la qualità della soluzione. Su questa divergenza si baserà, nel prossimo capitolo, il primo miglioramento dell'algoritmo per quanto concerne l'ottimizzazione mono-obiettivo.

3.3.2 Problemi multi-obiettivo

Compreso come GenIAL lavora su problemi mono-obiettivo, si continua l'analisi valutando l'algoritmo quando si vogliono raggiungere cinque obiettivi contemporaneamente, la minimizzazione del numero di stazioni, la minimizzazione degli equipment richiesti, la minimizzazione dei cambi di direzione, la minimizzazione delle skill richieste e la minimizzazione della workload-variance.

In molti problemi reali viene richiesta la simultanea ottimizzazione di una serie di obiettivi. Quando un problema di ottimizzazione richiede la presenza di più di una funzione obiettivo, il problema di trovare soluzioni ottimali prende il nome di ottimizzazione multi-obiettivo.

Una linea di assemblaggio è funzione di diversi parametri che devono essere adeguatamente bilanciati per ottenere massima produttività ed efficienza. È fondamentale riuscire a raggiungere numerosi obiettivi contemporaneamente per avere una migliore gestione dei costi, dei tempi e delle risorse aziendali.

Nel dettaglio dell'algoritmo genetico GenIAL, si parla di problema multi-obiettivo quando almeno due dei pesi delle funzioni sono diversi da zero.

Ai fini degli scopi di questa tesi, si considerano diversi da zero tutti i pesi delle funzioni.

Per compiere tutti i test si sono impostati, per le cinque funzioni obiettivo, i pesi indicati di seguito:

- η_1 = 0.2;
- η_2 = 0.1;
- η_3 = 0.2;
- η_4 = 0.2;
- η_5 = 0.3.

La scelta di tale combinazione considera l'importanza che viene attribuita agli obiettivi da un punto di vista logico all'interno delle linee di assemblaggio e anche quante volte in letteratura sono presenti problemi di bilanciamento che tentando di raggiungere tali obiettivi. Alla workload variance è stato assegnato peso maggiore poiché l'ottimizzazione di tale obiettivo tende a facilitare il raggiungimento del primo obiettivo che riguarda la minimizzazione del numero di stazioni. La valutazione su come l'algoritmo si comporta in casi multi-obiettivo si basa principlamente sul confronto di questi risultati con quelli ottenuti con il mono-obiettivo, per comprendere quanto l'algoritmo si discosti da essi. In tabella 3.12, vengono illustrati i risultati ottenuti per le 58 istanze quando l'algoritmo viene utilizzato per risolvere problemi multi-obiettivo. Tali risultati possono essere confrontati con quelli ottenuti nel mono-obiettivo e presenti nei paragrafi precedenti.

Caso di studio	N	S	E	D	WV
Barthold403	15	30	93	110	0,3137
Barthold434	14	28	87	117	0,2106
Barthold470	13	26	84	121	0,2058
Barthold513	12	24	87	113	0,3361
Barthold564	11	22	78	116	0,5491
Barthold626	10	20	75	115	0,8246
Barthold705	9	18	71	119	0,8405
Barthold805	8	15	15	117	0,0089
Gunther41	15	20	26	24	0,0249
Gunther44	13	20	24	25	0,0124
Gunther49	11	18	25	27	0,0047
Gunther54	11	19	26	26	0,0049
Gunther61	9	14	24	26	0,0135
Gunther69	8	13	23	24	0,02

Gunther81	7	13	20	22	0,0282
Hahn2004	8	16	46	18	22,272
Hahn2338	7	14	42	24	71,1403
Hahn2806	6	10	39	20	4,2403
Hahn3507	5	10	39	18	8,8904
Hahn4676	4	7	32	18	17,2117
Lutz11414	11	17	22	27	2,9409
Lutz11572	10	17	24	28	1,9823
Lutz11768	9	15	20	30	6,3119
Luzt12020	8	15	18	26	5,5008
Lutz12357	7	13	17	25	9,3592
Lutz12828	6	10	24	26	2,5687
Tonge160	25	39	62	62	0,0667
Tonge168	24	41	56	62	0,1022
Tonge176	23	37	63	60	0,1031
Tonge185	22	36	54	62	0,0601
Tonge195	20	35	63	63	0,0969
Tonge207	19	35	60	58	0,0553
Tonge220	18	32	56	59	0,0893
Tonge234	16	29	59	61	0,0397
Tonge251	15	28	56	64	0,0584
Tonge270	14	23	55	64	0,0532
Tonge293	13	25	49	59	0,073
Tonge320	12	24	52	56	0,0903
Tonge364	10	19	54	63	0,0294
Tonge410	9	18	46	56	0,0473
Tonge468	8	15	47	51	0,0493
Tonge527	7	14	40	58	0,1112
Warnecke54	34	43	57	48	0,0091
Warnecke56	33	41	57	48	0,0111
Warnecke58	32	42	53	44	0,0144
Warnecke60	30	41	58	50	0,009
Warnecke62	31	41	53	50	0,0127
Warnecke65	29	39	55	51	0,0163
Warnecke68	26	38	58	54	0,0106
Warnecke71	25	39	56	54	0,0095
Warnecke74	25	41	56	45	0,0201
Warnecke78	23	34	56	47	0,0248
Warnecke82	22	36	52	46	0,0273
Warnecke86	20	35	54	50	0,0207
Warnecke92	19	32	54	45	0,0279
Warnecke97	18	28	54	48	0,0275
Warnecke104	16	30	55	49	0,0167
Warnecke111	15	26	55	49	0,0104

Tabella 3.12 Risultati ottenuti da GenIAL per i problemi multi-obiettivo

Riassumendo, nella modalità di selezione attuale, l'algoritmo valuta i cromosomi in funzione della loro fitness, data semplicemente dalla somma pesata dei risultati ottenuti dal cromosoma stesso per ogni obiettivo. Tuttavia si possono riscontrare alcune incongruenze nei valori di fitness raggiunti poiché si possono presentare casi ambigui in cui più cromosomi hanno fitness molto simile, ma uno di loro è migliore rispetto agli altri in alcuni obiettivi, in queste occasioni però l'algoritmo non riesce a notare le differenze nelle soluzioni perdendo così l'opportunità di proporre all'utente una soluzione migliore.

Si può presentare la situazione in cui i cromosomi abbiano un valore di fitness molto simile tra loro nonostante i valori degli obiettivi possano essere differenti. L'algoritmo GenIAL non è in grado di fare una distinzione tra i singoli valori di questi cromosomi considerando la scelta di un cromosoma rispetto che un altro indifferente. Questo, è connesso alla tipologia di selezione utilizzata dall'algoritmo, la cosiddetta roulette wheel selection che sceglie un cromosoma rispetto ad un altro esclusivamente in base al valore della funzione di fitness, data dalla somma dei valori normalizzati ottenuti per i singoli obiettivi. Maggiore è la funzione di fitness, maggiore sarà la possibilità di un cromosoma di essere selezionato. Ciò può andar bene quando i pesi degli obiettivi sono molto divergenti e distanti tra loro, ma comincia a presentare qualche problema quando i pesi dei singoli obiettivi sono molto simili tra loro, come nel caso che si è testato.

Capitolo 4

4 Miglioramenti dell'algoritmo

In questa sezione della tesi si illustrano, dati i problemi riscontrati nella validazione dell'algoritmo, le soluzioni proposte e i risultati derivanti dalla loro implementazione.

4.1 Miglioramenti proposti per i problemi mono-obiettivo

Validando l'algoritmo GenIAL tramite i casi di studio Scholl, si è notato che la problematica principale che si presenta nei problemi mono-obiettivo è l'incapacità della funzione di fitness di rispecchiare esattamente la bontà di una soluzione, in alcuni casi sovrastimata e in altri sottostimata.

Dalle analisi precedenti, si è compreso che tale difficoltà risiede nell'errata definizione dei limiti inferiore e superiore dei singoli obiettivi fatta eccezione per la minimizzazione dei cambi di direzione per cui non si è riscontrata alcuna difettosità.

I bounds rappresentano il limite superiore (upper) ed inferiore (lower) dell'insieme delle soluzioni possibili. Nell'attuale algoritmo genetico GenIAL, il lower bound è inferiore rispetto al valore minimo mentre l'upper bound risulta essere in alcuni casi maggiore in altri minore del valore massimo che può assumere l'insieme.

Tale difficoltà però non riguarda la minimizzazione del numero di stazioni in cui il lower bound è calcolato tramite l'equazione 3:

$$LB_N = \frac{T_p}{T_c} \qquad \text{(eq. 8)}$$

Dove

T_p = tempo di produzione dato dalla somma dei tempi di processamento di tutte le operazioni che costituiscono il processo di assemblaggio;

T_c = Tempo ciclo.

Mentre l'upper bound nella maggior parte dei casi viene sottostimato.

4.1.1 Upper Bound per il numero di stazioni

L'Upper bound del numero di stazioni è calcolato come spiegato nel paragrafo 3.3.1.

Al crescere del lower bound, l'upper bound aumenta di una predefinita percentuale rispetto al limite inferiore. Tali percentuali comportano errate valutazioni nella funzione di fintess. Nell'esempio del caso Hahn, riportato nella tabella 4.1, la funzione di fitness assume valori inferiori rispetto alla reale differenza tra N, numero di stazioni individuato dall'algoritmo, e il lower bound, LB_N. Nonostante l'algoritmo si discosti di una sola unità dal valore migliore, la funzione di fitness risulta mediocre.

Caso di studio	**Tempo ciclo [s]**	**LB_N**	**UB_N**	**N**	**Funzione di fintess**
Hahn	4676	3	4	4	0
Hahn	3507	4	6	5	0.5
Hahn	2806	5	7	6	0.5
Hahn	2338	6	8	7	0.5
Hahn	2004	7	10	8	0.667

Tabella 4.4 Valori dei bound del numero di stazioini calcolati con la modalità precedente

Se la soluzione ottenuta si discosta di una sola unità dal valore ottimale allora il valore della funzione di fintess dovrebbe muoversi intorno all'uno invece di collocarsi su valori nulli.

Si ricorda che la funzione di fitness è calcolata come segue $F_i = 1 + \frac{(LB-x)}{(UB-LB)}$. Per farne aumentare i valori basta aumentarne il denominatore quindi la differenza tra l'upper bound e il lower bound.

Tramite numerose prove attuate sull'algoritmo, il metodo di calcolo dell'upper bound è stato modificato nel seguente modo:

- UB_N= ⌈LB_N+0,3*LB_N⌉ se LB_N è minore o uguale a dieci stazioni;
- UB_N= ⌈LB_N+0,4*LB_N⌉ se LB_N è compreso tra le 10 e le 30 stazioni;
- UB_N= ⌈LB_N+0,5*LB_N⌉ se LB_N è compreso tra le 30 e le 40 stazioni;
- UB_N= ⌈LB_N+0,6*LB_N⌉ in tutti gli altri casi.

Ciò migliora i valori delle funzioni di fitness, non solo aumentandoli ma anche diminuendoli nei casi di sovrastima.

4.1.2 Bound per le skill richieste

Il valore delle skill richieste si valuta come somma delle abilità tra le stazioni che costituiscono la linea di montaggio.

Nel capitolo precedente, si è potuto notare che il lower bound delle skill viene sottostimato dall'algoritmo poiché il suo valore è sempre pari a 1 o 2 e da come si è definito il lower bound ovvero somma delle skill richieste tra le stazioni, non è plausibile che il risultato sia sempre compreso in questo intervallo. Il lower bound ottiene i precedenti risultati perché l'algoritmo ipotizza che tutte le operazioni siano allocate nella stessa stazione considerando un valore infinito per il tempo ciclo. Se le

abilità richieste, dalle operazioni allocate sulla stazione, sono sia bassa che alta, quindi sia valore 1 che 2, il risultato del lower bound è 2, massima skill richiesta dalla stazione.

L'upper bound invece assume valori molto elevati. L'algoritmo connette l'UB_S con l'UB_N, supponendo che in ogni stazione sia posta almeno una operazione che richiede la massima skill (2) così che il livello di skill richiesto nella peggiore delle ipotesi sia 2*UB_N. Teoricamente il comportamento dell'upper bound dovrebbe essere quello appena spiegato ma in pratica i valori sono sempre differenti rispetto a ciò che ci si aspetta.

Riassumendo, il limite inferiore dell'insieme possibile di soluzioni è troppo sottostimato, e quindi i valori della funzione di fitness risultano scorretti. Questo non è l'unico errore all'interno della funzione di fitness, infatti anche l'upper bound viene aumentato, implicando un denominatore altissimo e un valore della funzione di fitness piuttosto elevato rispetto al valore non particolarmente buono della soluzione.

Si introduce uno schema ben preciso per calcolare i due bound e assicurare un valore della funzione di fitness che rispecchia la qualità della soluzione. L'algoritmo deve costruire una soluzione, ovvero un cromosoma rispettando determinati vincoli.

Per il calcolo del lower bound gli step sono i seguenti:

1. Creare una lista con tutte le operazioni che costituiscono il processo di assemblaggio;
2. Prendere la prima operazione dalla lista che richiede il livello più basso di abilità tra quelli richiesti dalle operazioni e inserirla nella prima stazione;
3. Se la skill richiesta da questa operazione coincide con la skill richiesta dall'operazione successiva all'interno della lista e la somma dei task time delle operazioni all'interno della stazione non supera il tempo ciclo, aggiungere l'operazione alla stazione ed eliminarla dalla lista, altrimenti procedere con lo step successivo;
4. Se la skill richiesta dall'operazione conincide con la skill richiesta dall'operazione successiva all'interno della lista ma la somma dei task time

delle operazioni supera il tempo ciclo allora creare una nuova stazione, inserire l'operazione e tornare allo step 2 altrimenti procedere con lo step 5;

5. Se la skill richiesta dall'operazione non coincide con la skill richiesta dall'operazione successiva all'interno della lista e la somma dei task time delle operazioni non supera il tempo ciclo, aggiungere l'operazione alla stazione, eliminarla dalla lista e ripetere lo step per tutte le operazioni della lista;
6. Se la lista è vuota stop;
7. Per ogni stazione prendere il valore della skill più elevato;
8. Sommare tutti i valori calcolati allo step 7 ottenendo il lower bound.

Per una maggiore comprensione di come vengano calcolati i lower bound, si riporta l'esempio del caso Hahn con tempo ciclo pari a 4676 [s] e configurazione D, che è stato presentato nel capitolo precedente in tabella 3.4. Utilizzando la nuova metodologia, l'algoritmo alloca le operazioni alle stazioni raggruppandole per skill richiesta in modo che non superino il tempo ciclo, quando ciò accade, l'algoritmo comincia a riempire una nuova stazione. Utilizzando la nuova metodologia, il valore del lower bound, senza considerare le precedenza tecnologiche, ottiene un livello di abilità pari a 5 con quattro stazioni, illustrato in tabella 4.2.

	N° operazione	Task time [s]	Skill	
stazione 1	**1**	970,98	0	**skill 0**
	4	141,96	0	
	7	99	0	
	10	180	0	
	13	111,96	0	
	16	237,96	0	
	19	600,96	0	
	22	69,96	0	
	25	105	0	
	28	69	0	
	31	69,96	0	
	34	69,96	0	
	37	124,98	0	
	40	127,98	0	
	43	90,96	0	
	46	486,96	0	
	49	79,98	0	

	52	741,96	0	
stazione 2	**2**	141,96	1	**skill 1**
	5	102,96	1	
	8	1206,96	1	
	11	81,96	1	
	14	420	1	
	17	258,96	1	
	20	79,98	1	
	23	88,98	1	
	26	330	1	
	29	99	1	
	32	157,98	1	
	35	52,98	1	
	38	352,98	1	
	41	64,98	1	
	44	90,96	1	
	47	138	1	
	50	64,98	1	
stazione 3	**53**	1084,98	1	**skill 2**
	3	141,96	2	
	6	96	2	
	9	159,96	2	
	12	60	2	
	15	1555,98	2	
	18	124,98	2	
	21	79,98	2	
	24	88,98	2	
	27	132	2	
	30	69,96	2	
	33	190,98	2	
	36	49,98	2	
	39	69,96	2	
stazione 4	**42**	1774,98	2	**skill 2**
	45	112,98	2	
	48	79,98	2	
	51	39,96	2	

Tabella 4.2 Lower bound delle skill con la nuova modalità

Tale procedura è la medesima per trovare il valore dell'upper bound, eccetto il confronto compiuto per scegliere di inserire o meno un'operazione all'interno della stazione, infatti solo se il valore della skill richiesta è diverso viene aggiunta alla stazione. Gli step sono illustrati di seguito:

1. Creare una lista con tutte le operazioni che costituiscono il processo di assemblaggio;
2. Prendere la prima operazione dalla lista che richiede il livello più elevato di abilità tra quelli richiesti dalle operazioni e inserirla nella prima stazione;
3. Se la skill richiesta da questa operazione differisce dalla skill richiesta dall'operazione successiva all'interno della lista e la somma dei task time delle operazioni all'interno della stazione non supera il tempo ciclo, aggiungere l'operazione alla stazione ed eliminarla dalla lista, altrimenti procedere con lo step successivo;
4. Se la skill richiesta dall'operazione differisce dalla skill richiesta dall'operazione successiva all'interno della lista ma la somma dei task time delle operazioni supera il tempo ciclo allora creare una nuova stazione, inserirel'operazione e tornare allo step 2 altrimenti procedere con lo step 5;
5. Se la skill richiesta dall'operazione coincide con la skill richiesta dall'operazione successiva all'interno della lista e la somma dei task time delle operazioni non supera il tempo ciclo, aggiungere l'operazione alla stazione, eliminarla dalla lista e ripetere lo step per tutte le operazioni della lista;
6. Se la lista è vuota stop;
7. Per ogni stazione prendere il valore della skill più elevato;
8. Sommare tutti i valori calcolati allo step 7 ottenendo l'upper bound.

Anche qui, per facilitare la comprensione del calcolo dell'upper bound, si riporta l'esempio del caso Hahn con tempo ciclo pari a 4676 [s] e configurazione D. I risultati sono riportati nella tabella 4.3. Utilizzando la nuova metodologia, l'algoritmo, per calcolare l'upper bound, inserisce le operazioni all'interno della stazione solo se richiedono una skill diversa rispetto all'operazione precedentemente introdotta nella stazione. Il valore che massimizza le skill richieste, nel caso proposto e senza considerare le precedenza tecnologiche, ottiene un livello di abilità pari a 8 con quattro stazioni.

	N° operazione	Task time [s]	Skill	
stazione 1	**1**	970,98	0	skill 2
	2	141,96	1	
	3	141,96	2	
	4	141,96	0	
	5	102,96	1	
	6	96	2	
	7	99	0	
	8	1206,96	1	
	9	159,96	2	
	10	180	0	
	11	81,96	1	
	12	60	2	
	13	111,96	0	
stazione 2	**14**	420	1	skill 2
	15	1555,98	2	
	16	237,96	0	
	17	258,96	1	
	18	124,98	2	
	19	600,96	0	
	20	79,98	1	
	21	79,98	2	
	22	69,96	0	
	23	88,98	1	
	24	88,98	2	
	25	105	0	
	26	330	1	
	27	132	2	
	28	69	0	
	29	99	1	
	30	69,96	2	
	31	69,96	0	
	32	157,98	1	
stazione 3	**33**	190,98	2	skill 2
	34	69,96	0	
	35	52,98	1	
	36	49,98	2	
	37	124,98	0	
	38	352,98	1	
	39	69,96	2	
	40	127,98	0	
	41	64,98	1	
	42	1774,98	2	

	43	90,96	0	
	44	90,96	1	
	45	112,98	2	
stazione 4	**46**	486,96	0	skill 2
	47	138	1	
	48	79,98	2	
	49	79,98	0	
	50	64,98	1	
	51	39,96	2	
	52	741,96	0	
	53	1084,98	1	

Tabella 4.3 Upper bound delle skill richieste con la nuova modalità

4.1.3 Bound per gli equipment richiesti

Alcune operazioni per essere compiute necessitano di un equipment o attrezzatura.

Lo strumento GenIAL ammette fino a 10 differenti tipologie di tools, codificati in Matlab con numeri dall'uno al dieci.

Come per le skill, anche i bound dell'insieme degli equipment richiesti tra le stazioni presentano alcune incertezze illustrate nel capitolo precedente.

A risoluzione di tali problemi, viene introdotta la medesima procedura utilizzata per i bound delle skill, la differenza risiede nell'oggetto del confronto che non è il livello di abilità ma l'equipment richiesto dall'operazione per essere processata.

Il procedimento, per il lower bound, è di seguito illustrato:

1. Creare una lista con tutte le operazioni che costituiscono il processo di assemblaggio;
2. Prendere la prima operazione dalla lista che richiede l'equipment con valore più basso possibile e inserirla nella prima stazione;
3. Se l'equipment richiesto da questa operazione coincide con l'equipment richiesto dall'operazione successiva all'interno della lista e la somma dei task time delle operazioni all'interno della stazione non supera il tempo ciclo,

aggiungere l'operazione alla stazione ed eliminarla dalla lista, altrimenti procedere con lo step successivo;

4. Se l'equipment richiesto da questa operazione coincide con l'equipment richiesto dall'operazione successiva all'interno della lista ma la somma dei task time delle operazioni supera il tempo ciclo allora creare una nuova stazione, inserire l'operazione e tornare allo step 2 altrimenti procedere con lo step 5;
5. Se l'equipment richiesto dall'operazione non coincide con l'equipment richiesto dall'operazione successiva all'interno della lista e la somma dei task time delle operazioni non supera il tempo ciclo, aggiungere l'operazione alla stazione, eliminarla dalla lista e ripetere lo step per tutte le operazioni della lista;
6. Se la lista è vuota stop;
7. Per ogni stazione prendere il valore degli equipment senza ripetizioni;
8. Sommare tutti i valori calcolati allo step 7 ottenendo il lower bound degli equipment, LB_E.

In questo modo si assicura un valore del lower bound maggiore rispetto al caso precedente e più veriterio in funzione del caso considerato.

Si riporta un esempio esplicativo del caso Lutz con tempo ciclo 2828 [s] e configurazione G illustrata nel capitolo precedente. In tabella 4.4, per ogni stazione riempita dall'algoritmo viene indicato il numero di equipment richiesti. L'algoritmo alloca le operazioni alle stazioni in funzione dell'equipment richiesto. Se la somma dei task time delle operazioni che richiedono lo stesso equipment supera il valore del tempo ciclo allora l'algoritmo inizia a riempira una nuova stazione.

	N° operazione	Task time [s]	Caso G	N° di equipment
stazione 1	1	457,98	1	1
	8	276	1	
	15	519,96	1	
	22	1399,98	1	
stazione 2	29	351,96	1	2
	2	195,96	2	

	9	213,96	2	
	16	456	2	
	23	645,96	2	
	30	511,98	2	
stazione 3	3	408	3	1
	10	261,96	3	
	17	543,96	3	
	24	201,96	3	
	31	457,98	3	
stazione 4	4	693,96	4	1
	11	615,96	4	
	18	678	4	
	25	327,96	4	
	32	324	4	
stazione 5	5	99,96	5	1
	12	351,96	5	
	19	543,96	5	
	26	499,98	5	
stazione 6	6	381,96	6	1
	13	858	6	
	20	387,96	6	
	27	351,96	6	
stazione 7	7	426	7	1
	14	237,96	7	
	21	258	7	
	28	195,96	7	

Tabella 4.4 Lower bound degli equipment calcolato con la modalità nuova

Per quanto concerne l'upper bound, UB_E, si è riscontrato, dalle validazioni precedenti di cui è possibile visionare i risultati al paragrafo 3.3.1, che il valore del limite superiore dell'insieme degli equipment richiesti in alcuni casi oltrepassa la peggiore delle ipotesi che si potrebbe presentare, in altri invece il valore dell'upper bound viene sottostimato.

Per risolvere tale problema, si è introdotta la stessa procedura dell'upper bound delle skill, ovviamente con i dovuti accorgimenti, e di seguito viene illustrata:

1. Creare una lista con tutte le operazioni che costituiscono il processo di assemblaggio;
2. Prendere la prima operazione dalla lista che richiede l'equipment con valore più basso possibile e inserirla nella prima stazione;

3. Se l'equipment richiesto da questa operazione è differente rispetto all'equipment richiesto dall'operazione successiva all'interno della lista e la somma dei task time delle operazioni all'interno della stazione non supera il tempo ciclo, aggiungere l'operazione alla stazione ed eliminarla dalla lista, altrimenti procedere con lo step successivo;
4. Se l'equipment richiesto da questa operazione è differente rispetto all'equipment richiesto dall'operazione successiva all'interno della lista ma la somma dei task time delle operazioni supera il tempo ciclo allora creare una nuova stazione, inserire l'operazione e tornare allo step 2 altrimenti procedere con lo step 5;
5. Se l'equipment richiesto dall'operazione coincide con l'equipment richiesto dall'operazione successiva all'interno della lista e la somma dei task time delle operazioni non supera il tempo ciclo, aggiungere l'operazione alla stazione, eliminarla dalla lista e ripetere lo step per tutte le operazioni della lista;
6. Se la lista è vuota stop;
7. Per ogni stazione prendere il valore degli equipment senza ripetizioni;
8. Sommare tutti i valori calcolati allo step 7 ottenendo l'upper bound degli equipment, UB_E.

Di seguito, nella tabella 4.5, si riportano i risultati ottenuti dall'upper bound con la nuova metodologia e inerenti la configurazione C del caso Lutz1 illustrata nel paragrafo 3.3.1 del capitolo precedente.

L'algoritmo, nella peggiore delle ipotesi, ottiene un numero di equipment pari a 17.

	N° operazione	Task time [s]	Caso C	N° equipment
stazione 1	1	457,98	1	3
	2	276	2	
	3	519,96	3	
	4	1399,98	1	
	21	99,96	3	
stazione 2	5	351,96	2	3
	6	195,96	3	
	7	213,96	1	
	8	456	2	

	9	645,96	3	
	10	511,98	1	
	11	408	2	
stazione 3	12	261,96	3	3
	13	543,96	1	
	14	201,96	2	
	15	457,98	3	
	16	693,96	1	
	17	615,96	2	
stazione 4	18	678	3	3
	19	327,96	1	
	20	324	2	
	22	351,96	1	
	23	543,96	2	
	24	499,98	3	
stazione 5	25	381,96	1	3
	26	858	2	
	27	387,96	3	
	28	351,96	1	
	29	426	2	
	30	237,96	3	
stazione 6	31	258	1	2
	32	195,96	2	

Tabella 4.5 Upper bound degli equipment calcolato con la nuova modalità

Con tale modalità di calcolo, l'algoritmo inserisce le operazioni nelle stazioni solo se l'equipment richiesto dall'operazione considerata è differente da quello delle operazioni inserite nella stazione e soltanto se la somma dei task time di tutte le operazioni in quella stazione non supera il tempo ciclo.

Se all'interno della stazione sono stati inseriti tutti i possibili equipment, allora l'algoritmo rianalizza tutte le operazioni per capire se è possibile aggiungere altre operazioni alla stazione senza superare il tempo ciclo. Quest'ultimo passaggio è comprensibile leggendo le operazioni allocate alla stazione 1 in cui è stata inserita l'operazione numero 21, la prima dopo la quattro che poteva essere aggiunta senza superare il tempo ciclo.

4.1.4 Upper Bound per la workload variance

La workload variance studia le variazioni di carico tra le stazioni, ovvero la differenza che si presenta tra il workstation time delle stazioni e il tempo ciclo, maggiore è tale differenza peggiore è la soluzione trovata.

Nella migliore situazione possibile, per ogni stazione, il workstation time coincide con il tempo ciclo, ragion per cui il lower bound di tale insieme è posto pari a zero.

La difficoltà inerente l'ottimizzazione della workload variance, è la sovrastima dell'upper bound che risulta molto superiore rispetto al valore che ottiene l'algoritmo; causando sovrastime anche nella funzione di fitness.

L'algoritmo GenIAL calcola l'upper bound della workload variance prendendo a riferimento l'upper bound del numero di stazioni, riempe le stazioni fino a quando il workstation time non raggiunge il massimo valore possibile, vicino al tempo ciclo, e alle rimanenti stazioni, senza nessuna operazione allocata, viene assegnato workstation time pari a zero. L'upper bound infine viene calcolato secondo la seguente formulazione:

$$UB_{WV} = \frac{intinf * (T_{am})^2}{UB_N} + \frac{(T_a - T_{am})^2}{UB_N} + (UB_N - intinf - 1) * \frac{(T_c - T_{am})^2}{UB_N}$$

Dove:

- $intinf$ = $\lfloor \frac{T_p}{T_c} \rfloor$ ovvero l'estremo inferiore del rapporto tra il tempo di produzione e il tempo ciclo;
- $T_a = T_c - T_s \; in \; cui \; T_s = T_p - intinf * T_c$;
- $T_{am} = \frac{(T_a + (UB_N - intinf - 1) * T_c)}{UB\_N}$;
- $UB_N =$ upper bound del numero di stazioni.

Tale configurazione crea sovrastime dell'upper bound e quindi valori della funzione di fitness elevati rispetto alla qualità della soluzione.

La modifica introdotta riguarda la modalità con cui l'algoritmo calcola l'upper bound. L'UB_WV è sempre connesso con l'UB_N però l'allocazione sulle stazioni viene eseguita random su tutte le stazioni. Nel dettaglio, gli step sono i seguenti:

1. Creare due liste, listaA una con tutte le operazioni e listaB con tutte le stazioni, pari a UB_N;
2. Prendere la prima operazione, inserirla nella prima stazione della listaB ed eliminarla dalla listaA;
3. Prendere la prima operazione della listaA, inserirla nella stazione successiva ed eliminarla dalla lista;
4. Ripetere lo step 3 fino a quando la stazione è l'ultima della lista;
5. Ripetere gli step dal 2 al 4 fino a quando la listaA non è vuota e quindi stop;
6. Calcolare di ogni stazione il workstation time e trovare il valore del lower bound.

In questo modo, il valore dell'UB_WV risulta inferiore rispetto ai valori calcolati con le modalità precedenti, nonostante gli UB_N calcolati con le nuove modalità siano superiori ai valori presentati dall'algoritmo senza modifiche.

Si prenda a riferimento il caso Gunther con tempo ciclo 44 [s]. In questo caso, l'upper bound del numero di stazioni, con le precedenti modalità era 14 mentre con le modalità introdotte diventa 16. Si illustra, in tabella 4.6, come vengono allocate le operazioni nell'ipotesi peggiore di workload variance.

N° operazione	Task time [s]	UB_WV new
1	28,98	stazione 1
17	1,98	
32	0,96	
2	3	stazione 2
18	1,98	
24	22,98	
3	4,98	stazione 3
19	18,96	
34	1,98	
4	21,96	stazione 4
5	6	stazione 5
20	28,98	
21	6	

35	1,98	
6	13,98	stazione 6
22	9,96	
7	1,98	stazione 7
9	21,96	
23	15,96	
8	4,98	stazione 8
24	22,98	
33	39,96	stazione 9
10	30	stazione 10
25	4,98	
11	22,98	stazione 11
26	4,98	
12	30	stazione 12
27	4,98	
13	22,98	stazione 13
14	1,98	stazione 14
28	39,96	
29	1,98	
15	18,96	stazione 15
30	4,98	
16	28,98	stazione 16
31	4,98	

Tabella 4.6 Upper bound della workload variance calcolato con la nuova modalità

Le operazioni vengono allocate alle stazioni in modo che tutte le stazioni massime che si possono riempire abbiano almeno una operazione da processare.

Secondo la letteratura, raggiungere un buon valore della workload variance, vicino allo zero, permette anche di ottenere una minimizzazione del numero di stazioni poiché si tende a raggruppare le operazioni il più possibile per cercare di riempire le stazioni fino al massimo del tempo disponibile. Se, invece, si vuole avere il caso peggiore allora è necessario riempire tutte le stazioni a disposizione senza pensare al tempo ciclo. Ottenere il valore massimo per il numero di stazioni è significato di raggiungere un pessimo risultato per la workload variance.

Si è modificata la modalità di calcolo dell'upper bound della workload variance in modo che il valore ottenuto risulti più inerente alla realtà. Inoltre, essendo il valore dell'upper bound inferiore a quello precedente, si ottiene una funzione di fitness più rappresentativa della soluzione trovata.

Infatti, nel caso Gunther di cui sopra, l'UB_WV con la modalità precedente risulta pari a 0.0899 mentre l'UB_WV con la nuova modalità è pari a 0.0095. La soluzione dell'algoritmo rimane la medesima, con un cromosoma con workload variance di valore 0.0047. In breve, la funziona di fitness varia, passando da 0.9354 a 0.5052, una soluzione non troppo buona.

Per avere una visione dettagliata sui risultati delle modifiche introdotte all'algoritmo per la risoluzione di problemi mono-obiettivo si rimanda al prossimo paragrafo.

4.2 Implementazione e risultati per i problemi mono-obiettivo

Il primo ostacolo da affrontare per i problemi mono-obiettivo, riguarda l'incapacità della funzione di fitness di rivelare la reale qualità di una soluzione. Tale discrepanza è dovuta ad una errata valutazione dei bound dei singoli obiettivi.

Nel grafico di figura 4.1, si illustra come la funzione di fitness, della soluzione trovata dall'algoritmo, varia con l'introduzione delle modifiche dei bound del numero di stazioni. I risultati proposti si riferiscono al caso Warnecke in cui i lower bound assumono valori compresi tra le 14 e le 29 stazioni. in blu si rappresenta il valore della funzione di fitness che si otteneva precedentemente, FN, mentre in rosso il valore della funzione di fitness determinata con gli upper bound calcolati con la nuova modalità, FN_new.

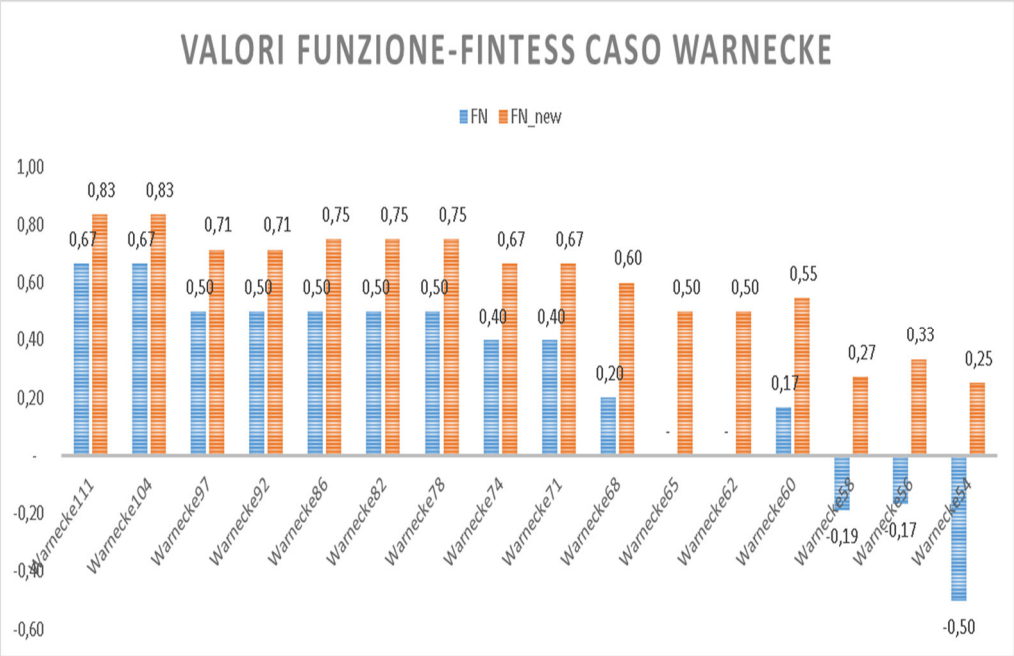

Tabella 4.1 Valori funzione di fitness caso Warnecke

I valori della "nuova" funzione di fitness sono superiori rispetto a quelli precedenti. Nei primi casi, FN_new assume valori superiori, più vicini alla soluzione ottima pari ad uno; riflettendo con correttezza il numero di stazioni trovato dall'algoritmo, che si discosta dall'ottimo di una o due unità.

Negli ultimi casi, in cui il valore della funzione di fitness risultava negativo, FN_new aumenta leggermente portandosi a volori positivi, sempre bassi perché il numero di stazioni continua a superare il lower bound di 7 o 9 unità.

Nel caso Tonge, illustrato in figura 4.2, FN_new aumenta in tutte le istanze testate.

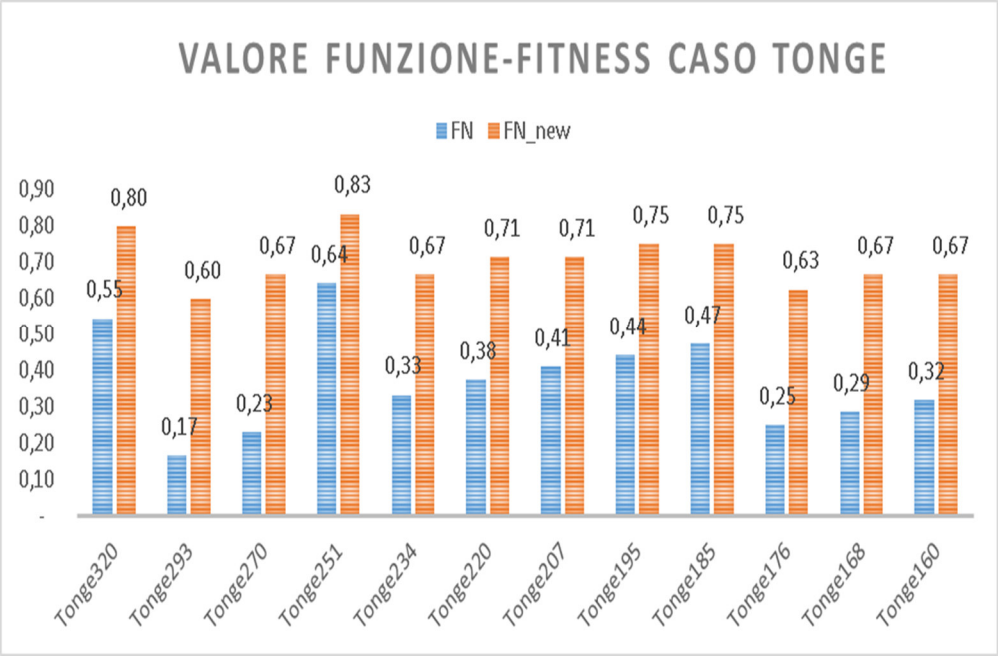

Figura 4.2 Valori funzione-fitness caso Tonge

Ad esempio, nella prima e quarta istanza, come riportato in tabella 4.7, il valore della funzione di fintess si aggirava intorno allo 0.6 nonostante le soluzioni trovate dall'algoritmo, N, fossero molto vicine al lower bound. Con la nuova modalità di calcolo dell'upper bound, UB_new, il valore della funzione di fitness, FN_new, indica con maggiore veridicità la qualità della soluzione trovata dall'algoritmo. Il valore delle stazioni ottenuto con le nuove modalità di calcolo dell'upper bound non è stato riportato poiché non varia.

Caso di studio	LB	UB	N	FN	UB_new	N_new	FN_new
Tonge320	11	14	12	0.55	16	12	0.80
Tonge251	14	17	15	0.64	20	15	0.83

Tabella 4.7 Valori del numero di stazioni e il valore di fitness per due casi Tonge

Con le modifiche apportate, l'utente, leggendo il valore della funzione di fitness, può capire con maggiore chiarezza la qualità della soluzione trovata dall'algoritmo che precedentemente era sottostimata.

Negli altri problemi in cui si perseguono rispettivamente la minimizzazione delle abilità richieste, la minimizzazione degli equipment e la minimizzazione della workload variance si ha la situazione opposta, la funzione di fintess viene sovrastimata. Infatti con le modifiche apportate ai bound dei tre insiemi, il valore della funzione di fintess che si ottiene è inferiore rispetto ai precedenti valori in quasi tutte le istanze.

Il lower bound delle skill assumeva sempre valori 0, 1 o 2, mentre con le modifiche apportate all'algoritmo, LB_S non assume mai valori di questo tipo eccetto nei casi seguenti:

- LB_S= 0 se tutte le operazioni non richiedono nessuna skill;
- LB_S=1 se una operazione richiede skill pari ad uno e tutte le altre zero;
- LB_S=2 se una operazione richiede skill pari a due e tutte le altre pari a zero.

L'upper bound invece risulta essere inferiore in tutti i casi rispetto alle modalità di calcolo precedenti. In questo modo, l'insieme delle skill richiesto diventa più ristretto, e questa restrizione incide sul valore della funzione di fitness evidenziato nell'istogramma in figura 4.3.

Il valore FS_new rappresenta i valori della funzione di fitness ottenuti con le nuove modalità di calcolo dei bound e tali valori vengono confrontati con quelli precedenti della funzione di fintess, FS, indicati in blu.

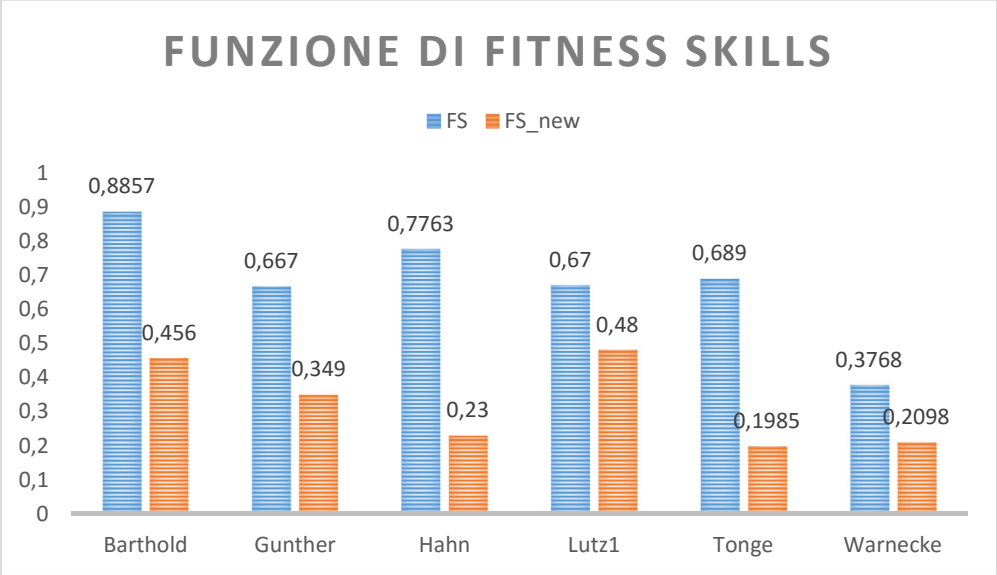

Figura 4.3 Valori funzione-fitness per la minimizzazione delle skills

In media, si comprende che la soluzione che l'algoritmo individuava non era molto buona.

Per comprendere perché ciò accade, in tabella 4.8, sono rappresentati alcuni esempi. I test sono stati compiuti con la configurazione D utilizzata nella tabella del capitolo precedente. La configurazione alternava i valori 0,1,2 in modo da avere tutte e tre le tipologie di skill e in media con lo stesso quantitativo. Il valore delle skill trovato con le nuove modalità di calcolo dei bound non viene indicato poiché rimane invariato.

Caso di studio	LB	UB	Skills	Fitness	LB_new	UB_new	Fitness_new
Barthold805	2	58	14	0.7857	10	16	0.333
Gunther81	2	20	10	0.556	9	14	0.8
Hahn4676	2	21	7	0.7386	5	8	0.333
Lutz12828	2	16	9	0.50	6	12	0.50
Tonge320	2	32	14	0.60	8	14	0
Warnecke111	2	38	24	0.3889	14	28	0.2143

Tabella 4.8 Valori dei bounds e della funzione di fitness per la minimizzazione delle skills

Nel caso Barthold con tempo ciclo 805 [s], il lower bound che si otteneva era due mentre l'upper bound era di 58 skills, il risultato trovato dall'algoritmo coincideva con il valore 14 e la funzione di fintess uguale a 0.7857. Con le modifiche introdotte, il lower bound aumenta mentre l'upper bound diminuisce, la funzione di fintness diminuisce poiché il numeratore aumenta, il denominatore si riduce quindi il valore

che viene sottratto all'uno è minore. Si ricorda infatti che la funzione obiettivo normalizzata viene calcolata come $F_i = 1 + \frac{(LB-x)}{(UB-LB)}$, nel dettaglio $FS = 1 + \frac{(2-14)}{(58-2)} = 1 + \frac{-12}{56} = 0.7857$ mentre

$FS_{new} = 1 + \frac{(10-14)}{(16-10)} = 1 + \frac{-4}{6} = 0.333.$

Passando alla minimizzazione degli equipment, il lower bound assumeva un valore massimo pari a 10. Con le modifiche apportate, in particolar modo al lower bound, l'estremo inferiore di questo insieme ottiene valori uguali a quelli precedenti solo nei seguenti casi:

- LB_E= 1 se tutte le stazioni non richiedono nessun equipment tranne una che per processare le operazioni che gli sono state assegnate ne richiede uno;
- LB_E= n, in cui n Є [2,10], se tutte le stazioni richiedono in totale n equipment oppure solo n stazioni richiedono ciascuna un equipment.

A titolo di esempio per LB_E= 3 si illustrano in tabella 4.9, tre configurazioni utilizzate per il caso Gunther con tempo ciclo 81 [s].

Con la configurazione A, l'agoritmo posiziona i tre equipment richiesti nella stessa stazione in modo che il totale degli equipment richiesti è tre, LB=3. Nel caso B invece si nota che solo tre operazioni richiedono ciascuna un equipment che tra l'altro è lo stesso, in questo caso specifico il lower bound è pari a uno proprio perché tende ad inserire le operazioni nella stessa stazione. Nella configurazione C invece, molte stazioni richiedono lo stesso equipment, in questo caso il lower bound è pari a tre perché l'algoritmo riesce a riempire fino a tre stazioni con quelle operazioni.

N° operazione	Task time [s]	Caso A	Caso B	Caso C
1	28,98	0	0	0
2	3	0	0	4
3	4,98	0	4	4
4	21,96	0	0	0
5	6	0	0	0
6	13,98	0	0	4
7	1,98	0	0	4
8	4,98	0	0	4
9	21,96	0	0	0
10	30	1	0	0
11	22,98	0	0	0
12	30	0	4	4
13	22,98	0	0	4
14	1,98	0	0	4
15	18,96	0	0	4
16	28,98	3	4	4
17	1,98	0	0	4
18	1,98	0	0	0
19	18,96	4	0	0
20	28,98	0	0	0
21	6	0	0	0
22	9,96	0	0	0
23	15,96	0	0	0
24	22,98	0	0	0
25	4,98	0	0	0
26	4,98	0	0	4
27	4,98	0	0	4
28	39,96	0	0	4
29	1,98	0	0	4
30	4,98	0	0	4
31	4,98	0	0	4
32	0,96	0	0	4
33	39,96	0	0	4
34	1,98	0	0	4
35	1,98	0	0	4

Tabella 4.9 Configurazioni degli equipment

L'upper bound degli equipment invece nelle configurazioni precedenti, risulta assumere valori inferiori rispetto agli elevati valori ottenuti con le modalità precedenti. Con la configurazione A, il valore dell'upper bound è pari a 3 perché tende a posizionare gli equipment in tre stazioni differenti. Con la configurazione B, il valore dell'upper bound è 3 perché posiziona le tre operazioni in tre stazioni differenti e infine

nella configurazione C l'upper bound è pari a 6 poiché riempe le stazioni fino al massimo valore possibile, dato dal tempo ciclo pari a 81 [s], inserendo in ognuna almeno un'operazione che richiede l'equipment. In precedenza invece, con la configurazione A si otteneva un LB = 2 e un UB = 53, con la configurazione B invece LB = 1 e UB = 72 e infine con la configurazione C LB = 1 e UB = 91.

Di conseguenza, si ottengono valori dei bound molto differenti dai precedenti valori. Ciò si traduce in un valore della funzione di fitness minore rispetto a quello che si otteneva precedentemente. Nella figura 4.4, si mostrano i valori che si ottengono in media per i vari casi di studio. In rosso i valori della funzione di fitness degli equipment che si ottiene con le nuove modalità di calcolo dei bound, mentre in blu i valori della fitness che si ottenevano in media precedentemente.

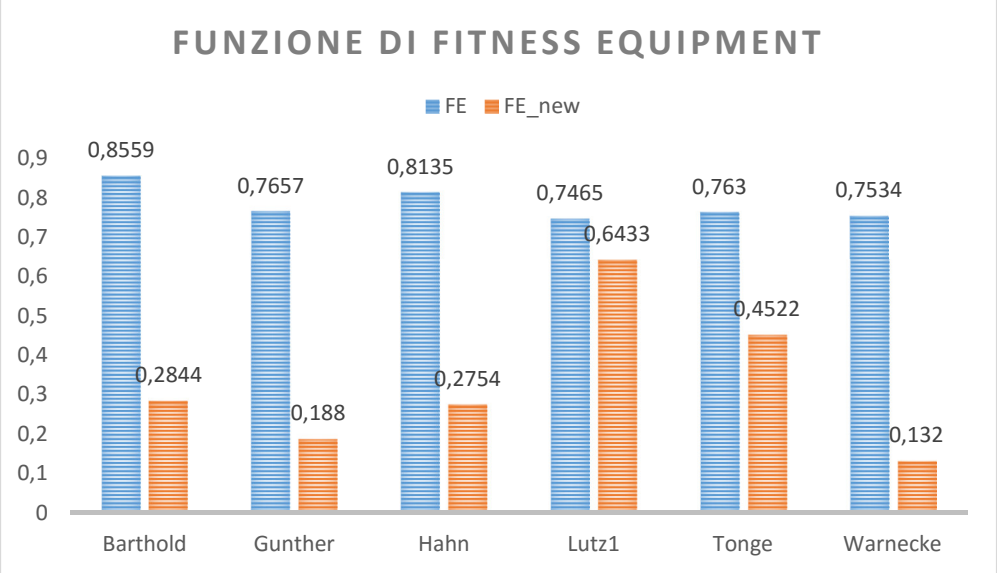

Figura 4.4 Valori della funzione di fitness della minimizzazione degli equipment

Anche nel caso della minimizzazione degli equipment, le soluzioni trovate dall'algoritmo non erano tanto buone quanto indicava la funzione di fitness.

Infine, si mostrano, in figura 4.5, i risultati ottenuti per la minimizzazione della workload variance di sei casi: Barthold con tempo ciclo di 805 [s], Gunther con tempo ciclo 81 [s], Hahn con 4676 [s], Lutz1 con tempo ciclo 2828 [s], Tonge con 527 [s] e

Warnecke con 111 [s]. Anche in questo caso la funzione di fitness, trovata con le nuova modalità di calcolo dell'upper bound, FWV_new, risulta diminuita rispetto ai valori precedenti, FWV in blu.

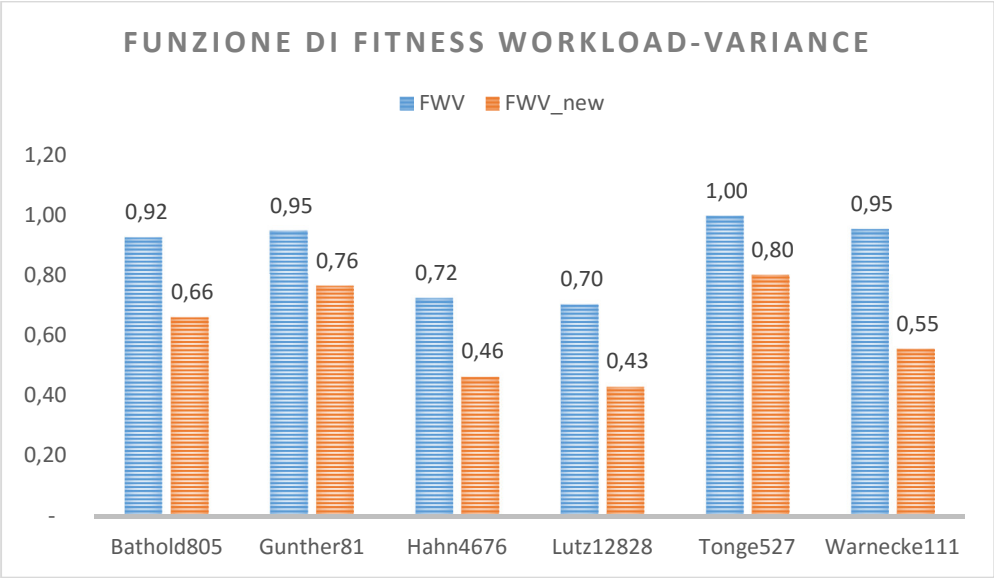

Figura 4.5 Valori funzione di fitness della minimizzazione della workload variance

Nella maggior parte dei casi proposti si ha una visione migliore della qualità delle soluzioni trovate dall'algoritmo di cui la bontà viene valutata in modo più realistico.

Se si vogliono migliorare le soluzioni trovate dall'algoritmo bisogna andare più in profondità e capire nel dettaglio come GenIAL seleziona i cromosomi considerati migliori ad ogni iterazione dell'algoritmo.

4.3 Ulteriori miglioramenti

L'algoritmo genetico, come illustrato nel paragrafo 2.2.4, ad ogni iterazione applica alla popolazione di individui almeno due operatori, quello di crossover e quello di mutazione, per assicurare l'evoluzione ed il miglioramento di nuovi cromosomi figli. Dopo la riproduzione però, non tutti gli individui possono sopravvivere e continuare a riprodursi poiché coloro che non possiedono i geni migliori sono costretti a fermarsi. Ragion per cui, l'algoritmo impiega un ulteriore operatore alla sua popolazione, la cosiddetta selezione.

Nel dettaglio, l'algorimo GenIAL utilizza la *Roulette wheel selection* tecnica con la quale la probabilità che un cromosoma ha di essere selezionato è proporzionale alla sua fitness. In figura 4.6, si hanno 7 cromosomi, sette individui, ognuno con un determinato valore percentuale di fitness rispetto al totale. L'operatore di selezione funziona come l'indicatore di una roulette che ruota fino a fermarsi. Maggiore è la porzione di roulette a disposizione del cromosoma, maggiore sarà la possibilità che l'indicatore si fermi su di esso.

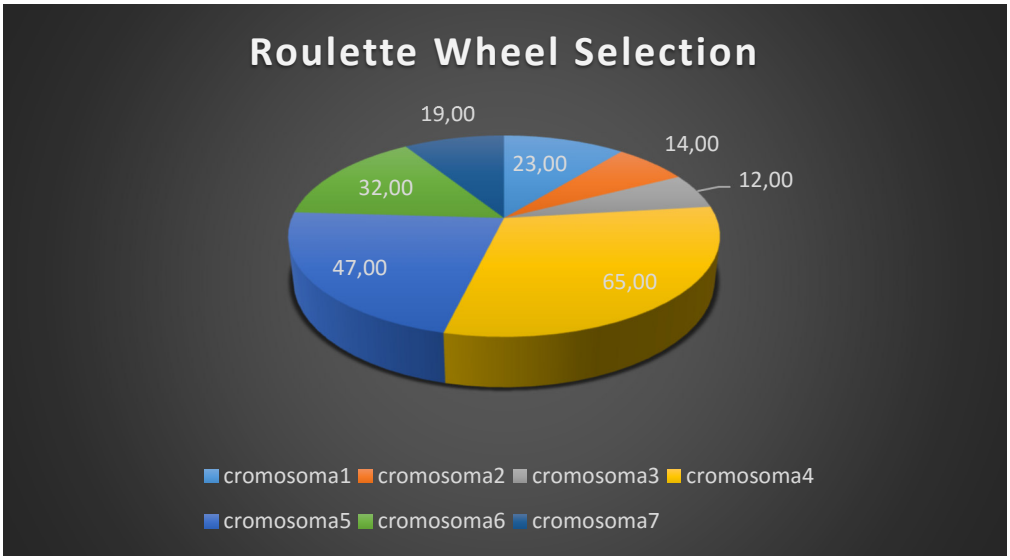

Figura 4.6 Roulette Wheel Selection

Questa tipologia di selezione però, mostra qualche difficoltà:

- la convergenza prematura, se un individuo ha fitness molto maggiore della media della popolazione ma molto minore della massima fitness possibile, tenderà ad essere sempre selezionato e quindi a generare una popolazione mediocre;
- la stagnazione, ovvero ciò che accade dopo un certo numero di generazioni, quando tutti gli individui hanno una buona fitness e quindi tendono ad avere la stessa probabilità di essere selezionati.

Infatti gli individui che, in una generazione sono considerati mediocri a livello di soluzione, sono invece i cromosomi che potrebbero avere un margine di miglioramento più elevato.

Lo spazio delle soluzioni a disposizione dell'algoritmo è molto vasto. Il problema principale è che l'algoritmo GenIAL tende a soffermarsi su una sola porzione di spazio quando trova un cromosoma con fintess molto superiore alla norma e quindi perlustra quell'interno fino alla condizione di stop senza ottene miglioramenti sostanziali nella soluzione.

Per risolvere tale problema si introduce un altro tipo di selezione che tende rendere la probabilità di selezione più omogenea su tutti i cromosomi, in modo tale che tutti abbiano maggiori possibilità di essere selezionati.

Con la nuova selezione, la probabilità di essere selezionato di ogni cromosoma è pari alla sua fitness moltiplicata per un fattore crescente al decrescere del valore della funzione di fitness. L'algoritmo ordina i cromosomi in ordine di fitness decrescente; il cromosoma con più grande e con più piccolo valore di fitness vengono trasportati alla nuova generazione di default. Per quanto concerne gli altri cromosomi, ad ognuno viene assegna una probabilità di essere selezionato. La probabilità di essere ogni cromosoma è calcolata come indicato nell'equazione (9):

$$PS_i = \frac{(Fitness_{i-1} - Fitness_{i+1})}{(Fitness_{max} - Fitness_{min})} \qquad \text{(eq. 9)}$$

In cui

- $Fitness_{max}$ = maggiore fitness all'interno della popolazione;
- $Fitness_{min}$ = minore fitness all'interno della popolazione;
- $Fitness_{i-1}$ = fitness del cromosoma precedente a i;
- $Fitness_{i+1}$ = fitness del cromosoma successivo a 1.

Trovati i valori della probabilità di selezione, l'algoritmo li assegna in maniere decrescente, ovvero la maggiore probabilità viene assegnata al cromosoma con minore fitness e di seguito per gli altri.

In tabella 4.10, sono riportate le probabilità di selezione con i relativi valori di fitness per i cromosomi di cui sopra.

Cromosoma	Fitness sul totale	Probabilità di selezione	Fitness_new
1	23	0.24	5.52
2	14	0.62	8.68
3	12	∞	∞
4	65	∞	∞
5	47	0.13	6.11
6	32	0.16	5.12
7	19	0.45	8.55

Tabella 4.10 Valutazione dei cromosomi con la Rank Selection

La fitness totale è data dalla fitness vecchia moltiplicata per il fattore di selezione assegnato. La situazione che si ha con la nuova selezione è illustrata nella figura 4.7, in cui i cromosomi hanno tutti più o meno la stessa probabilità di essere selezionati. Non vengono illustrati il cromosoma 3 e 4 poiché sono quelli presi di default poiché rappresentano la fitness peggiore e migliore.

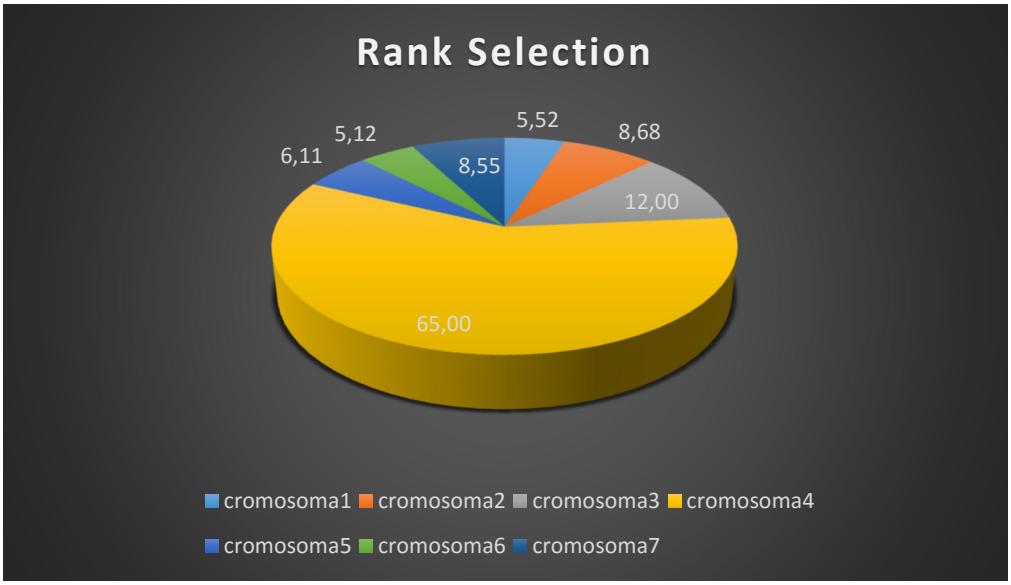

Figura 4.7 Rank selection

In questo modo si riescono ad ottenere miglioramenti nelle soluzioni individuate dall'algoritmo poiché GenIAL riesce a muoversi in punti dello spazio che prima non considerava. Tale valutazione della fitness, tale tipologia di selezione, viene utilizzata allo scopo di aumentare lo spazio perlustrato dall'algoritmo, favorendo la diversità all'interno della popolazione durante le iterazioni.

Durante l'ultima iterazione però, la selezione si basa solo sulla fitness e il fattore di selezione diventa uguale per tutti i cromosomi.

Nel caso del numero di stazioni, nel grafico di figura 4.8, si confronta la soluzione ottenuta con la nuova selezione sia con ciò che si otteneva prima sia con il valore dei dati Scholl. I casi considerati sono solo quelli in cui la soluzione ottenuta dall'algoritmo differiva da quella di Scholl.

La soluzione nuova coincide con la soluzione Scholl nella maggior parte dei casi e in alcune istanze, più critiche, vi è un notevole miglioramento rispetto al numero di stazioni precedenti.

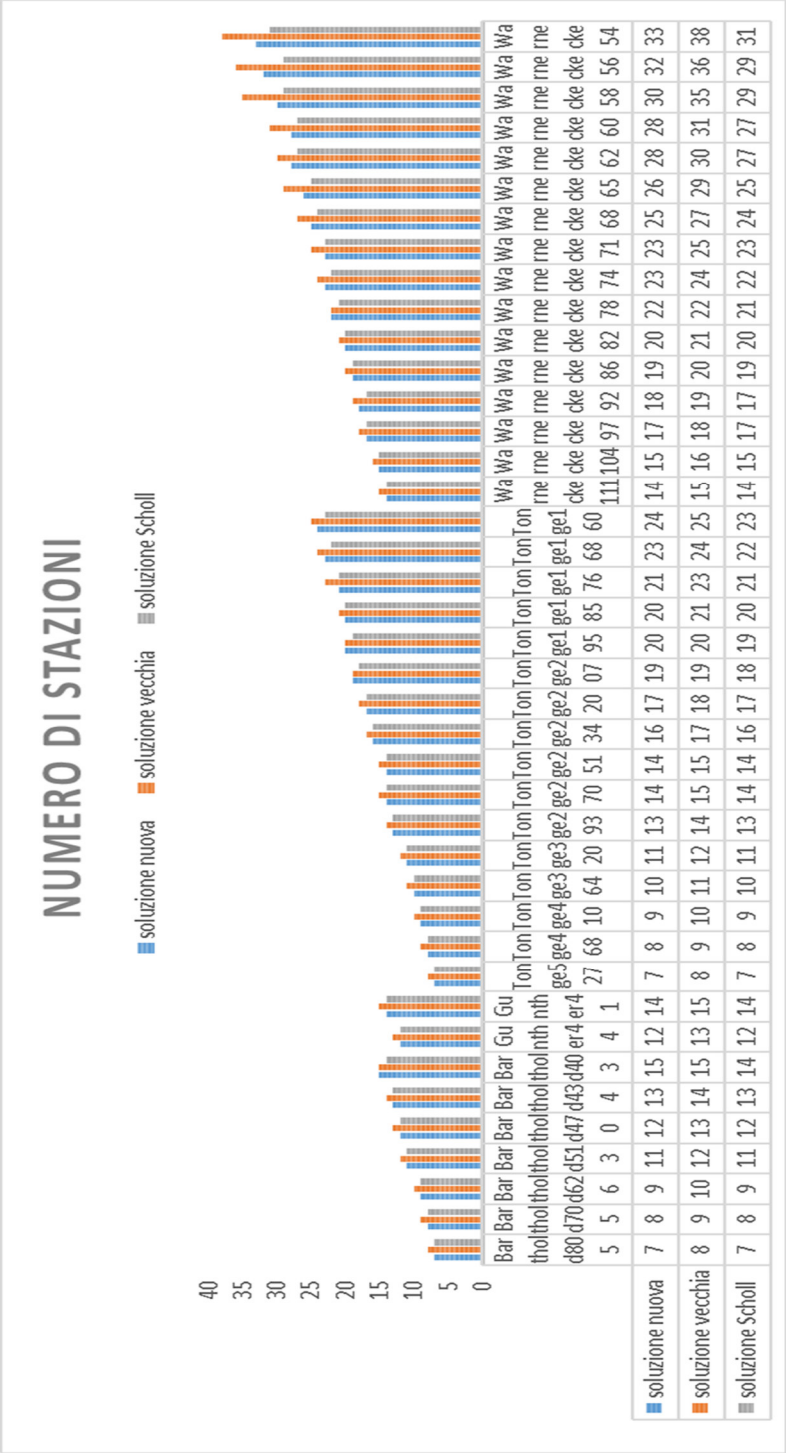

Figura 4.8 Numero di stazioni per i casi mono-obiettivo

Nel dettaglio si analizza una istanza per ogni caso di studio, i cui risultati sono indicati nella tabella 4.11. Il numero di stazioni trovato dall'algoritmo con la nuova tipologia di selezione è minore rispetto a quello precedente e con le nuove modalità di calcolo

dell'upper bound, anche la funzione di fitness rispecchia in maniera più adatta la bontà della soluzione stessa. Infatti nell'istanza Gunther41, Tonge60 e Warnecke60 il numero di stazioni trovato, N_new, si avvicina maggiormente al valore del lower bound, fino a coincidere con esso nell'istanza Barthold470.

Istanza	N	Fitness	N_new	Fitness_new
Barthold470	13	0.667	12	1
Gunther41	15	0	14	0.6
Tonge160	24	0.333	25	0.666
Warnecke60	31	-0.666	28	0.81

Tabella 4.11 Valori numero di stazioni e funzione di fitness casi mono-obiettivo

Un miglioramento lieve rispetto alle soluzioni individuate con la *roulette wheel selection*, si possono visualizzare anche nelle altre ottimizzazioni. Nel grafico in figura 4.9, sono riportati i risultati delle skill ottenuti dall'algoritmo con la nuova modalità di selezione, la rank selection in rosso, ponendo a zero tutti i pesi degli obiettivi tranne quello riguardante la minimizzazione delle skill. In blu sono indicati i valori che si ottenevano con la roulette wheel selection. Nel caso Barthold si riescono ad ottenere notevoli miglioramenti mentre negli altri casi di studio si notano miglioramenti meno marcati.

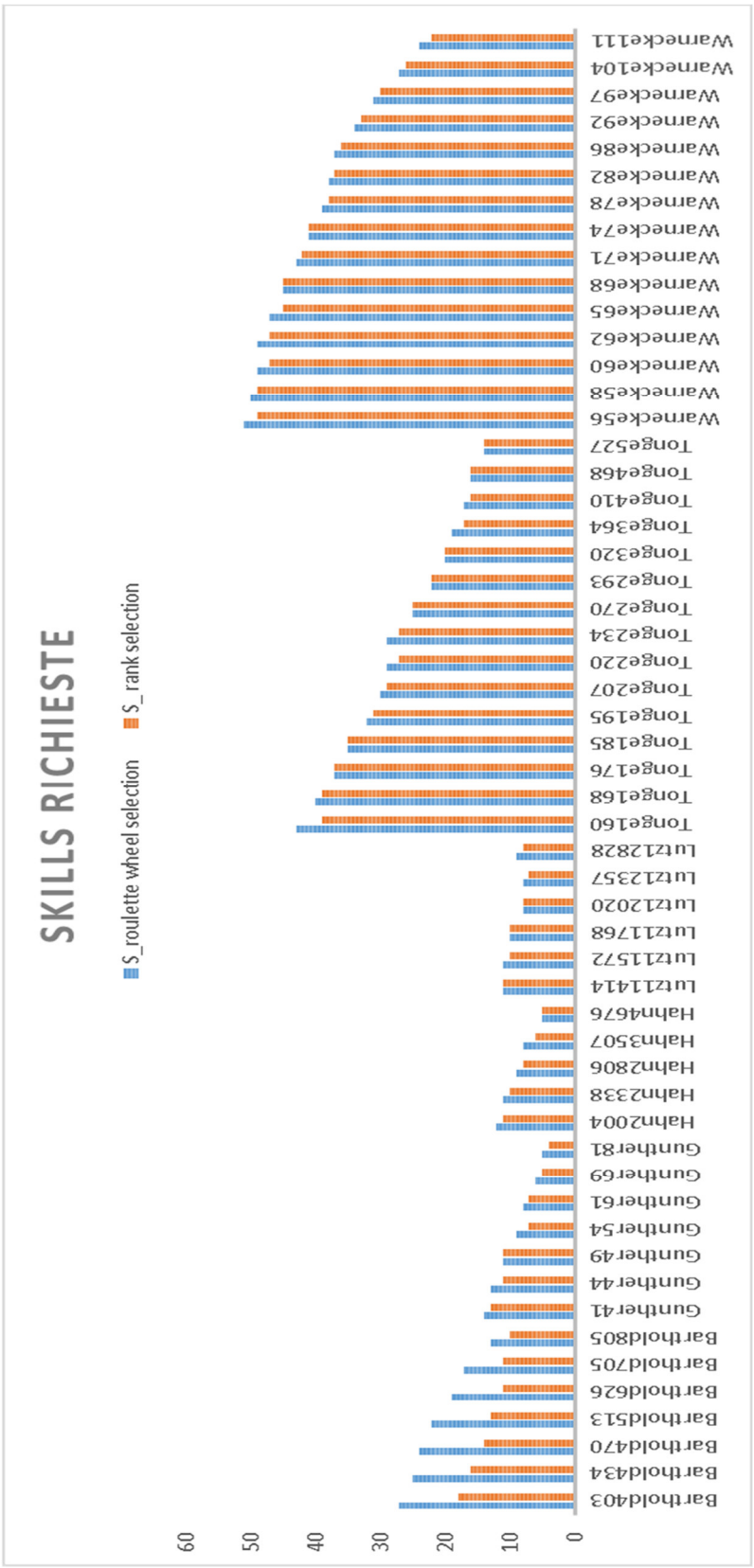

Figura 4.9 Valori delle skills dei casi mono-obiettivo ottenute con la rank selection e confrontate con i valori ottenuti con la roulette wheel selection

Anche la minimizzazione degli equipment presenta miglioramenti rispetto ai valori trovati con la modalità di selezione precedente. Nel grafico in figura 4.10, si riportano i risultati ottenuti con la *rank selection* confrontati con i valori ottenuti con la *roulette wheel selection*, in blu. Sulle ascisse sono riportati i casi di studio con i rispettivi tempi ciclo mentre sulle ordinate sono riportati i valori degli equipment totati che possiede la soluzione considerata ottima. Anche qui si nota un evidente miglioramento nelle istanze Barthold mentre un miglioramento meno marcato è presente anche negli altri casi di studio.

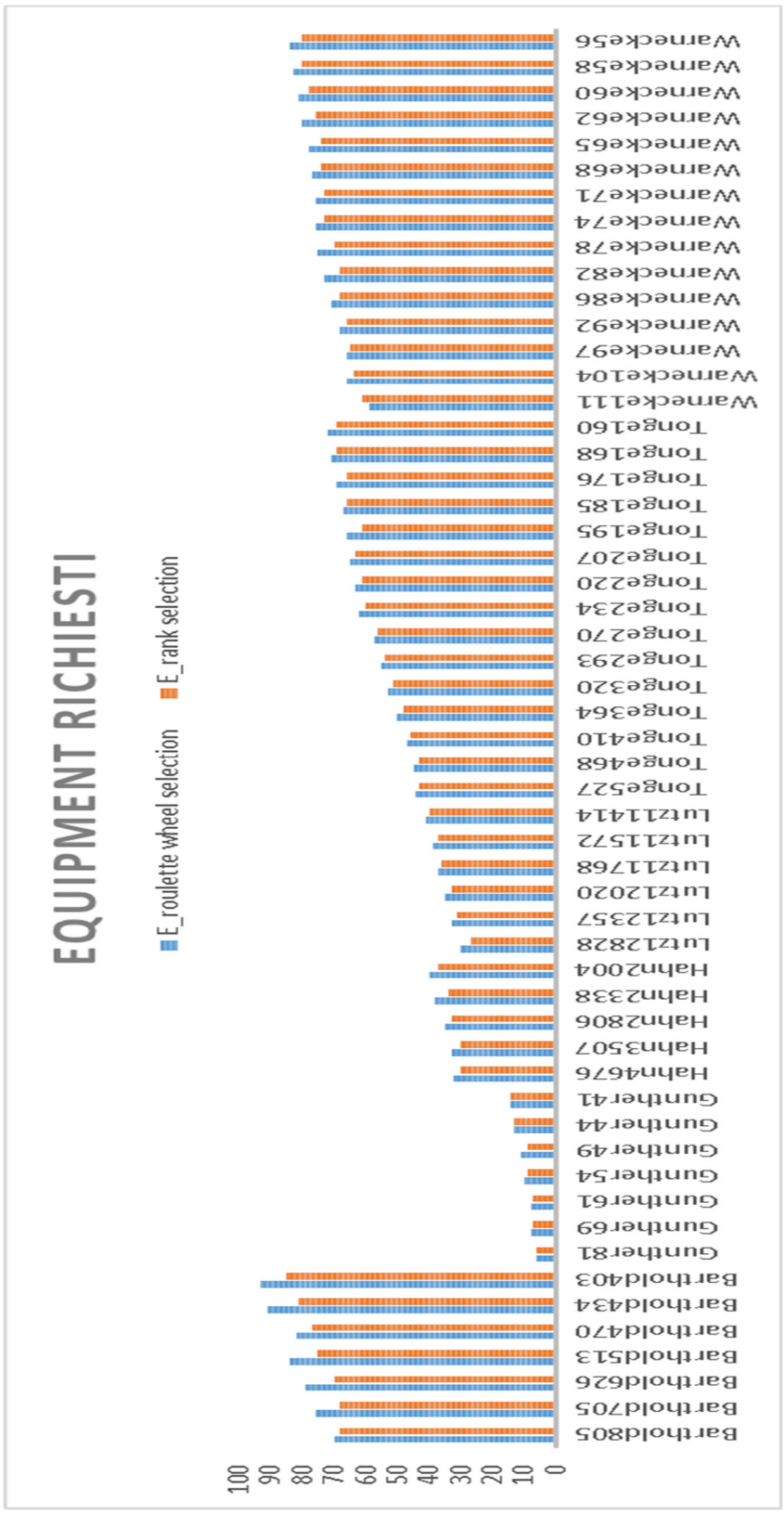

Figura 4.10 Valori degli equipment dei casi mono-obiettivo ottenute con la rank selection e confrontate con i valori ottenuti con la roulette wheel selection

Per quanto concerne la Workload-variance, anche in questo caso l'algoritmo GenIAL riesce ad individuare cromosomi migliori rispetto a quelli precedenti, ottenendo una minimizzazione della varianza del carico di lavoro maggiore rispetto a quella ottenuta con la precedente modalità di selezione.

I risultati raggiunti nelle diverse istanze, sono illustrate nel grafico in figura 4.11, in cui si riporta nelle ascisse il valore in secondi della workload variance mentre nelle ordinate si riportano i casi di studio con i rispettivi tempi ciclo.

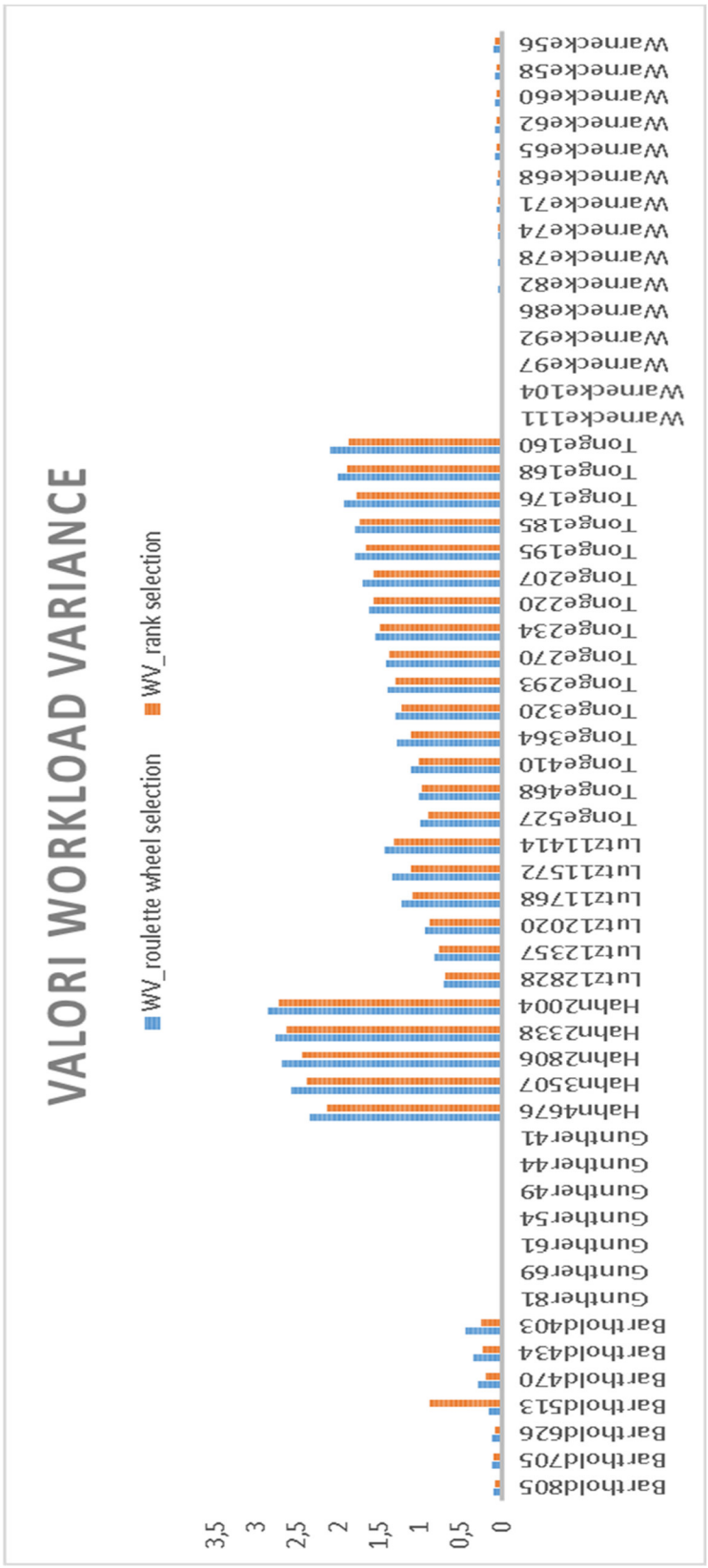

Figura 4.11 Valori della workload variance dei casi mono-obiettivo ottenute con la rank selection e confrontate con i valori ottenuti con la roulette wheel selection

Le soluzioni trovate dall'algoritmo nell'ottimizzazione dei singoli obiettivi sono migliorate rispetto a quando l'algoritmo utilizzava la *roulette wheel selection.*

Problema diverso si presenta quando l'algoritmo deve ottimizzare più obiettivi contemporaneamente, e viene affrontato nel prossimo paragrafo.

4.4 Miglioramenti proposti per i problemi multi-obiettivo

Prima di affrontare i miglioramenti proposti per i problemi multi-obiettivo si ricorda che in tali problemi di ottimizzazione risulta necessario trovare una soluzione che sia ottima contemporaneamente per i diversi obiettivi considerati e che spesso si trovano in competizione tra loro.

Quando si affronta questo tipo di problema, la soluzione ottimale ottenuta non è mai unica, ma esistono un insieme di soluzioni ugualmente ottimali rispetto al problema dato. È compito dell'operatore di selezione decidere quale delle soluzioni all'interno dell'insieme è più adatta in funzione delle esigenze.

Esistono diversi metodi per risolvere problemi di ottimizzazione multi-obiettivo, anche algoritmi genetici differenti se utilizzati per risolvere problemi mono-obiettivo o multi-obiettivo, quindi è necessario modificare l'operatore di selezione che si è introdotto precedentemente, quando l'obiettivo da ottimizzare era unico.

Prima di selezionare i cromosomi in base alla loro fitness, è auspicabile che l'algoritmo genetico GenIAL ordini i cromosomi in funzione del peso che è stato dato ai singoli obiettivi e successivamente ne calcoli la fitness.

Si è dato un nome a questa nuova tipologia di selezione *importance selection* i cui step sono espletati di seguito:

1. Creare una lista con tutti i pesi delle funzioni;
2. Prendere l'obiettivo che ha peso maggiore ed eliminare il peso dalla listaA;

3. Confrontare i cromosomi della generazione attuale tra loro in funzione del valore di tale obiettivo, quelli che hanno i valori maggiori inserirli nella lista1 ed eliminarli dalla lista dei cromosomi;
4. Prendere l'obiettivo che ha peso maggiore tra quelli rimasti nella liste ed eliminarlo dalla lista;
5. Confrontare i cromosomi della lista1 tra loro in funzione del valore di tale obiettivo, quelli che hanno i valori maggiori inserirli in una lista2 ed eliminarli dalla lista1;
6. Prendere l'obiettivo che ha peso maggiore tra quelli rimasti nella liste ed eliminarlo dalla lista;
7. Confrontare i cromosomi della lista2 tra loro in funzione del valore di tale obiettivo, quelli che hanno i valori maggiori inserirli in una lista3 ed eliminarli dalla lista2;
8. Prendere l'obiettivo che ha peso maggiore tra quelli rimasti nella liste ed eliminarlo dalla lista;
9. Confrontare i cromosomi della lista3 tra loro in funzione del valore di tale obiettivo, quelli che hanno i valori maggiori inserirli in una lista4 ed eliminarli dalla lista3;
10. Prendere l'obiettivo che ha peso maggiore tra quelli rimasti nella liste ed eliminarlo dalla lista;
11. Confrontare i cromosomi della lista4 tra loro in funzione del valore di tale obiettivo, quelli che hanno i valori maggiori inserirli in una lista5 ed eliminarli dalla lista4;
12. Rentegrare i pesi nella lista dei pesi tranne il peso maggiore;
13. Ripetere gli step dal 4 all'11 fino a quando le liste dalla 1 alla 4 non sono vuote;
14. Se la lista1, lista2, lista3 e lista4 sono vuote, reintegra tutti i pesi nella lista dei pesi e torna allo step 2;
15. Se non ci sono altri cromosomi della generazione vai allo step 16;
16. Assegna ad ogni cromosoma nella lista5 una probabilità di essere selezionata, *importance index*, decrescente e stop.

Successivamente l'algoritmo seleziona il cromosoma in base alla sua probabilità di selezione data dal prodotto della sua fitness per la probabilità assegnatagli dall'*importance index*.

Per comprendere tutti i passaggi svolti dall'i*mportance selection*, viene compiuto un esempio. Si supponga di assegnare agli obiettivi gli stessi pesi utilizzati per compiere la validazione multi-obiettivo ovvero:

- η_1 = 0.2;
- η_2 = 0.1;
- η_3 = 0.2;
- η_4 = 0.2;
- η_5 = 0.3.

Si supponga di avere 10 cromosomi alla generazione i-esima e che ognuno di essi abbia un valore per ogni obiettivo da raggiungere. I valori riportati nella tabella 4.12 devono essere considerati solo a livello esplicativo. Maggiore è il valore dell'obiettivo migliore è la soluzione del cromosoma considerato.

Cromosoma	Obiettivo 1	Obiettivo 2	Obiettivo 3	Obiettivo 4	Obiettivo 5
1	10	6	5	8	3
2	6	3	2	3	7
3	7	5	3	7	4
4	9	8	6	2	6
5	3	2	4	5	5
6	8	1	7	6	2
7	2	7	1	1	10
8	1	10	8	9	9
9	5	9	9	4	8

10	4	4	10	10	1

Tabella 4.12 Esempio cromosomi

Il primo step che compie l'algoritmo è il confronto dei cromosomi in funzione del valore obiettivo più importante ovvero il numero cinque.

Da questo primo confronto, i cromosomi che sopravvivono sono il cromosoma 2,4,7,8,9 i quali vengono confrontati a loro volta in funzione del valore che assume l'obiettivo secondo per importanza, la minimizzazione del numero di stazioni. Il cromosoma 2 ha un valore dell'obiettivo pari a sei che è minore rispetto a quello ottenuto dal cromosoma 4 perciò il cromosoma 2 viene scartato. Tale paragone viene compiuto per i cinque cromosomi portando ad ottenere un'altra lista di cromosomi formata dal 4 e dal 9. Questi a loro volta vengono paragonati in funzione del valore dell'obiettivo terzo per importanza, la minimizzazione degli equipment richiesti. Vengono compiuti tutti i confronti fino ad ottenere la lista finale costituita dal cromosoma numero 9.

L'algoritmo elimina il cromosoma 9 dalla lista dei cromosomi e ripete gli step fino a quando tutti cromosomi sono stati ordinati. In riferimento all'esempio di cui sopra, la lista ordinata dei cromosomi diventa la seguente: 9,8,2,7,4,10,5,3,1 e 6.

Trovato l'ordine dei cromosomi, viene assegnato ad ognuno una probabilità di selezione, calcolata con le stesse modalità della *rank selection* illustrata per i casi mono-obiettivo. La probabilità infinita di essere selezionati viene assegnata al primo e all'ultimo cromosoma nella lista dell'importance selection che non devono essere obbligatoriamente gli individui con la migliore e la peggiore fitness.

Nel momento in cui l'algoritmo si trova all'ultima iterazione, la selezione avviene semplicemente considerando la fintess maggiore. Tale strategia di selezione, infatti, viene implementate per allargare lo spazio perlustrato dall'algoritmo e permettergli quindi di individuare soluzioni posizionate in punti dello spazio che precedentemente l'algoritmo non considerava. I risultati dell'importance selection sono confrontati con quelli della roulette wheel selection in tabella 4.13.

	Disposizione Importance selection			Disposizione roulette wheel selection	
Cromosoma	Fitness	Index	Fitness_new	**Cromosoma**	Fitness
9	6.9	∞	∞	**8**	7.3
8	7.3	0.375	2.7375	**9**	6.9
2	4.6	0.28125	1.29375	**1**	6.1
7	4.5	0.28125	1.265625	**4**	6
4	6	0.1875	1.125	**10**	5.5
10	5.5	0.1875	1.03125	**3**	5.1
5	4.1	0.15625	0.640625	**6**	4.9
3	5.1	0.15625	0.796875	**2**	4.6
1	6.1	0.125	0.7625	**7**	4.5
6	4.9	∞	∞	**5**	4.1

Tabella 4.13 Valutazione dei cromosomi con Importance selection e con Roulette wheel selection

L'ordine di importanza assegnato ai cromosomi non è lo stesso nei due casi. La prima differenza che si nota è il cromosoma presente in prima posizione in entrambe le selezioni, nella nuova il 9 mentre nella vecchia l'8. I due cromosomi si discostano notevolmente per il secondo obiettivo per importanze ovvero il numero di stazioni. Il cromosoma 9 ottiene un punteggio di cinque mentre il cromosoma 8 raggiunge l'unità. Mentre per l'obiettivo quarto per importanza, il cromosoma 9 ottiene un punteggio 4 mentre il cromsoma 8 un punteggio di nove. Le differenze invece tra gli altri obiettivi si differiscono di una sola unità. Con la Roulette wheel selection, l'algoritmo ha favorito il cromosoma con la fitness maggiore senza comprendere se in profondità e soprattutto per gli obiettivi più importanti, erano presenti cromosomi con risultati migliori.

In questo modo, l'algoritmo è in grado di trovare soluzioni migliori delle precedenti e di cui si riportano i risultati in tabella 4.14.

In tabella N_new indica il numero di stazioni delle soluzioni che l'algoritmo riesce a trovare con la nuova selezione e rispettivamente S_new rappresenta il valore delle skill, E_new il numero di equipment, D_new i cambi di direzione e infine WV-new indica il valore della workload-variance.

Possono essere confrontati con le soluzioni ottenute precedentemente e che possono essere visualizzate nel paragrafo 3.3.2.

Nelle istanze Warnecke, si ottiene un miglioramento notevole nei primi due obiettivi per importanza, la workload variance e il numero di stazioni. Nel caso Lutz1, in cui si ottenevano buoni valori per i primi due obiettivi, si è riusciti ad ottimizzare i restanti obiettivi. Nelle istanze Tonge si riesce ad ottenere un leggero miglioramento in tutti gli obiettivi eccetto una leggera stabilità nel numero di skill, ultimo per importanza. Nelle istanze Hahn, le più complesse a livello di grafo, si ottengono leggeri miglioramenti nei primi due obiettivi e discreta stabilità negli altri.

Caso di studio	N_new	S_new	E_new	D_new	WV_new
Barthold403	14	28	91	108	0,3001
Barthold434	13	27	87	115	0,1991
Barthold470	12	26	84	119	0,2012
Barthold513	11	23	84	112	0,2997
Barthold564	10	20	78	113	0,4438
Barthold626	9	20	74	112	0,7604
Barthold705	8	18	70	117	0,7966
Barthold805	7	14	15	115	0,0078
Gunther41	14	18	26	22	0,0112
Gunther44	12	17	23	21	0,0098
Gunther49	11	16	23	25	0,0044
Gunther54	10	15	23	25	0,0039
Gunther61	8	13	22	24	0,0122
Gunther69	8	13	22	22	0,05
Gunther81	7	11	19	22	0,028
Hahn2004	8	16	45	17	22,177
Hahn2338	7	14	42	22	71,0351
Hahn2806	6	10	38	20	4,1332
Hahn3507	5	10	37	17	8,8832

Hahn4676	4	7	32	17	17,2008
Lutz11414	11	16	22	25	2,9325
Lutz11572	10	15	22	26	1,9766
Lutz11768	9	14	20	29	6,3099
Luzt12020	8	13	19	24	5,4221
Lutz12357	7	12	17	24	9,3486
Lutz12828	6	10	23	25	2,5567
Tonge160	24	37	60	60	0,0446
Tonge168	23	37	55	61	0,0977
Tonge176	22	36	61	59	0,0885
Tonge185	22	35	52	59	0,0621
Tonge195	19	35	60	60	0,0875
Tonge207	18	34	58	57	0,0466
Tonge220	17	31	54	58	0,0788
Tonge234	16	29	57	59	0,0229
Tonge251	14	27	56	63	0,0575
Tonge270	14	23	55	62	0,0522
Tonge293	13	25	48	57	0,069
Tonge320	12	24	50	55	0,0898
Tonge364	10	19	52	60	0,0278
Tonge410	9	17	44	54	0,0481
Tonge468	8	14	46	50	0,0486
Tonge527	7	14	40	56	0,1053
Warnecke54	33	42	56	47	0,0076
Warnecke56	31	41	55	46	0,0088
Warnecke58	31	40	53	44	0,0135
Warnecke60	29	40	57	49	0,007
Warnecke62	29	39	52	50	0,0125
Warnecke65	28	37	53	50	0,0156
Warnecke68	25	37	55	53	0,009
Warnecke71	24	36	55	52	0,096
Warnecke74	23	36	54	42	0,0184
Warnecke78	22	34	54	44	0,0266
Warnecke82	21	36	52	43	0,0247
Warnecke86	19	33	53	47	0,0156
Warnecke92	18	32	53	44	0,0237
Warnecke97	17	27	52	45	0,0203
Warnecke104	16	27	54	47	0,0159
Warnecke111	15	26	55	46	0,0094

Riassumendo, si sono introdotte due modofiche sostanziali all'algoritmo GenIAL.

La prima ha riguardato una nuova metodologia di calcolo per i bounds degli insiemi che ha permesso di ottenere una maggiore congruenza tra la qualità della soluzione e il valore della funzione di fitness. Tale modifica non ha apportato miglioramenti a livello di soluzioni individuate dall'algoritmo. Per tale motivo si è implementate una tipologia di selezione diversa sia nei problemi mono-obiettivo, sia in quelli multi-obiettivo. La selezione immessa nell'algoritmo, permette a GenIAL di trovare soluzioni migliori a quelle precedenti soprattutto perché riesce a muoversi in uno spazio di individui superiore rispetto a quello precedente.

Capitolo 5

5 Conclusioni e sviluppi futuri

Il presente lavoro di tesi ha avuto come obiettivo la validazione e il miglioramento dell'algoritmo genetico GenIAL utilizzato per il bilanciamento di linee di assemblaggio manuali.

Bilanciare una linea di assemblaggio significa assegnare le operazioni di montaggio alle stazioni di lavoro in modo da ottenere vantaggi in termini di costi, tempi e risorse. Per effettuare tale bilanciamento è stato utilizzato l'algoritmo genetico basato sulla teoria evoluzionistica di Darwin, secondo cui solo gli individui che si adattano meglio all'ambiente hanno la possibilità di sopravvivere e trasmettere i propri geni agli individui figli.

Gli algoritmi genetici riproducono questo meccanismo utilizzando operatori genetici di combinazione e selezione e valutando, successivamente, la capacità di adattamento degli individui, tramite i valori della funzione di fitness, maggiore è tale valore maggiore la probabilità dell'individuo di sopravvivere e duplicare i propri geni.

Punto di partenza della tesi è stata la descrizione delle linee di assemblaggio e una classificazione dei principali problemi che possono interessare il processo di assemblaggio di un prodotto. Per la risoluzione di questi problemi vengono utilizzate, in letteratura, tecniche euristiche e meta-euristiche, tra le quali l'algoritmo genetico GenIAL.

Moto importante è stata la scelta di casi di studio che permettessero di testare con oggettività l'algoritmo e di comprendere la sua capacità di trovare soluzioni ottime. A tale proposito, per compiere la validazione dell'algoritmo, è stato impiegato un insieme di dati, utilizzato per comparare le procedure di risoluzione a livello mondiale.

Si è eseguita la validazione su due fronti differenti, i problemi mono-obiettivo e i problemi multi-obiettivo. I dati provenienti dai test hanno portato alla luce una prima inefficienza dell'algoritmo riguardante la divergenza tra il valore della funzione di fitness e la bontà della soluzione trovata, risolta modificando le metodologie che l'algoritmo adoperava per calcolare i valori del limite inferiore e superiore degli insiemi. Con tali modifiche, l'utente riesce con maggiore sicurezza a valutare la qualità della soluzione trovata dall'algoritmo tramite il valore della funzione di fitness.

Compresa la reale bontà dell'algoritmo, si è proceduto con la variazione del metodo di selezione, necessario per trasportare gli individui più promettenti di una popolazione da una generazione all'altra. Sono state applicate due tipologie di selezione differenti, una per i casi mono-obiettivo, una per i problemi multi-obiettivo. Entrambe permettono all'algoritmo di muoversi, durante le iterazioni, su uno spazio delle soluzioni più ampio e riuscire ad ottenere soluzioni migliori rispetto ai risultati ottenuti in precedenza.

Il presente lavoro di tesi pone le basi per decisioni future inerenti il bilanciamento e che attualmente coinvolgono i maggiori studiosi della letteratura.

Le problematiche inerenti il bilanciamento delle linee di assemblaggio, di cui una classificazione si è compiuta nel primo capitolo, sono varie e differenti tra loro. Al momento non esiste una algoritmo abbastanza flessibile da riuscire a risolverle tutte, ma esistono differenti versioni della stessa procedura che vengono modificate e rivisitate in funzione del problema da risolvere. Obiettivo futuro, sarà la realizzazione e implementazione di un algoritmo genetico, partendo da GenIAL, che sia in grado di:

- risolvere i problemi di tipologia SALBP-2, SALBP-E e SALBP-F delle linee di assemblaggio senza richiedere, ad ogni iterazione, modifiche e variazioni nei criteri di calcolo, necessari alla risoluzione del problema;
- risolvere contemporanemaente problemi multi-obiettivo e mono-obiettivo con la medesima tipologia di operatori genetici.

Bibliografia

T. Al-Hawari, O. Al-Araidah, *Development of a genetic algorithm for multi-objective assembly line balancing using multiple assgnment approach*, 2015

J. Anderson, C. Ferris, *Genetic Algorithms for combinatorial Optimization: The Assembly Line Balancing Problem*, 1993

R. Aydogan and P. Yolum. *Effective negotiation with partial preference information*, 2010

N. Barathwaj, P. Raja, *Optimization of assembly line balancing using genetic algorithm,* 2015

O. Brudaru, C. Rotaru, *Dynamic segregative genetic algorithm for assembly lines baancing,* 2010

O. Brudaru, B. Valmar, *Genetic algorithm with embryonic chromosomes for assembly line baalancing with fuzzy processing times,* 2004

S. Choudhary, S. Agrawal, *Multi-objective mixed model assembly line balancing problem,* 2015

K. Chung, *A multi-objective hybrid genetic algorithm to minimize total cost and delivery tardiness in a reverse logistics*, 2015

C. Coello, *D. A. Van Veldhuizen, Evolutionary algorithms for solving multi-objective problems. Kluwer*, 2002

Y. Collette and P. Siarry, *Multi-objective optimization: principles and case studies. Springer*, 2004

K. Deb, *Multi-objective optimization using evolutionary algorithms,* Chichester, 2001

J. Dou, J. Li, *A novel feasible task sequence-oriented discrete particle swarm algorithm for simple assembly line balancing problem of type 1*, 2013

M. Ehrgott, *A characterization of lexicographic max-ordering solutions. In Methods of Multicriteria Decision Theory*, 1997

E. Falkenauer, *A Hybrid Grouping Genetic Algorithm*, 1996

Fan Shu, Zi-Qi-Zu, Chao Mi, *Research on Engine Assembly Line Balancing Based on an Improved Genetic Algorithm*, 2013

S. Ginoria, G.L. Samuel, *Optimisation of machine loading problem using a genetic algorithm-based heuristic*, 2015

J. Goncalves, J. De Almeida, *A Hybrid Genetic Algorithm for Assembly Line Balancing*, 2002

T. Hager, M. Ahmed, *A genetic algorithm and a local search procedure for workload smoothing in assembly lines*, 2015

Hoffmann, Thomas, *Eureka a hybrid system for assmbly line balancing*, 1992

M. Jusop, Ab RAshid, *A review on simple assembly line balancing Type-1 problem*, 2015

H. Kailia, S. Dehuri, *Multi-objective genetic and fuzzy approaches in rule mining problem of knowledge discovery in databases*, 2014

Leu, Matheson, *Assembly Line Balancing Using Genetic Algorithms with Heuristic-Generated Initial Population and Multiple Evaluation Criteria*, 1994

Y. Lou, *A task scheduling algorithm based on genetic algorithm and ant colony optimization algorithm with multi- QoS constraints in cloud computing*, 2015

N. Mohd Razali, J. Geraghty, *Biologically Inspired Genetic Algorithm to Minimize Idle Time of the Assembly Line Balancing*, 2011

R. Nogueras, C. Cotta, *Studying self-balancing strategies in island-based multimemetic algorithm*, Malaga, 2014

S. Ozmehmet, *A review of the current applications of genetic algorithms in assembly line balancing*, 2007

M. Pini, *Methods for optimum allocation*, 2013

U. Saif, Z. Guan, *Multi-objective artificial bee colony algorithm for simultaneous sequenzing and balancing of mixed-model assembly line*, 2014

A. Scholl, *A classification of assembly line balancing problem*, 2007

A. Scholl, *Data of Assembly Line Balancing Problems*, 1997

A. Scholl, C. Becker, *State-of-the-art exact and heuristic solution procedures for simple assembly line balancing*, 2004

Su, Jun Li Chun, *A novel feasible task sequence-oriented discrete particle swarm algorithm for simple assembly line balancing problem of type 1*, 2013

Q. Tang, Z. Xiao, Y. Liang, *Novel Approach for Balancing Manual Automobile Assembly Based on Genetic Agorithm*, 2010

L. Tiacci, *Coupling a genetic algorithm approach and a discrete event simulator to design mixed model un-paced assembly lines with parallel workstations and stochastic task times*, 2015

Y. Tsujimura, M. Gen, *Solving fuzzy Assembly-line Balancing Problem with Genetic Algorithms,* 1995

Y. Wu, *GA based placement optimization for hybrid distributed storage*, 2015

R. Zhang, *A modified genetic algorithm for multi-criteria optimization based on Eucalyptus*, 2015

L.F. Zhang, Xi Zhou, *A novel fitness allocation algorithm for maintaining a constant selective pressure during GA procedure*, 2014

X. Zhu, D-Y Lei, *Bi-level hybrid genetic algorithm for three-dimensional container loading problem with balancing constrains*, 2015