Politecnico di Torino

Corso di Laurea Magistrale in Ingegneria Matematica

Anno accademico 2024/2025
Sessione di Laurea Marzo 2025

Tesi di Laurea Magistrale

Ottimizzazione dei parametri di un modello di simulazione di una supply-chain

Relatore
Prof. Paolo Brandimarte

Tutor Aziendali
Dott. Nicolò Mazzi
Dott. Andrea Boccolucci

Candidata
Erica Susca

Ai miei genitori
A mia nonna Pina
A Goblin

Sommario

Questo lavoro di tesi è stato realizzato con il supporto di Spindox, un'azienda operante nel settore ICT, che ha fornito una supervisione costante durante lo sviluppo del progetto.

Lo scopo di questa tesi è sviluppare un framework per l'ottimizzazione dei parametri di un modello di simulazione di una supply-chain. Il framework proposto garantisce versatilità e robustezza, risultando applicabile anche a problemi black-box generici caratterizzati da alta complessità e funzioni non analitiche.

Il primo capitolo introduce i concetti fondamentali dell'ottimizzazione basata su simulazione, con particolare attenzione ai problemi black-box e ai modelli surrogati, come le Radial Basis Functions, oltre a tecniche di campionamento come il Latin Hypercube Sampling.

Il secondo capitolo analizza l'algoritmo di ottimizzazione OPUS e confronta le sue prestazioni con i risultati ottenuti mediante metodi di ricerca diretta, quali Particle Swarm Optimization (PSO) e Differential Evolution (DE). Viene descritto il funzionamento di OPUS, insieme ai risultati ottenuti utilizzando una funzione test analitica.

Il terzo capitolo applica OPUS a un modello semplificato di supply-chain, evidenziando la capacità dell'algoritmo di ottimizzare i parametri del sistema e bilanciare le sue componenti, anche in presenza di variazioni nei parametri.

Il quarto capitolo esplora l'integrazione di OPUS con un approccio di decomposizione basato su Differential Grouping (DG2) e Cooperative Coevolution (CC). Sono presentati risultati sperimentali volti ad analizzare l'efficacia della decomposizione, confrontando le prestazioni ottenute nei casi con e senza decomposizione.

La tesi si conclude con una discussione sui risultati ottenuti e con un suggerimento per un possibile sviluppo futuro, relativo all'adozione di una strategia di decomposizione parallela, finalizzata a migliorare l'efficienza computazionale del processo di ottimizzazione.

Ringraziamenti

Vorrei ringraziare tutti coloro che, in un modo o nell'altro, mi hanno aiutata a raggiungere un traguardo per me molto importante.
Innanzitutto ringrazio il professor Brandimarte, per avermi guidata nella fase conclusiva di questo percorso accademico, in particolare per la sua disponibilità, la sua preziosa esperienza e per i suoi utili consigli.
Desidero inoltre esprimere la mia gratitudine all'azienda Spindox e, in particolare, a Nicolò e Andrea, per avermi guidata con competenza e disponibilità durante lo sviluppo di questa tesi. Grazie a loro ho avuto l'opportunità di approfondire tematiche per me di grande interesse, avvalendomi di strumenti e di un prezioso confronto.
Un ringraziamento speciale va ai miei genitori che mi hanno sempre spronata a dare il massimo in ogni cosa, non solo nello studio, supportata e che non mi hanno mai fatto pesare niente.
Non posso non ringraziare mia nonna che ogni giorno mi accudisce, mi supporta e sopporta, mi strappa una risata e mi accontenta in ogni desiderio, soprattutto culinario.
Un insolito ringraziamento va al mio cane Goblin che mi tiene sempre compagnia, ogni giorno mi dimostra il suo affetto e funge da peluche antistress.
Un pensiero speciale va ad Aurora, che non ha mai fatto mancare un "in bocca al lupo" prima di ogni esame ed è stata il mio porta fortuna. Mi ha sempre incoraggiata, credendo nelle mie capacità più di chiunque altro, persino più di me stessa.
Infine un ringraziamento va a tutti i miei amici e compagni di università che hanno condiviso con me sacrifici, successi e gioie. In particolare, desidero ringraziare Fabio, con cui ho avuto il piacere di confrontarmi e di collaborare nei momenti di difficoltà, trovando sempre supporto e stimolo reciproco.

Indice

Introduzione

Nel contesto competitivo e dinamico delle moderne supply-chain, l'ottimizzazione dei processi riveste un ruolo cruciale per garantire efficienza operativa, riduzione dei costi e adattabilità alle mutevoli condizioni di mercato. La crescente complessità dei sistemi di supply-chain, richiede che essi siano spesso modellati attraverso simulazioni computazionali avanzate e complesse. Tuttavia, l'efficacia di questi modelli di simulazione dipende fortemente dalla corretta definizione dei loro parametri, che devono essere ottimizzati per garantire risultati accurati e applicabili.

L'obiettivo di questa tesi è quindi quello di sviluppare un framework per l'ottimizzazione dei parametri di un modello di simulazione applicato al contesto delle supply-chain, dove il problema di ottimizzare un modello di simulazione è trattato come un problema di tipo *black box*. Per problema di tipo black box, si intende che non si conoscono in forma analitica la funzione obiettivo e/o i vincoli del problema, in quanto questi non sono esplicitamente noti, ma si ha soltanto la possibilità di valutare la funzione obiettivo in punti specifici dello spazio di ricerca, attraverso una simulazione dispendiosa in termini di tempo e risorse computazionali.
Proprio per questi motivi le tecniche tradizionali basate sul gradiente possono risultare inefficaci o meglio inapplicabili, rendendo necessaria l'adozione di metodologie alternative, come l'ottimizzazione senza derivate (*derivative-free optimization*, DFO). Queste metodologie rappresentano quindi strumenti pratici e realistici per affrontare problemi di questo tipo, che rientrano nell'ambito dell'Ottimizzazione basata sulla simulazione (*Simulation-Based Optimization*), che mira a sviluppare strategie per ottimizzare sistemi complessi rappresentati mediante simulazioni.
Per questo motivo, si ritiene opportuno fornire una panoramica dei principali metodi di ottimizzazione senza derivate (DFO) presenti in letteratura, analizzandone le caratteristiche principali.

In T. Foroud [2018], vengono presentati i principali algoritmi di ottimizzazione senza derivate (*derivative-free optimization*, DFO), tra cui il Particle Swarm Optimization (PSO), il Differential Evolution (DE) e il Pattern Search (PS).
Questi algoritmi, sono detti metodi di ricerca diretta (*Direct Search Methods*), ed appartenendo alla categoria dei metodi DFO, non richiedono il calcolo delle derivate della funzione obiettivo. Essi si basano invece unicamente sui valori della funzione valutati in punti specifici dello spazio delle soluzioni, proprio per questi motivi, sono particolarmente adatti per problemi *black box*.

Questi metodi si distinguono per la loro semplicità implementativa e la capacità di

affrontare problemi complessi senza la necessità di modelli analitici precisi. Ogni metodo segue una strategia di esplorazione dello spazio delle soluzioni, cercando iterativamente miglioramenti nella funzione obiettivo. Si fornisce di seguito una breve descrizione dei tre metodi principali analizzai in T. Foroud [2018]:

- **PSO (Particle Swarm Optimization)**: è un algoritmo di ricerca globale ispirato al comportamento sociale degli uccelli e dei pesci. Utilizza una popolazione di particelle che si muovono nello spazio delle soluzioni, aggiornando le loro posizioni in base alle esperienze personali e collettive, al fine di trovare la soluzione ottimale.
- **DE (Differential Evolution)**: è un metodo di ricerca globale che sfrutta un insieme di soluzioni candidate. I nuovi agenti vengono generati combinando le posizioni di altri agenti esistenti, accettando solo quelli che migliorano la funzione obiettivo.
- **PS (Pattern Search)**: è un metodo di ricerca locale che utilizza un pattern di punti per valutare la funzione obiettivo. Il pattern viene aggiornato sequenzialmente per cercare un punto di minimo, riducendo la dimensione della ricerca se non si trova un miglioramento.

Tuttavia, la semplicità implementativa dei metodi di ricerca diretta comporta alcuni svantaggi. In particolare, A. Kiuchi [2020] sottolinea un limite fondamentale di questi approcci: l'elevato numero di valutazioni richiesto per esplorare lo spazio delle soluzioni, che può diventare proibitivo quando le simulazioni sono particolarmente costose.

I *Response Surface Methods* (RSM) rappresentano un'alternativa valida ai metodi di *Direct Search*, in quanto permettono di superare le criticità legate all'elevato numero di valutazioni della funzione obiettivo.

Tra i RSM, J.R. Gardner [2014] e T. Ogura [2022] propongono l'adozione di un framework basato sull'ottimizzazione Bayesiana, che utilizza in particolare i Processi Gaussiani (*Gaussian Process*, GP). Questo approccio ha l'obiettivo di ridurre il numero di valutazioni necessarie, approssimando la funzione obiettivo tramite un modello surrogato, il quale può essere valutato con un costo computazionale inferiore rispetto alla funzione obiettivo originale. Si procede costruendo iterativamente il modello surrogato a partire dai dati disponibili. Ad ogni iterazione, il modello seleziona il punto più promettente per la valutazione successiva utilizzando una funzione di acquisizione. La funzione obiettivo costosa viene valutata solo in questo punto, e successivamente il modello viene aggiornato con i nuovi dati, migliorando così l'accuratezza della stima e ottimizzando l'efficacia delle iterazioni successive.

In J.R. Gardner [2014] e Y. Tang [2012] vengono presentate due strategie principali per affrontare problemi vincolati, in cui, oltre alla funzione obiettivo di tipo *black box*, sono presenti vincoli che possono essere anch'essi di tipo *black box*. La prima (Y. Tang [2012]) prevede l'introduzione di penalità elevate per i punti che violano i vincoli, trasformando il problema in uno senza vincoli. La seconda (J.R. Gardner [2014]) utilizza un processo gaussiano separato per modellare i vincoli stessi. In particolare, la probabilità di rispettare i vincoli viene incorporata nella funzione di acquisizione, che viene adattata per esplorare le regioni promettenti dello spazio che rispettano i vincoli con alta probabilità.

Infine, Regis [2014] esplora un approccio alternativo basato su *Radial Basis Function* (RBF) combinato con il PSO. Questo metodo, denominato OPUS (Optimization by Particle Swarm Using Surrogates), sfrutta un modello RBF per pre-selezionare le particelle più promettenti, riducendo così il numero di valutazioni costose della funzione obiettivo. L'approccio può anche essere esteso per gestire vincoli di uguaglianza e disuguaglianza, trasformandoli in funzioni di penalità incluse nella funzione obiettivo, come è mostrato in Y. Tang [2012] . Questo permette di spingere le soluzioni verso regioni ammissibili dello spazio delle soluzioni, migliorando l'efficienza complessiva del processo di ottimizzazione.

Capitolo 1

Cenni teorici

Il presente capitolo ha l'obiettivo di fornire una panoramica sulle caratteristiche dei problemi di Simulation-Based Optimization, trattati come problemi black box, e di analizzare le peculiarità delle funzioni black box stesse. Comprendere le caratteristiche di questi problemi è fondamentale per individuare l'approccio più efficace per la loro risoluzione, poiché si contraddistinguono per la loro complessità e per la difficoltà di modellare esplicitamente la funzione obiettivo.

Inoltre, verranno introdotti i principali concetti teorici relativi ai modelli surrogati, che rappresentano strumenti fondamentali in alcuni metodi di risoluzione avanzata, già citati nell'introduzione e che saranno approfonditi nei capitoli successivi. I modelli surrogati permettono di semplificare l'analisi e la ricerca di soluzioni per per problemi di questo tipo, creando una rappresentazione approssimativa ma computazionalmente più gestibile. Comprendere come sono costruiti e quali sono le loro proprietà aiuta a sfruttarli al meglio nell'applicazione pratica degli approcci di ottimizzazione.

1.1 Problemi di Simulation-Based Optimization e funzioni black box

Si consideri un problema di Simulation-Based Optimization, dove $\mathbf{x} \in \mathbb{R}^d$ rappresenta il vettore associato alle variabili decisionali del problema e $f : \mathbb{R}^d \to \mathbb{R}$ rappresenta la funzione obiettivo che si vuole ottimizzare. Si assuma un dominio d-dimensionale dello spazio dei parametri rappresentato da $\Omega = \{\mathbf{x} \in \mathbb{R}^d : \mathbf{x}_{L,j} \leq \mathbf{x}_j \leq \mathbf{x}_{U,j},\ j = 1, \ldots, d\}$, il problema di ottimizzazione che considerato è il seguente S. Jakobsson [2010]:

$$\begin{aligned} \min_{\mathbf{x}} \quad & f(\mathbf{x}) \\ \text{s.t.} \quad & \mathbf{x}_{L,j} \leq \mathbf{x}_j \leq \mathbf{x}_{U,j},\ j = 1, \ldots, n \end{aligned} \tag{1.1}$$

dove i vincoli sono di tipo box e la funzione obiettivo f è una funzione black box.
In questo caso si considera un problema dove sono presenti solo vincoli di box, ma è possibile considerare anche problemi con vincoli più generali di uguaglianza e disuguaglianza, in quanto introducendo penalità elevate per i punti che violano i vincoli all'interno della

funzione obiettivo, il problema può essere ricondotto al problema 1.1, come illustrato in J.R. Gardner [2014]. In particolare, se si avesse un problema del tipo:

$$\begin{aligned} \min_{\mathbf{x}} \quad & f(\mathbf{x}) \\ \text{s.t.} \quad & g_i(\mathbf{x}) \leq 0, \quad i \in E \\ & h_j(\mathbf{x}) = 0, \quad j \in I \end{aligned} \tag{1.2}$$

questo potrebbe essere trasformato in un problema non vincolato, utilizzando una funzione di penalità in questo modo:

$$\min F(x) = \min \left\{ f(x) + R \sum_{i \in E} [\max(0, g_i(x))]^2 + R \sum_{j \in I} [h_j(x)]^2 \right\} \tag{1.3}$$

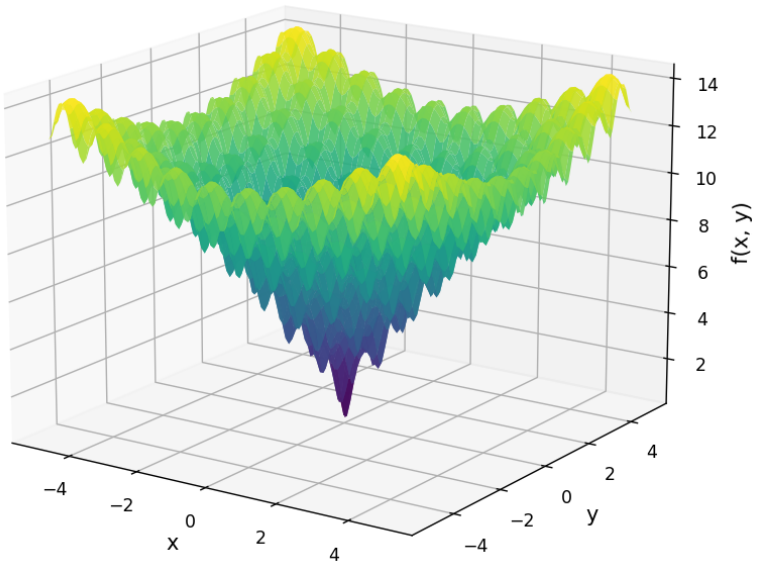
Il secondo e terzo termine del lato destro dell'equazione rappresentano i termini di penalità, pesati da un fattore di penalità $R > 0$, dove R è un coefficiente scelto in maniera opportuna, se R è molto grande allora i vincoli saranno rispettati in modo rigoroso. Se i vincoli vengono violati in $\mathbf{x}$, i termini di penalità assumono valori elevati, spingendo la soluzione verso la regione ammissibile. Viceversa, quando i vincoli sono soddisfatti, i termini di penalità sono nulli.

In un problema del tipo 1.1, la funzione obiettivo f presenta le seguenti caratteristiche principali, come illustrato in Gutmann [2001]:

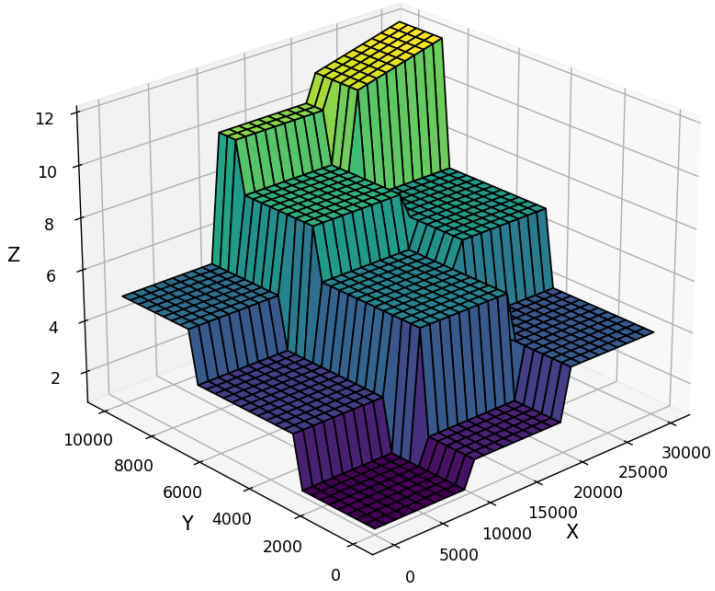
- Essendo una funzione black box, l'unica informazione disponibile consiste nella possibilità di valutare f in punti specifici. Non sono disponibili informazioni relative alle derivate di f, né è noto se la funzione sia derivabile o meno. Un esempio di funzione non derivabile è riportato in Figura 1.1b.
- La funzione f potrebbe essere multi-modale, ovvero caratterizzata dalla presenza di più minimi locali, rendendo la ricerca del minimo globale più complessa. Un esempio di funzione multi-modale è riportato in Figura 1.1a.
- Poiché la valutazione di f è il risultato di una simulazione, la funzione simulata $\tilde{f}$ potrebbe essere costosa da valutare sia in termini di tempo che di risorse computazionali.
- Il tempo complessivo richiesto per il processo di ottimizzazione è fortemente influenzato dal numero di valutazioni della funzione obiettivo. Pertanto, un aspetto cruciale consiste nel ridurre al minimo il numero di valutazioni necessarie per individuare una soluzione ottimale.

Un altro aspetto importante delle simulazioni, sottolineato in A.I.J. Forrester [2008] e accennato in S. Jakobsson [2010], è che l'output potrebbe contenere del rumore, che varia a seconda del tipo di fenomeno simulato. Ad esempio, il rumore può derivare da:

- *Errore umano*, come errori o bugs nella scrittura del codice per la simulazione.



(a) Funzione di Ackley, riportata come esempio di funzione multimodale.



(b) Funzione a gradoni, riportata come esempio di funzione non differenziabile.

Figura 1.1: Confronto tra due tipologie di funzioni utilizzate nell'ottimizzazione.

- *Errore sistematico*, dovuto a imperfezioni nell'esperimento fisico (nel caso di simulazioni fisiche) o a semplificazioni e approssimazioni introdotte nel codice di simulazione (nel caso di simulazione al computer).

- *Errore casuale*, questo presente solo nel caso di simulazioni fisiche, legato ad inaccuratezze negli strumenti di misura o nella raccolta dei dati.

Per affrontare problemi con rumore, si distingue tra la funzione reale *true function*, indicata con f, e l'output della simulazione, indicato con $\tilde{f}$, chiamato *noisy function*. L'obiettivo è minimizzare f, ma spesso si ha accesso solo a $\tilde{f}$. In assenza di rumore, le due funzioni coincidono ($f = \tilde{f}$).

1.2 Approssimazione di funzioni e modelli surrogati

Nel caso di funzioni la cui forma analitica non è nota e per le quali la valutazione in un singolo punto può risultare estremamente onerosa in termini computazionali, come per il caso di funzioni black box, le tecniche di approssimazione assumono un ruolo cruciale. Questi metodi permettono di ottenere una funzione surrogata s, definita in forma analitica, che fornisce un'approssimazione accurata della funzione originale f. La funzione s è progettata per essere facilmente valutabile, richiedendo un ridotto sforzo computazionale rispetto alla funzione di partenza.
Ci sono diverse tecniche di approssimazione, tra le quali l'approssimazione utilizzando funzioni polinomiali. Tali tecniche funzionano bene in spazi mono dimensionali, dove le funzioni interpolanti esistono sempre. I polinomi, infatti, possono essere utilizzati come interpolanti, a condizione che il loro grado sia sufficientemente elevato. Quando si ha a che fare con problemi in più dimensioni però, come nei contesti e applicazioni pratiche, esistono sempre distribuzioni di dati per cui non è possibile trovare un polinomio che interpola i dati.
Infatti, come spiegato in M. Buhmann [2001], le funzioni approssimanti sono costruite a partire da funzioni di base, ossia una collezione di funzioni più semplici, scelte in modo tale che ogni funzione approssimante ottenuta possa essere espressa come una combinazione lineare delle funzioni base; per esempio nel caso dei polinomi in spazi mono dimensionali, il set delle funzioni di base è dato da $\{1, x, x^2, x^3 ...\}$.
Tuttavia, se le funzioni base vengono fissate a priori e si tenta di interpolare punti dati arbitrari in uno spazio di due o più dimensioni, esisteranno inevitabilmente distribuzioni dei punti per le quali l'interpolazione risulterà impossibile utilizzando la base scelta. Di conseguenza, è necessario selezionare le funzioni di base solo dopo aver acquisito informazioni sui punti dati.
Ne consegue che tecniche di approssimazione come l'interpolazione polinomiale, l'uso di splines (cioè funzioni polinomiali definite a tratti) o le tecniche di interpolazione mediante il metodo dei minimi quadrati non sono adeguate per problemi che coinvolgono più di una dimensione. Per tale motivo, nei contesti applicativi reali, si ricorre all'utilizzo delle funzioni a base radiale (*Radial Basis Functions*).

1.2.1 Radial Basis Functions

Le funzioni a base radiale sono mezzi per approssimare funzioni continue che dipendono da due o più variabili, attraverso combinazioni di termini basati su una singola funzione

di una variabile, denotata con $\phi : \mathbb{R}^+ \to \mathbb{R}$ e chiamata *Radial Basis Function*. Come evidenziato in M. Buhmann [2001] la differenza tra l'approssimazione tramite funzioni a base radiale e gli altri tipi di approssimazione citati precedentemente, consiste nel fatto che in questo caso non si ha più un set di funzioni di base, bensì si ha una **singola funzione di base** con cui costruire l'approssimazione. Per passare da ϕ a una funzione multivariata, si definisce una nuova funzione Φ a partire dal vettore $\mathbf{x} = (x_1, \ldots, x_d) \in \mathbb{R}^d$, come segue: $\Phi(\mathbf{x}) = \phi(||\mathbf{x}||_2)$, dove $||\mathbf{x}||_2$ è la norma Eulcidea di $\mathbf{x}$; poichè la funzione $\Phi(\mathbf{x})$ dipende solo dalla distanza di $\mathbf{x}$ dall'origine, la funzione è detta radialmente simmetrica, da qui il nome di funzioni a base radiale.
Si consideri ora la funzione a valori scalari $f : \mathbb{R}^d \to \mathbb{R}$ e un set di un numero finito di punti $\{\mathbf{x}_j \in \mathbb{R}^d \quad j = 1, \ldots, n\}$, il set di funzioni a base radiale è definito a partire da una singola funzione di base ϕ, effettuando una traslazione di questa funzione nei punti dati: $\phi(||\mathbf{x} - \mathbf{x}_j||_2)$, in questa maniera si ha una funzione di base per ogni punto del set finito $\{\mathbf{x}_j \in \mathbb{R}^d \quad j = 1, \ldots, n\}$, questi punti prendono il nome di **centri** e le funzioni di base ottenute vengono combinate linearmente per formare la funzione di approssimazione s.

$$s(\mathbf{x}) = \sum_j \lambda_j \phi(||\mathbf{x} - \mathbf{x}_j||_2) \tag{1.4}$$

dove i λ_j sono coefficienti reali. Un elemento distintivo delle funzioni a base radiale, infatti, è la loro dipendenza dai dati, caratteristica che rappresenta un vantaggio significativo rispetto all'interpolazione polinomiale o ad altre tecniche di interpolazione, come evidenziato in M. Buhmann [2001]. Questa proprietà consente alle funzioni a base radiale di rendere l'interpolazione possibile anche in spazi ad alta dimensionalità.
In particolare, si vuole scegliere s in modo tale che le condizioni di interpolazione siano rispettate nei punti dati, ovvero in modo tale che i valori di s ed f in tutti i punti del set finito coincidano:

$$s(\mathbf{x}_k) = f(\mathbf{x}_k) \quad k = 1, \ldots, n$$

o più esplicitamente:

$$\sum_j \lambda_j \phi(||\mathbf{x}_k - \mathbf{x}_j||_2) = f(\mathbf{x}_k) \quad k = 1, \ldots, n \tag{1.5}$$

Come evidenziato in A.I.J. Forrester [2008], la funzione di interpolazione potrebbe essere in grado di approssimare funzioni altamente non lineari, ma per trovare i coefficienti reali λ_j è sufficiente risolvere delle equazioni lineari (1.5), le quali possono essere riarrangiate in un sistema e riscritte in forma matriciale, in questo modo:

$$\boldsymbol{\Phi}\boldsymbol{\lambda} = \mathbf{F}$$

Dove $\boldsymbol{\lambda}$ è il vettore dei coefficienti $\lambda_j, j = 1, \ldots, n$, $\mathbf{F}$ è il vettore delle valutazioni della funzione nei punti dati $f(\mathbf{x}_j), j = 1, \ldots, n$ e infine $\boldsymbol{\Phi}$ denota la *Matrice di Gram*, ed è definita così: $\boldsymbol{\Phi}_{j,k} = \phi(||\mathbf{x}_j - \mathbf{x}_k)||_2 \quad i, j = 1, \ldots, n$.
Per determinare il vettore $\boldsymbol{\lambda}$, è necessario risolvere l'equazione $\boldsymbol{\lambda} = \boldsymbol{\Phi}^{-1}\mathbf{F}$. Da ciò deriva l'esigenza che la matrice $\boldsymbol{\Phi}$ sia invertibile, il che implica che essa debba essere quadrata e non singolare. Per questo motivo, la scelta della funzione a base radiale riveste

un'importanza cruciale e ricade spesso su un insieme ristretto di possibilità Gutmann [2001]:

$$\begin{aligned}
&\phi(r) = r \quad \text{(lineare)},\\
&\phi(r) = r^3 \quad \text{(cubica)},\\
&\phi(r) = r^2 \log r \quad \text{(thin plate spline)},\\
&\phi(r) = \sqrt{r^2 + c^2} \quad \text{(multiquadrica)},\\
&\phi(r) = e^{-\gamma r^2} \quad \text{(gaussiana)},
\end{aligned} \tag{1.6}$$

dove $r > 0$ e γ è una costante positiva prefissata.

La matrice Φ, però, potrebbe essere singolare. Ad esempio, se $\phi(r) = r^2 \log r$, $n = d+1$ e i punti $\mathbf{x}_1, \ldots, \mathbf{x}_{d+1}$ formano un simplesso i cui lati hanno lunghezza 1, allora $\Phi = 0$. Quindi, in questo caso non esiste alcun interpolante. Tuttavia, qualunque insieme di dati $f_1, \ldots, f_{d+1}$ può essere interpolato da un polinomio lineare. Per questo motivo la funzione di approssimazione s definita in (1.4) viene modificata in questo modo:

$$s(\mathbf{x}) = \sum_j \lambda_j \phi(||\mathbf{x} - \mathbf{x}_j||_2) + p(\mathbf{x}) \tag{1.7}$$

dove il primo termine, ovvero quello che è rimasto inalterato rispetto alla formula precedente $\sum_j \lambda_j \phi(\|\mathbf{x} - \mathbf{x}_j\|_2)$, consente di modellare componenti altamente non lineari della funzione. A questo si aggiunge il termine $p(\mathbf{x})$, un polinomio appartenente allo spazio dei polinomi di grado minore o uguale a m, ovvero $p(\mathbf{x}) \in \Pi_m$. Il ruolo di $p(\mathbf{x})$ è quello di modellare la componente più regolare (ovvero *smooth*) della funzione. In generale, si vogliono trovare dei valori di m che garantiscano l'**esistenza** e l'**unicità** del polinomio interpolante. Per affrontare questo problema bisogna introdurre il concetto di **Conditional Definitness** S. Jakobsson [2010].

Definizione 1.2.1. *Sia* $\mathcal{V}_m \subset \mathbb{R}^n$ *tale che:*

$$\mathcal{V}_m = \left\{ \mathbf{c} \in \mathbb{R}^n : \sum_{j=1}^{n} c_j p(\mathbf{x}_j) = 0,\ \forall p \in \Pi_{m-1}(\mathbb{R}^d) \right\},$$

dove $\Pi_{m-1}(\mathbb{R}^d)$ *è l'insieme dei polinomi di grado al massimo* $m-1$.

Una funzione radiale continua $\phi : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ *è detta* ***condizionalmente positiva definita di ordine*** m *se, per ogni* $n \in \mathbb{N}$, *qualsiasi insieme di punti distinti* $\mathbf{x}_1, \ldots, \mathbf{x}_n \in \mathbb{R}^d$ *e ogni* $\mathbf{c} \in \mathcal{V}_m \setminus \{0\}$, *si ha:*

$$\sum_{j=1}^{n} \sum_{k=1}^{n} c_j c_k \phi(\mathbf{x}_j, \mathbf{x}_k) > 0.$$

La funzione radiale $\phi(\mathbf{x}, \mathbf{y}) := \phi(\|\mathbf{x} - \mathbf{y}\|)$ *è detta condizionalmente positiva definita di ordine* m *se è soddisfatta la definizione sopra. Inoltre, una funzione condizionalmente positiva definita di ordine 1 è anche condizionalmente positiva definita per ogni* $m > 1$, *ovvero* $\mathcal{V}_m \subset \mathcal{V}_1$.

In Gutmann [2001], $\mathcal{V}_m$, è definito come l'insieme di tutti i vettori $\boldsymbol{\lambda} \in \mathbb{R}^n$ che soddisfano la relazione

$$\sum_{i=1}^{n} \lambda_i q(\mathbf{x}_i) = 0 \quad \forall q \in \Pi_m, \tag{1.8}$$

dove Π_m rappresenta lo spazio dei polinomi di grado al più m. Formalmente, si pone $\mathcal{V}_{-1} := \mathbb{R}^n$.

Come riportato in Gutmann [2001], che fa riferimento ai risultati ottenuti da Powell in Powell [1992], nei casi di *cubic* e *thin plate splines*, vale la relazione

$$\boldsymbol{\lambda}^T \boldsymbol{\Phi} \boldsymbol{\lambda} > 0 \quad \forall \boldsymbol{\lambda} \in \mathcal{V}_1 \setminus \{0\}, \tag{1.9}$$

mentre nei casi di *linear* e *multiquadric splines*, si ha

$$\boldsymbol{\lambda}^T \boldsymbol{\Phi} \boldsymbol{\lambda} < 0 \quad \forall \boldsymbol{\lambda} \in \mathcal{V}_0 \setminus \{0\}. \tag{1.10}$$

Nel caso delle *Gaussian Radial Basis Functions*, si verifica invece

$$\boldsymbol{\lambda}^T \boldsymbol{\Phi} \boldsymbol{\lambda} > 0 \quad \forall \boldsymbol{\lambda} \in \mathbb{R}^n \setminus \{0\}. \tag{1.11}$$

Se si definisce m_0 come:

- $m_0 = 1$ per i casi di *cubic* e *thin plate splines*,
- $m_0 = 0$ per i casi di *linear* e *multiquadric splines*,
- $m_0 = -1$ nel caso *Gaussian*.

allora le disuguaglianze (1.9)–(1.11) possono essere sintetizzate come segue:

$$(-1)^{m_0+1} \boldsymbol{\lambda}^T \boldsymbol{\Phi} \boldsymbol{\lambda} > 0 \quad \forall \boldsymbol{\lambda} \in \mathcal{V}_{m_0} \setminus \{0\}. \tag{1.12}$$

Dopo aver scelto la funzione radiale ϕ, si fissa un intero $m \geq m_0$ e si impone che $\boldsymbol{\lambda}$ appartenga a $\mathcal{V}_m$. Sia $\hat{m}$ la dimensione dello spazio Π_m, e sia $p_1, \ldots, p_{\hat{m}}$ una base per questo spazio. La matrice P è quindi definita come segue:

$$P := \begin{pmatrix} p_1(\mathbf{x}_1) & \cdots & p_{\hat{m}}(\mathbf{x}_1) \\ \vdots & \ddots & \vdots \\ p_1(\mathbf{x}_n) & \cdots & p_{\hat{m}}(\mathbf{x}_n) \end{pmatrix}. \tag{1.13}$$

Lo spazio $\mathcal{V}_m$ è allora definito come l'insieme di tutti i vettori $\boldsymbol{\lambda} \in \mathbb{R}^n$ che soddisfano $P^T \boldsymbol{\lambda} = 0$. Inoltre, sempre come riportato in Gutmann [2001], che a sua volta fa riferimento a Powell [1992], si può dimostrare che la matrice

$$A = \begin{pmatrix} \Phi & P \\ P^T & 0 \end{pmatrix} \in \mathbb{R}^{(n+\hat{m})\times(n+\hat{m})} \tag{1.14}$$

è non singolare se e solo se i punti $\mathbf{x}_1, \ldots, \mathbf{x}_n$ soddisfano la condizione:

$$q \in \Pi_m \text{ e } q(\mathbf{x}_i) = 0,\ i = 1, \ldots, n \implies q \equiv 0. \tag{1.15}$$

Nel caso delle funzioni radiali Gaussiane con $m = -1$, invece, la matrice P e la condizione (1.15) non sono richieste, in quanto ci si riconduce alla funzione di approssimazione s nella forma di (1.4); di conseguenza, i coefficienti della funzione s in (1.7) sono univocamente determinati dal sistema:

$$s(\mathbf{x}_i) = f_i, \quad i = 1, \dots, n, \tag{1.16}$$

$$\sum_{i=1}^{n} \lambda_i p_j(\mathbf{x}_i) = 0, \quad j = 1, \dots, \hat{m}. \tag{1.17}$$

In generale, invece, indicando ancora con F il vettore che contiene i valori $f_1, \dots, f_n$, il sistema (1.17) può essere rappresentato in forma matriciale, come riportato in Gutmann [2001]:

$$\begin{pmatrix} \mathbf{\Phi} & P \\ P^T & 0 \end{pmatrix} \begin{pmatrix} \boldsymbol{\lambda} \\ \mathbf{c} \end{pmatrix} = \begin{pmatrix} \mathbf{F} \\ \mathbf{0}_{\hat{m}} \end{pmatrix}. \tag{1.18}$$

dove $\boldsymbol{\lambda} = (\lambda_1, \dots, \lambda_n)^T \in \mathbb{R}^n$, $\mathbf{c} \in \mathbb{R}^{\hat{m}}$ e $\mathbf{0}_{\hat{m}}$ è il vettore nullo in $\mathbb{R}^{\hat{m}}$. Le componenti di $\mathbf{c}$ sono i coefficienti del polinomio p rispetto alla base $p_1, \dots, p_{\hat{m}}$. Per determinare la funzione approssimante s nella forma di (1.7), è necessario risolvere il sistema lineare (1.18) rispetto ai vettori $\boldsymbol{\lambda}$ e $\mathbf{c}$. Poiché ora si ha la garanzia che il sistema sia non singolare e invertibile, la soluzione esiste ed è unica.

1.2.2 Radial Basis Function per dati rumorosi

Nel caso in cui l'output $\mathbf{F} = \{f(\mathbf{x}_1), f(\mathbf{x}_2), \dots, f(\mathbf{x}_n)\}$ contenga del rumore, utilizzare il metodo precedente potrebbe portare a un modello che si adatta eccessivamente ai dati e quindi potrebbe portare ad avere overfitting; questo avviene poichè il modello non riesce a distinguere in modo efficace tra il vero output e il rumore presente nei dati. Per risolvere questo problema, come illustrato in A.I.J. Forrester [2008], si introduce una maggiore flessibilità nel modello tramite un *parametro di regolarizzazione* w, che viene aggiunto alla diagonale principale della matrice di Gram. Così facendo, la funzione di approssimazione s non passerà più necessariamente attraverso i punti dati, e il vettore $\boldsymbol{\lambda}$ sarà la soluzione dei minimi quadrati:

$$\boldsymbol{\lambda} = (\mathbf{\Phi} + w\mathbf{I})^{-1}\mathbf{F}, \tag{1.19}$$

dove $\mathbf{I}$ è una matrice identità $n \times n$. Idealmente, il valore di w dovrebbe essere uguale alla varianza del rumore nei dati $\mathbf{F}$, ma poiché di solito non è noto, rimane come uno dei parametri da stimare.

Un approccio alternativo, sempre mostrato in A.I.J. Forrester [2008], per costruire una funzione di approssimazione attraverso dati rumorosi utilizzando funzioni di base radiali, consiste nel ridurre il numero di basi. Un numero di basi inferiore a n produce una matrice $\mathbf{\Phi}$ non quadrata, ed è possibile utilizzare l'Equazione (1.5) per ottenere una stima dei minimi quadrati di w.

1.2.3 Latin Hypercube Sampling (LHS)

La presenza di punti molto vicini tra loro nello spazio può generare problemi di mal-condizionamento nella matrice $\mathbf{\Phi}$, compromettendo la stabilità numerica del processo di interpolazione, come spiegato in A.I.J. Forrester [2008]. Per evitare il problema di una copertura non uniforme dello spazio durante la creazione iniziale di un modello di interpolazione, sono state sviluppate diverse tecniche di campionamento dei punti. Queste tecniche mirano a garantire che i punti iniziali siano distribuiti in modo tale da coprire uniformemente l'intero spazio di interesse, per questa caratteristica esse sono dette *space-filling*. Una delle metodologie più utilizzate a questo scopo è il *Latin Hypercube Sampling (LHS)*.

Il *Latin Hypercube Sampling* è una tecnica di campionamento stratificato progettata per generare distribuzioni di punti in uno spazio multidimensionale in modo da ottenere una copertura uniforme. Si consideri uno spazio d-dimensionale delimitato da un dominio $\Omega \subseteq \mathbb{R}^d$, che può essere rappresentato come un ipercubo unitario $[0,1]^d$ senza perdita di generalità. Si vogliono campionare N punti $(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N)$.
L'idea alla base del LHS è suddividere ciascuna dimensione dello spazio in N intervalli uguali e campionare un punto in ciascun intervallo, mantenendo la proprietà di uniformità marginale. Per implementare questa procedura, come spiegato in M. Petelet [2010] è necessario seguire questi step:

1. **Suddivisione degli intervalli**: Ogni dimensione dello spazio è suddivisa in N intervalli di uguale lunghezza. In termini generali, la suddivisione per una dimensione j è rappresentata come:
$$I_{i,j} = \left[\frac{i-1}{N}, \frac{i}{N}\right), \quad i = 1, 2, \ldots, N.$$
2. **Selezione casuale nei sottodomini**: Per ciascun intervallo $I_{i,j}$ di una data dimensione j, viene selezionato un valore $x_{i,j}$ in modo casuale:
$$x_{i,j} = \frac{i-1}{N} + \xi_{i,j} \cdot \frac{1}{N},$$
dove $\xi_{i,j}$ è una variabile casuale uniformemente distribuita su [0,1]. Questo garantisce che il punto $x_{i,j}$ sia contenuto nell'intervallo $I_{i,j}$.
3. **Permutazione delle dimensioni**: Una volta determinati i punti per ogni dimensione, le N coordinate vengono permutate indipendentemente lungo ciascuna dimensione per evitare correlazioni tra le variabili. In altre parole, si genera una permutazione π_j per ogni dimensione j, e i campioni finali sono:
$$\mathbf{x}_i = (x_{i,\pi_1}, x_{i,\pi_2}, \ldots, x_{i,\pi_d}), \quad i = 1, 2, \ldots, N.$$

Questa costruzione assicura che:

- ogni dimensione j abbia distribuzione marginale uniforme su [0,1];
- ogni intervallo $I_{i,j}$ sia occupato esattamente una volta per ciascuna dimensione j, evitando sovrapposizioni e garantendo la proprietà di *space-filling*.

Queste proprietà sono particolarmente utili per costruire modelli surrogati e migliorare l'accuratezza delle simulazioni numeriche.

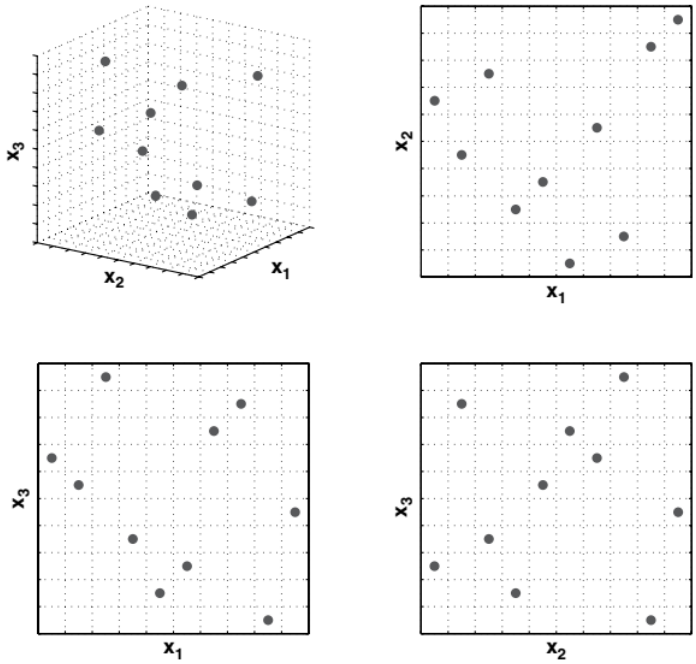


Figura 1.2: Figura 1.4 (pag. 16) tratta da A.I.J. Forrester [2008]. Rappresentazione di un campionamento Latin Hypercube Sampling (LHS) con tre variabili e dieci punti. L'immagine mostra i punti campionati nello spazio tridimensionale (in alto a sinistra), accompagnati dalle relative proiezioni bidimensionali. In ciascuna proiezione, tutti i dieci punti risultano visibili, e ogni riga e colonna delle celle contiene esattamente un punto, garantendo una distribuzione uniforme.

Effetti dell'imposizione di vincoli Il Latin Hypercube Sampling (LHS) può essere combinato con vincoli sulle variabili per garantire che i campioni rispettino condizioni specifiche. Tuttavia, l'applicazione di vincoli successiva al campionamento può compromettere la proprietà di *space-filling* del LHS, alterando la copertura uniforme dello spazio. In particolare, se i vincoli impongono alle variabili di rimanere entro determinati intervalli o di soddisfare relazioni specifiche, i punti inizialmente distribuiti in modo uniforme potrebbero risultare concentrati in alcune regioni dello spazio, lasciando altre aree scoperte. Questo effetto può ridurre significativamente l'efficacia del campionamento nello spazio Ω.

Per affrontare queste problematiche, si possono adottare approcci modificati, come il *Constrained Latin Hypercube Sampling (cLHS)*, descritto in M. Petelet [2010] che incorpora direttamente i vincoli nella fase di generazione dei punti, evitando così di compromettere significativamente la qualità del campionamento.

Un'alternativa efficace, intuitiva e più semplice, sempre accennata in M. Petelet [2010], consiste nel campionare inizialmente le variabili indipendenti dalle altre utilizzando l'LHS e, successivamente, campionare le variabili dipendenti in modo iterativo, basandosi sui valori già assegnati alle variabili indipendenti. Ad esempio, se una variabile dipende da

un'altra già campionata, il campionamento viene effettuato in base ai valori precedentemente generati, rispettando i vincoli noti tra le variabili. Questo metodo consente di preservare, per quanto possibile, la distribuzione uniforme dello spazio nelle fasi iniziali del campionamento.

Questi approcci, tuttavia, presuppongono che i vincoli tra le variabili di input siano noti a priori. Qualora, invece, le relazioni di dipendenza tra le variabili non siano note, si è costretti a procedere con un campionamento standard utilizzando il LHS per tutte le variabili. In questo scenario, sarà il simulatore a gestire le eventuali dipendenze tra le variabili, verificando se i vincoli sono soddisfatti o meno per ciascun punto campionato. Per esempio il simulatore potrebbe restituire non solo il valore della funzione obiettivo, ma anche un valore associato alla violazione del vincolo.

Capitolo 2

L'algoritmo OPUS: Analisi e confronto con i metodi di ricerca diretta

Tra i diversi approcci alternativi ai metodi di ricerca diretta, emergono metodi che sfruttano l'utilizzo di modelli surrogati e tecniche di ottimizzazione bayesiana per ridurre il numero di valutazioni costose della funzione obiettivo. In questo contesto, si è scelto di implementare l'algoritmo *Optimization by Particle Swarm Using Surrogates (OPUS)*, tradotto in italiano come *Ottimizzazione tramite Sciame di Particelle con Modelli Surrogati*, presentato nel lavoro di ricerca Regis [2014]. Tale algoritmo rappresenta un ibrido efficace ed efficiente tra i metodi di ricerca diretta, noti per la loro capacità di esplorare in maniera esaustiva lo spazio di ricerca, e i metodi basati su modelli surrogati, che offrono un'approssimazione della funzione obiettivo al fine di ottimizzare il processo di esplorazione.

Nell'algoritmo OPUS, il modello surrogato viene costruito e aggiornato iterativamente, con l'obiettivo di ridurre il numero di valutazioni dirette della funzione obiettivo, mantenendo al contempo un buon equilibrio tra esplorazione e sfruttamento. Questo approccio consente di limitare i costi computazionali associati alle valutazioni della funzione obiettivo, tipici dei metodi di ricerca diretta, preservando comunque un'elevata qualità della soluzione.

In questo capitolo, si descriverà nel dettaglio il funzionamento dell'algoritmo OPUS, evidenziandone le caratteristiche principali. Inoltre, verranno effettuati confronti pratici con i metodi di ricerca diretta per verificare e quantificare i vantaggi dell'algoritmo OPUS, ponendo particolare attenzione ai limiti dei metodi di ricerca diretta già discussi nell'introduzione della tesi. In particolare, i metodi di *Direct Search* presi in esame per il confronto saranno il *Differential Evolution (DE)* e il *Particle Swarm Optimization (PSO)*. Si procede innanzitutto con la descrizione dettagliata dei due metodi di ricerca diretta.

2.1 Metodi di Ricerca Diretta

2.1.1 Particle Swarm Optimization (PSO)

L'algoritmo *Particle Swarm Optimization* (PSO) è una tecnica di ottimizzazione globale ispirata al comportamento sociale osservabile nei gruppi di animali, come gli stormi di uccelli o i banchi di pesci. L'idea alla base del PSO consiste nel simulare l'interazione collettiva di questi gruppi per trovare la soluzione ottimale di un problema di ottimizzazione, sfruttando entrambe la componente sociale e individuale delle particelle di uno sciame.

Concetti Fondamentali: Nel PSO, lo spazio di ricerca è esplorato da uno *sciame* composto da s particelle. Ogni particella rappresenta una *soluzione candidata* al problema e possiede due caratteristiche principali:

- una *posizione* $\mathbf{x}^i(t) \in \mathbb{R}^d$, che descrive il punto nello spazio in cui la particella si trova all'iterazione t,
- una *velocità* $\mathbf{v}^i(t) \in \mathbb{R}^d$, che determina il cambiamento nella posizione della particella tra due iterazioni consecutive.

Ogni particella registra inoltre la sua miglior posizione conosciuta, $\mathbf{y}^i(t)$, e l'intero sciame mantiene traccia della miglior posizione globale, $\hat{\mathbf{y}}(t)$.

Aggiornamento delle Particelle: L'evoluzione dello sciame avviene attraverso un aggiornamento iterativo delle velocità e delle posizioni delle particelle. In particolare, la *velocità* $\mathbf{v}^i(t+1)$ viene aggiornata considerando tre componenti principali:

1. **Inerzia**: $\mathbf{v}^i(t)$, che rappresenta il movimento della particella sulla base della velocità precedente, ponderato da un fattore d'inerzia $i(t)$;
2. **Componente Cognitiva**: un termine proporzionale alla distanza tra la posizione corrente $\mathbf{x}^i(t)$ e la miglior posizione individuale $\mathbf{y}^i(t)$, scalato dal parametro di cognizione μ e da un valore casuale $\boldsymbol{\omega}_1^i(t) \sim U([0,1]^d)$;
3. **Componente Sociale**: un termine proporzionale alla distanza tra la posizione corrente $\mathbf{x}^i(t)$ e la miglior posizione globale $\mathbf{y}(t)$, scalato dal parametro sociale ν e da un valore casuale $\boldsymbol{\omega}_2^i(t) \sim U([0,1]^d)$.

L'aggiornamento della velocità è quindi descritto dalla seguente equazione:

$$v_j^i(t+1) = i(t)v_j^i(t) + \mu\omega_{1,j}^i(t)(y_j^i(t) - x_j^i(t)) + \nu\omega_{2,j}^i(t)(\hat{y}_j(t) - x_j^i(t)) \tag{2.1}$$

dove:

- $v_j^i(t+1)$ è la componente j-esima della velocità della particella i all'iterazione $t+1$;
- $x_j^i(t)$ e $y_j^i(t)$ rappresentano, rispettivamente, la componente j-esima della posizione corrente e della miglior posizione personale;

- $\hat{y}_j(t)$ è la miglior posizione globale nella dimensione j.

Le velocità aggiornate vengono poi limitate nei confini specificati, $[v_{\min}, v_{\max}]$, per evitare che le particelle si spostino troppo rapidamente:

$$v_j^i(t+1) = \min\left(\max(v_j^i(t+1), v_{\min}\right) \tag{2.2}$$

Dopo aver calcolato la nuova velocità, la posizione della particella viene aggiornata con:

$$\mathbf{x}^i(t+1) = \mathbf{x}^i(t) + \mathbf{v}^i(t+1) \tag{2.3}$$

Infine, la posizione viene proiettata nei limiti dello spazio di ricerca, $[\mathbf{a}, \mathbf{b}]$, per garantire che le particelle non escano dai confini.

Criteri di Arresto: L'algoritmo PSO continua ad iterare fino al raggiungimento di un criterio di arresto specifico. Tipicamente, questi criteri includono:

- il raggiungimento di un numero massimo di iterazioni, $T_{\max}$;
- l'assenza di miglioramenti significativi nella miglior posizione globale per un determinato numero di iterazioni.

Lo pseudocodice dell'algoritmo PSO è tratto da Regis [2014] ed è riportato in 1.

2.1.2 Differential Evolution (DE)

L'algoritmo *Differential Evolution* (DE) è una tecnica di ottimizzazione globale progettata per affrontare problemi complessi in spazi di ricerca continui. Si basa su un approccio stocastico iterativo che utilizza operazioni di mutazione, crossover e selezione per esplorare e sfruttare lo spazio delle soluzioni. Il DE è noto per la sua semplicità e per la capacità di trovare soluzioni robuste anche in presenza di funzioni non lineari e non derivabili.

Concetti Fondamentali: Nell'algoritmo DE, una popolazione di NP individui rappresenta il set di *soluzioni candidate* al problema. Ogni individuo è caratterizzato da:

- una *posizione* $\mathbf{x}^i(t) \in [\mathbf{a}, \mathbf{b}] \subseteq \mathbb{R}^d$, che rappresenta una possibile soluzione al problema all'iterazione t.

L'algoritmo tiene traccia della miglior soluzione corrente, $\mathbf{x}_{\text{best}}(t)$, che minimizza la funzione obiettivo $f(\mathbf{x})$. La popolazione evolve iterativamente per migliorare la qualità delle soluzioni.

Operazioni Principali: L'algoritmo DE si basa su tre operazioni principali che vengono applicate a ogni individuo della popolazione in ciascuna iterazione:

1. **Mutazione**: Per ogni individuo della popolazione, chiamato individuo target $\mathbf{x}^i(t)$, viene generato un *vettore mutato* $\mathbf{v}^i(t)$ combinando casualmente due altri individui della popolazione con l'individuo migliore corrente $\mathbf{x}_{\mathbf{best-candidate}}(t)$:

$$\mathbf{v}^i(t) = \mathbf{x}_{\mathbf{best-candidate}}(t) + F \cdot (\mathbf{x}^{n_1}(t) - \mathbf{x}^{n_2}(t)), \tag{2.4}$$

   dove:

- n_1, n_2 sono indici distinti tra loro e diversi dall'indice associato all'individuo target, essi sono scelti casualmente tra $1, \ldots, NP$;
- $F \in [0,2]$ è un parametro di scala che controlla l'intensità della mutazione.

2. **Crossover**: Il *vettore mutato* viene combinato con l'individuo target $\mathbf{x}^i(t)$ per generare un *vettore trial* $\mathbf{y}^i(t)$:

$$y_j^i(t) = \begin{cases} v_j^i(t), & \text{se } r_j < CR, \\ x_j^i(t), & \text{altrimenti,} \end{cases} \tag{2.5}$$

   dove:

   - $CR \in [0,1]$ è la probabilità di crossover;
   - $r_j \sim U([0,1])$ è un valore casuale per la dimensione j.

3. **Selezione**: Il *vettore trial* viene confrontato con l'individuo target sulla base della funzione obiettivo $f(\mathbf{x})$:

$$\mathbf{x}^i(t+1) = \begin{cases} \mathbf{y}^i(t), & \text{se } f(\mathbf{y}^i(t)) < f(\mathbf{x}^i(t)), \\ \mathbf{x}^i(t), & \text{altrimenti.} \end{cases} \tag{2.6}$$

   Se il vettore trial migliora il valore della funzione obiettivo, sostituisce l'individuo target nella popolazione.

Proiezione ai Limiti: Dopo ogni operazione, di mutazione e di crossover, i vettori che escono dai limiti dello spazio di ricerca $[\mathbf{a}, \mathbf{b}]$ vengono riportati nei confini ammissibili:

$$\mathbf{v}^i(t) = \min(\max(\mathbf{v}^i(t), \mathbf{a}), \mathbf{b}) \tag{2.7}$$

$$\mathbf{y}^i(t) = \min(\max(\mathbf{y}^i(t), \mathbf{a}), \mathbf{b}) \tag{2.8}$$

Criteri di Arresto: L'algoritmo termina quando soddisfa uno dei seguenti criteri:

- viene raggiunto un numero massimo di iterazioni, $T_{\max}$;
- l'algoritmo non mostra miglioramenti significativi nella funzione obiettivo per un certo numero di iterazioni consecutive.

Lo pseudocodice dell'algoritmo DE è riportato in 2.

Algorithm 1: Particle Swarm Optimization (PSO) Regis [2014]

Input: Funzione $f : [\mathbf{a}, \mathbf{b}] \subseteq \mathbb{R}^d \to \mathbb{R}$ da minimizzare, dimensione della popolazione s, posizioni iniziali $\mathbf{x}^1(0), \ldots, \mathbf{x}^s(0) \in [\mathbf{a}, \mathbf{b}]$, fattore d'inerzia $i(t)$ per ogni iterazione t, parametro cognitivo μ, parametro sociale ν, limiti di velocità $\mathbf{v}_{\min}, \mathbf{v}_{\max}$, e numero massimo di iterazioni $T_{\max}$.

Output: La miglior posizione trovata dall'algoritmo, la sua valutazione, il numero di iterazioni e il numero di valutazioni.

Funzione `PSO`**:**

- **Passo 1: Valutare le posizioni iniziali dello sciame**
- **for** $i = 1, \ldots, s$ **do**
  - Calcola $f(\mathbf{x}^i(0))$
- **Passo 2: Determinare le velocità iniziali delle particelle**
- **for** $i = 1, \ldots, s$ **do**
  - Genera $\mathbf{u}^i$ come un valore casuale distribuito uniformemente in $[\mathbf{a}, \mathbf{b}]$
  - Imposta $\mathbf{v}^i(0) = \frac{1}{2}(\mathbf{u}^i - \mathbf{x}^i(0))$
- **Passo 3: Inizializzare le migliori posizioni per ciascuna particella e la migliore globale**
- **for** $i = 1, \ldots, s$ **do**
  - Imposta $\mathbf{y}^i(0) = \mathbf{x}^i(0)$
- Sia $\hat{\mathbf{y}}^i(0)$ la posizione con il valore più piccolo di f tra $\{\mathbf{y}^1(0), \ldots, \mathbf{y}^s(0)\}$
- Imposta il contatore delle iterazioni $t = 0$
- **while** $t < T_{\max}$ **do**
  - **Passo 4: Aggiornare le velocità delle particelle**
  - **for** $i = 1, \ldots, s$ **do**
    - **for** $j = 1, \ldots, d$ **do**
      - Aggiorna la componente di velocità:
      - $v^i_j(t+1) = i(t)v^i_j(t) + \mu\omega^i_{1,j}(t)(y^i_j(t) - x^i_j(t)) + \nu\omega^i_{2,j}(t)(\hat{y}_j(t) - x^i_j(t))$
      - dove $\omega^i_{1,j}(t), \omega^i_{2,j}(t) \sim U([0,1])$
      - Limita la velocità:
      - $v^i_j(t+1) = \min\left(\max(v^i_j(t+1), v_{\min}), v_{\max}\right)$
  - **Passo 5: Aggiornare le posizioni delle particelle**
  - **for** $i = 1, \ldots, s$ **do**
    - Aggiorna la posizione:
    - $\mathbf{x}^i(t+1) = \mathbf{x}^i(t) + \mathbf{v}^i(t+1)$
    - $\mathbf{x}^i(t+1) = \text{proj}_{[\mathbf{a},\mathbf{b}]}(\mathbf{x}^i(t+1)) = \min\left(\max(\mathbf{x}^i(t+1), \mathbf{a}), \mathbf{b}\right)$
    - dove la proiezione garantisce che le particelle rimangano nel dominio
  - **Passo 6: Valutare le posizioni dello sciame**
  - **for** $i = 1, \ldots, s$ **do**
    - Calcola $f(\mathbf{x}^i(t+1))$
  - **Passo 7: Aggiornare le migliori posizioni**
  - **for** $i = 1, \ldots, s$ **do**
    - **if** $f(\mathbf{x}^i(t+1)) < f(\mathbf{y}^i(t))$ **then**
      - Aggiorna la miglior posizione individuale: $\mathbf{y}^i(t+1) = \mathbf{x}^i(t+1)$
      - **if** $f(\mathbf{y}^i(t+1)) < f(\hat{\mathbf{y}}^i(t))$ **then**
        - Aggiorna la miglior posizione globale: $\hat{\mathbf{y}}^i(t+1)) = \mathbf{y}^i(t+1)$
    - **else**
      - Mantieni la miglior posizione precedente: $\mathbf{y}^i(t+1) = \mathbf{y}^i(t)$
  - **Passo 8: Controllare la condizione di terminazione**
  - Se i criteri di stop non sono soddisfatti, incrementa l'iterazione: $t \leftarrow t + 1$
- **return** $\hat{\mathbf{y}}$,$f(\hat{\mathbf{y}})$,t,$s + s * t$

Algorithm 2: Differential Evolution (DE)

Input: Funzione obiettivo $f : [\mathbf{a}, \mathbf{b}] \subseteq \mathbb{R}^d \to \mathbb{R}$ da minimizzare, Popolazione iniziale $\mathbf{x} = \{\mathbf{x}^1, \mathbf{x}^2, \ldots, \mathbf{x}^{NP}\}$, Limiti inferiori $\mathbf{a}$ e superiori $\mathbf{b}$, Numero di individui nella popolazione NP, Probabilità di crossover CR, fattore di mutazione F, numero massimo di iterazioni $T_{\max}$.

Output: La miglior posizione trovata dall'algoritmo, la sua valutazione, il numero di iterazioni e il numero di valutazioni.

Funzione `DE_Optimizer`**:**

- **Passo 1: Inizializzazione**
- Imposta il contatore delle iterazioni $t = 0$
- Calcola il valore iniziale della funzione obiettivo: $f_{\text{best}} = f(\mathbf{x}^1)$
- Imposta $\mathbf{x}_{\text{best}} = \mathbf{x}^1$ come il miglior individuo iniziale
- **while** $t < T_{\max}$ **do**
  - **Passo 2: Mutazione e Crossover**
  - **for** $i = 1, \ldots, NP$ **do**
    - Identifica $\mathbf{x}_{\text{best-candidate}}$ come il miglior individuo nella popolazione corrente
    - Seleziona 2 individui casuali distinti $\mathbf{x}_2, \mathbf{x}_3$ dalla popolazione
    - Calcola il vettore mutato $\mathbf{v}$ secondo la regola:

$$\mathbf{v} = \mathbf{x}_{\text{best-candidate}} + F \cdot (\mathbf{x}_1 - \mathbf{x}_2) \tag{2.9}$$

    - Proietta il vettore $\mathbf{v}$ nei limiti:

$$\mathbf{v} = \min(\max(\mathbf{v}, \mathbf{a}), \mathbf{b}) \tag{2.10}$$

    - Genera il vettore trial $\mathbf{y}$ tramite crossover binario:

$$y_j = \begin{cases} v_j & \text{se } r_j < CR \\ x_{i,j} & \text{altrimenti} \end{cases} \tag{2.11}$$

      dove $r_j \sim U([0,1])$ è un numero casuale uniforme.
    - Proietta il vettore $\mathbf{y}$ nei limiti:

$$\mathbf{y} = \min(\max(\mathbf{y}, \mathbf{a}), \mathbf{b}) \tag{2.12}$$

    - **Passo 3: Selezione**
    - **if** $f(\mathbf{y}) < f(\mathbf{x}_i)$ **then**
      - Sostituisci l'individuo corrente: $\mathbf{x}_i = \mathbf{y}$
      - **if** $f(\mathbf{y}) < f_{best}$ **then**
        - Aggiorna il miglior valore globale:
        - $\mathbf{x}_{\text{best}} = \mathbf{y}$
        - $f_{\text{best}} = f(\mathbf{y})$
  - Incrementa il contatore:
  - $t = t + 1$
- **Passo 4: Controllare la condizione di terminazione**
- Se i criteri di stop non sono soddisfatti, incrementa l'iterazione: $t \leftarrow t + 1$
- **Passo 5: Calcolo finale**
- Valuta la funzione obiettivo per ogni individuo nella popolazione finale:
- $f_{\text{eval}} = \{f(\mathbf{x}^1), f(\mathbf{x}^2), \ldots, f(\mathbf{x}^{NP})\}$
- Identifica il miglior individuo nella popolazione finale:
- $\mathbf{x}_{\text{best}} = \mathbf{x}^{i_{\text{best}}}$ dove $i_{\text{best}} = \arg\min(f_{\text{eval}})$
- Calcola il miglior valore della funzione obiettivo: $f_{\text{best}} = \min(f_{\text{eval}})$
- **return** $\mathbf{x}_{\text{best}}, f_{\text{best}}, t, NP \cdot t + NP$

2.2 Optimization by Particle Swarm Using Surrogates (OPUS)

L'algoritmo OPUS (*Optimization by Particle Swarm Using Surrogates*) combina tecniche di ottimizzazione per sciami di particelle (PSO, *Particle Swarm Optimization*) con un modello surrogato basato su funzioni a base radiale (RBF, *Radial Basis Functions*). Questo approccio ibrido sfrutta le capacità di esplorazione globale del PSO insieme alla capacità del modello surrogato di fornire stime accurate della funzione obiettivo, riducendo il numero di valutazioni computazionalmente costose, rendendo OPUS particolarmente adatto a problemi di ottimizzazione complessi. L'approccio è stato presentato e descritto in dettaglio in Regis [2014].

2.2.1 Panoramica dell'Algoritmo OPUS

L'algoritmo OPUS può essere descritto attraverso i seguenti passaggi principali, in accordo con lo pseudocodice presentato nei **Passi 1-12** dell'Algoritmo 3 e 4, tratto da Regis [2014]:

- **Valutazione iniziale dello *space-filling design*** (*Passo 1*): L'algoritmo inizia con la valutazione della funzione obiettivo $f(\mathbf{z}^i)$ per un set iniziale di punti $\{\mathbf{z}^1, \ldots, \mathbf{z}^k\}$, generati tramite uno *space-filling design* (ad esempio, un Latin hypercube design). Questi punti coprono in modo uniforme lo spazio delle soluzioni $[\mathbf{a}, \mathbf{b}] \subseteq \mathbb{R}^d$ e costituiscono la base iniziale per la costruzione del modello surrogato.
- **Inizializzazione delle particelle dello sciame** (*Passi 2-4*):
  - Le posizioni iniziali dello sciame $\{\mathbf{x}^i(0)\}_{i=1}^s$ vengono selezionate tra i punti dello *space-filling design* con i valori più bassi di f.
  - Le velocità iniziali delle particelle, indicate come $\mathbf{v}^i(0)$, sono generate calcolando la metà della differenza tra un insieme di punti casuali $\{\mathbf{u}^i\}$ distribuiti nello spazio delle soluzioni e le posizioni iniziali delle particelle, ovvero:

$$\mathbf{v}^i(0) = \frac{1}{2}\left(\mathbf{u}^i - \mathbf{x}^i(0)\right). \tag{2.13}$$

  - Inoltre, si inizializzano le migliori posizioni individuali per ogni particella $\mathbf{y}^i(0)$ e la posizione globale migliore $\hat{\mathbf{y}}(0)$.
- **Costruzione del modello surrogato** (*Passo 5*): Il modello surrogato $s_t(\mathbf{x})$, basato su funzioni RBF, viene costruito utilizzando tutti i punti valutati fino a quel momento, $\{\mathbf{x}^i(t)\}$. Il modello approssima la funzione obiettivo f e guida la ricerca durante le fasi successive.
- **Ottimizzazione iterativa con PSO e surrogato** (*Passi 6-9*): Durante ogni iterazione, analogamente al *Particle Swarm Optimization* (PSO) tradizionale, le particelle aggiornano le proprie posizioni e velocità sfruttando due componenti principali: la componente cognitiva, basata sulla migliore posizione individuale precedentemente

esplorata e la componente sociale, basata sulla migliore posizione globale identificata all'interno dell'intera popolazione.

Tuttavia, a differenza del PSO convenzionale, l'algoritmo OPUS introduce un modello surrogato per stimare la qualità delle nuove posizioni delle particelle. In particolare, per ciascuna particella vengono generate l particelle di prova, le cui posizioni vengono inizialmente valutate mediante il modello surrogato. Tra queste, solo la particella di prova con il miglior valore stimato viene successivamente valutata utilizzando la funzione obiettivo reale. Più nello specifico:

- Durante ogni iterazione, vengono generate nuove velocità di prova $\mathbf{v}^{i,\ell}(t+1)$ e posizioni di prova $\mathbf{x}^{i,\ell}(t+1)$ per ciascuna particella i, utilizzando la regola di aggiornamento:

$$v_j^{i,\ell}(t+1) = i(t)v_j^i(t) + \mu\omega_{1,j}^{i,\ell}(t)\big(y_j^i(t) - x_j^i(t)\big) + \nu\omega_{2,j}^{i,\ell}(t)\big(\hat{y}_j(t) - x_j^i(t)\big), \quad (2.14)$$

  dove $\omega_{1,j}^{i,\ell}(t), \omega_{2,j}^{i,\ell}(t) \sim U([0,1])$, e la velocità è limitata da $\mathbf{v}_{\min}$ e $\mathbf{v}_{\max}$.
- Il modello surrogato $s_t(\mathbf{x})$, basato su *Radial Basis Functions* (RBF), è progettato per fornire una stima approssimativa della funzione obiettivo per le nuove posizioni esplorate dalle particelle. Esso viene utilizzato per selezionare la posizione di prova $\mathbf{x}^i(t+1)$ più promettente, che viene successivamente valutata tramite la funzione obiettivo reale f. Questo approccio consente di ridurre significativamente il numero di valutazioni dirette della funzione obiettivo reale. In questo modo, è possibile preservare l'accuratezza dell'ottimizzazione minimizzando al contempo il costo computazionale. Il modello surrogato, quindi, viene aggiornato con le nuove posizioni valutate, rimuovendo eventuali duplicati.

- **Affinamento locale** (*Passo 10*): Nella fase di affinamento locale, dopo ogni iterazione, un solver di ottimizzazione viene utilizzato per individuare un minimo globale approssimato $\mathbf{x}_{t+1}^*$ del modello surrogato $\tilde{s}_t(\mathbf{x})$ all'interno di una regione delimitata da un ipercubo di lato prefissato ξ, centrato nel miglior punto globale attuale $\hat{\mathbf{y}}(t+1)$.

- **Aggiornamento del modello surrogato e valutazioni** (*Passo 11*):
  - Se il minimo stimato dal surrogato $\mathbf{x}_{t+1}^*$ è sufficientemente distante ($> \delta$) dai punti già valutati, viene calcolata $f(\mathbf{x}_{t+1}^*)$ e aggiunta al set di valutazioni esistenti. Questo consente di migliorare ulteriormente la qualità delle soluzioni trovate.
  - Il modello surrogato viene aggiornato con le nuove posizioni valutate, rimuovendo eventuali duplicati.

- **Criteri di arresto** (*Passo 12*): L'algoritmo termina quando viene raggiunto il numero massimo di iterazioni $T_{\max}$ o quando non si osservano miglioramenti significativi per un certo numero di iterazioni consecutive.

2.2.2 Scelta del modello surrogato e dei parametri

La scelta del modello surrogato e dei parametri riveste un ruolo fondamentale per garantire l'efficacia e l'efficienza dell'algoritmo OPUS-RBF. In questa sezione vengono illustrate le motivazioni e le scelte effettuate.

Modello surrogato basato su *Radial Basis Functions*

Come suggerito in Regis [2014], per costruire il modello surrogato, è stato adottato un approccio basato sulle *Radial Basis Functions* (RBF). Dati n punti distinti $\mathbf{u}^{(1)}, \ldots, \mathbf{u}^{(n)} \in \mathbb{R}^d$ e i corrispondenti valori della funzione $f(\mathbf{u}^{(1)}), \ldots, f(\mathbf{u}^{(n)})$, il modello OPUS-RBF utilizza un interpolante della forma:

$$s(\mathbf{x}) = \sum_{i=1}^{n} \lambda_i \phi(\|\mathbf{x} - \mathbf{u}^{(i)}\|) + p(\mathbf{x}), \quad \mathbf{x} \in \mathbb{R}^d, \tag{2.15}$$

dove $\|\cdot\|$ rappresenta la norma euclidea, $\lambda_i \in \mathbb{R}$ per $i = 1, \ldots, n$, $p(\mathbf{x})$ è un polinomio lineare in d variabili e $\phi(r)$ è definita come una funzione radiale cubica: $\phi(r) = r^3$.
Altre possibili scelte per la funzione $\phi(r)$ includono la *thin plate spline*, le forme *multi-quadric* e *Gaussian*. Tuttavia, è stato privilegiato un modello RBF cubico poiché studi precedenti (Regis [2014], cita ad esempio B.Y. Qu [2012], R.G. Regis [2004]) ne hanno evidenziato l'efficacia, la robustezza e l'affidabilità nell'interpolazione di superfici complesse. Inoltre, in B. Wei [2012] si suggerisce che le RBF cubiche possano essere più adatte rispetto alle RBF Gaussiane in contesti di ottimizzazione basata su modelli surrogati. In R.G. Regis [2004] si evidenzia, inoltre, che nel caso delle RBF cubiche, a differenza delle Gaussiane, non è necessario stimare parametri aggiuntivi, rendendo il modello più semplice.

Per costruire il modello RBF cubico sopra definito, si fa riferimento alla parte teorica del Capitolo 1 e a Regis [2014], in particolare la matrice $\Phi \in \mathbb{R}^{n \times n}$ è definita come:

$$\Phi_{ij} = \phi(\|\mathbf{u}^{(i)} - \mathbf{u}^{(j)}\|), \quad i, j = 1, \ldots, n. \tag{2.16}$$

Inoltre, si definisce la matrice $P \in \mathbb{R}^{n \times (d+1)}$, in cui la i-esima riga è data da $[1, (\mathbf{u}^{(i)})^T]$. Si ricorda inoltre che il modello RBF cubico che interpola i punti $(\mathbf{u}^{(1)}, f(\mathbf{u}^{(1)})), \ldots, (\mathbf{u}^{(n)}, f(\mathbf{u}^{(n)}))$ viene ottenuto risolvendo il seguente sistema lineare:

$$\begin{bmatrix} \Phi & P \\ P^T & \mathbf{0}_{(d+1)\times(d+1)} \end{bmatrix} \begin{bmatrix} \boldsymbol{\lambda} \\ \mathbf{c} \end{bmatrix} = \begin{bmatrix} \mathbf{F} \\ \mathbf{0}_{d+1} \end{bmatrix}, \tag{2.17}$$

dove $\mathbf{F} = [f(\mathbf{u}^{(1)}), \ldots, f(\mathbf{u}^{(n)})]^T \in \mathbb{R}^n$, $\boldsymbol{\lambda} = [\lambda_1, \ldots, \lambda_n]^T \in \mathbb{R}^n$, $\mathbf{c} = [c_1, \ldots, c_{d+1}]^T \in \mathbb{R}^{d+1}$ rappresenta i coefficienti del polinomio lineare $p(\mathbf{x})$, e $\mathbf{0}_{(d+1)\times(d+1)}$ è una matrice nulla di dimensione $(d+1) \times (d+1)$.

Si ricorda ancora che la matrice dei coefficienti del sistema è invertibile se e solo se il rango di P è pari a $d+1$. Questa condizione è soddisfatta se tra i punti $\{\mathbf{u}^{(1)}, \ldots, \mathbf{u}^{(n)}\}$ esiste un sottoinsieme di $d+1$ punti affinamente indipendenti. Un insieme di punti $\{\mathbf{y}^{(0)}, \mathbf{y}^{(1)}, \ldots, \mathbf{y}^{(k)}\}$ in $\mathbb{R}^d$ è affinamente indipendente se le differenze $\mathbf{y}^{(1)} - \mathbf{y}^{(0)}, \mathbf{y}^{(2)} - \mathbf{y}^{(0)}, \ldots, \mathbf{y}^{(k)} - \mathbf{y}^{(0)}$ sono linearmente indipendenti.

Parametri dell'algoritmo

Per la scelta dei parametri dell'algoritmo, si è deciso di adottare in gran parte le impostazioni proposte nel lavoro Regis [2014], poiché tali parametri sono stati impiegati in diversi esperimenti descritti nel suddetto lavoro, ottenendo risultati computazionali soddisfacenti. In particolare, l'inizializzazione delle particelle avviene tramite un *Latin Hypercube Design* (LHD) composto da $d+1$ punti, che rappresentano il numero minimo di punti necessario per costruire il modello surrogato, garantendo così una buona copertura iniziale dello spazio di ricerca. La popolazione iniziale delle particelle viene scelta come sottoinsieme dei punti LHD che mostrano i migliori valori della funzione obiettivo. Il numero di particelle da mantenere dipende dal caso d'uso e può quindi essere scelto come un giusto compromesso tra l'efficacia della soluzione e il costo computazionale.

I parametri inerziali e sociali sono stati selezionati come segue: il fattore di inerzia è stato fissato a $i(t) = 0.72984$, mentre i parametri cognitivo e sociale sono stati impostati entrambi a $\mu = \nu = 1.496172$. Questi valori sono tipicamente impostati in questo modo nell'algoritmo PSO.

Nel paper Regis [2014], i valori massimi e minimi per le componenti dei vettori velocità sono stati rispettivamente impostati a:

$$-\min_{1\le i\le d}\frac{b_i-a_i}{4}, \quad \min_{1\le i\le d}\frac{b_i-a_i}{4}. \tag{2.18}$$

dove $[\mathbf{a},\mathbf{b}]$ sono i limiti associati al dominio delle variabili; la lunghezza del lato del box per il raffinamento locale, invece, è stata impostata a $\xi = 0.1\min_{1\le i\le d}(b_i-a_i)$.

Nel lavoro citato, i domini associati alle funzioni test erano omogenei, ovvero ogni variabile apparteneva allo stesso dominio. Tuttavia, in un contesto più generale, le variabili possono appartenere a domini molto diversi tra loro e potrebbero essere presenti anche variabili binarie. Per tale motivo, si è scelto di non adottare le stesse impostazioni del paper per quanto riguarda le velocità minima e massima, nonché la lunghezza del lato del box per il raffinamento locale. Tali parametri sono stati, pertanto, impostati come segue:

$$v_{\min,i} = -\frac{(b_i-a_i)}{4} \quad v_{\max,i} = \frac{(b_i-a_i)}{4} \tag{2.19}$$

$$\xi_i = 0.1(b_i-a_i) \tag{2.20}$$

Per quanto riguarda il numero di posizioni di prova per ogni particella, esso è stato fissato a $r = 10d$, come nel lavoro citato. La distanza soglia per determinare quando un punto è troppo vicino a un altro è stata fissata a:

$$\delta = 0.0005\sqrt{d}\min_{\substack{1\le i\le d\\ \text{i non binaria}}}(b_i-a_i). \tag{2.21}$$

A differenza di quanto fatto in Regis [2014], nella selezione degli indici delle variabili per il calcolo del minimo, vengono considerati esclusivamente gli indici che non corrispondono a variabili binarie. Questa scelta è motivata dall'esigenza di evitare che il valore di δ risulti eccessivamente piccolo.

La scelta di questi parametri consente di raggiungere un equilibrio ottimale tra esplorazione dello spazio di ricerca e sfruttamento.

Algorithm 3: Optimization by Particle Swarm Using Surrogates (OPUS): Step 1-6 Regis [2014]

Input: Funzione da minimizzare: $f : [\mathbf{a}, \mathbf{b}] \to \mathbb{R}$, dove $[\mathbf{a}, \mathbf{b}] \subseteq \mathbb{R}^d$,
Dimensione della popolazione: s,
Space-filling design: $\{\mathbf{z}^1, \ldots, \mathbf{z}^k\} \subseteq [\mathbf{a}, \mathbf{b}]$ con $k \geq s$
Parametri PSO ($i(t)$, μ, ν, $\mathbf{v}_{\min}$, $\mathbf{v}_{\max}$),
Numero di posizioni di prova per ogni particella: r,
Modello surrogato (ad esempio, RBF cubico con coda polinomiale lineare),
Lunghezza del lato della scatola per il raffinamento locale: ξ,
Solver di ottimizzazione per il raffinamento locale,
Distanza soglia per determinare se un punto è troppo vicino ad un altro punto: δ,
Numero massimo di iterazioni: $T_{\max}$.

Output: La miglior posizione trovata dall'algoritmo, la sua valutazione, il numero di iterazioni e il numero di valutazioni.

Passo 1 (Valutazione dei punti dello space-filling design):
for $i = 1, \ldots, k$ **do**
  Calcola $f(\mathbf{z}^i)$.
Passo 2 (Determinazione delle posizioni iniziali dello sciame):
Scegli le posizioni iniziali dello sciame $\mathbf{x}^1(0), \ldots, \mathbf{x}^s(0)$ come gli s punti di $\{\mathbf{z}^1, \ldots, \mathbf{z}^k\}$ con i valori migliori della funzione obiettivo.
Passo 3 (Determinazione delle velocità iniziali delle particelle):
for $i = 1, \ldots, s$ **do**
  (3a) Genera $\mathbf{u}^i$ casualmente in $[\mathbf{a}, \mathbf{b}]$.
  (3b) Imposta $\mathbf{v}^i(0) = \frac{1}{2}(\mathbf{u}^i - \mathbf{x}^i(0))$
Passo 4 (Inizializzazione della miglior posizione per ciascuna particella e della miglior posizione globale):
Imposta $\mathbf{y}^i(0) = \mathbf{x}^i(0)$, $i = 1, \ldots, s$, e imposta $\hat{\mathbf{y}}(0)$ come il punto in $\{\mathbf{y}^1(0), \ldots, \mathbf{y}^s(0)\}$ con il valore minimo della funzione f.
Inoltre, imposta $\mathcal{E}_0 = \emptyset$ e $t = 0$.
Passo 5 (Costruzione del modello surrogato):
Usa tutti i punti valutati $\left(\bigcup_{j=0}^{t} \bigcup_{i=1}^{s} \{\mathbf{x}^i(j)\}\right) \cup \mathcal{E}_t$ per costruire un modello surrogato $s_t(\mathbf{x})$ che approssimi la funzione obiettivo.
Passo 6 (Determinazione delle nuove posizioni delle particelle):
for $i = 1, \ldots, s$ **do**
  (6a) Generazione delle velocità di prova:
  for $\ell = 1, \ldots, r$ **do**
    for $j = 1, \ldots, d$ **do**

$$v_j^{i,\ell}(t+1) = i(t)v_j^i(t) + \mu\omega_{1,j}^{i,l}(t)(y_j^i(t) - x_j^i(t)) + \nu\omega_{2,j}^{i,l}(t)(\hat{y}_j(t) - x_j^i(t)).$$

      dove: $\omega_{1,j}^{i,l}(t), \omega_{2,j}^{i,l}(t) \sim U([0,1])$
      $v_j^{i,\ell}(t+1) = \min(\max(v_{j,\min}, v_j^{i,\ell}(t+1)), v_{j,\max})$.
    Genera le posizioni di prova:
    $\mathbf{x}^{i,\ell}(t+1) = \mathbf{x}_i(t) + \mathbf{v}_i^{\ell}(t+1)$.
    Proietta le posizioni di prova nei limiti $[\mathbf{a}, \mathbf{b}]$:
    $\mathbf{x}^{i,\ell}(t+1) = \min(\max(\mathbf{a}, \mathbf{x}^{i,\ell}(t+1)), \mathbf{b})$.
  (6b) Selezione della posizione più promettente usando il surrogato:
  Usa il modello surrogato $s_t(\mathbf{x})$ per stimare la posizione più promettente tra i punti $\{\mathbf{x}^{i,1}(t+1), \mathbf{x}^{i,2}(t+1), \ldots, \mathbf{x}^{i,r}(t+1)\}$.
  Imposta $\mathbf{x}^i(t+1)$ come posizione stimata migliore e $\mathbf{v}^i(t+1)$ come la velocità corrispondente.

Algorithm 4: Optimization by Particle Swarm Using Surrogates (OPUS): Passi 7-12 Regis [2014]

Passo 7 (Valutazione delle nuove posizioni dello sciame):
for $i = 1, \ldots, s$ **do**
    Calcola $f(\mathbf{x}^i(t+1))$.
Passo 8 (Aggiornamento della miglior posizione per ciascuna particella e della miglior posizione globale):
for $i = 1, \ldots, s$ **do**
    if $f(\mathbf{x}^i(t+1)) < f(\mathbf{y}^i(t))$ **then**
        Imposta $\mathbf{y}^i(t+1) = \mathbf{x}^i(t+1)$.
        if $f(\mathbf{y}^i(t+1)) < f(\hat{\mathbf{y}}(t))$ **then**
            Imposta $\hat{\mathbf{y}}(t+1) = \mathbf{y}^i(t+1)$
    else
        Imposta $\mathbf{y}^i(t+1) = \mathbf{y}^i(t)$.
Passo 9 (Riadattamento del modello surrogato):
Usa tutti i punti valutati

$$\left(\bigcup_{j=0}^{t+1} \bigcup_{i=1}^{s} \{\mathbf{x}^i(j)\} \right) \cup \mathcal{E}_t$$

per costruire un nuovo modello surrogato $\tilde{s}_{t+1}(\mathbf{x})$ per la funzione obiettivo.
Passo 10 (Raffinamento locale della miglior posizione globale):
Usa il solver di ottimizzazione per un raffinamento locale per trovare un punto di minimo globale $\mathbf{x}^*_{t+1}$ del surrogato $\tilde{s}_{t+1}(\mathbf{x})$ all'interno del box

$$[\hat{\mathbf{y}}(t+1) - \xi/2, \hat{\mathbf{y}}(t+1) + \xi/2] \cap [\mathbf{a}, \mathbf{b}].$$

Passo 11 (Determinare se il punto di minimo del surrogato è sufficientemente lontano dai punti precedenti):
if $\mathbf{x}^*_{t+1}$ *è distante almeno* δ *da tutti i punti precedentemente valutati* **then**
    (11a) Valutazione del punto di minimo del surrogato: Calcola $f(\mathbf{x}^*_{t+1})$.
    (11b) Aggiornamento della miglior posizione globale:
    if $f(\mathbf{x}^*_{t+1}) < f(\hat{\mathbf{y}}(t+1))$ **then**
        Imposta $\hat{\mathbf{y}}(t+1) = \mathbf{x}^*_{t+1}$.
    (11c) Aggiornamento dei punti di raffinamento locale:
    Imposta $\mathcal{E}_{t+1} = \mathcal{E}_t \cup \{\mathbf{x}^*_{t+1}\}$.
else
    (11d) Mantenimento dei punti di raffinamento locale:
    Imposta $\mathcal{E}_{t+1} = \mathcal{E}_t$.
Passo 12 (Controllo del criterio di terminazione):
Se i criteri di arresto sono soddisfatti, termina.
Altrimenti, incrementa $t \leftarrow t+1$ e torna al Passo 5.
return $\hat{\mathbf{y}}$,$f(\hat{\mathbf{y}})$,t,$s + s * t$

2.3 Analisi Comparativa dei Metodi Implementati su una Funzione Test

2.3.1 Confronto metodi

Di seguito, nelle tabelle [2.1, 2.2, 2.3] si riportano tutti i risultati sperimentali ottenuti utilizzando rispettivamente tre diversi algoritmi di ottimizzazione: Particle Swarm Optimization (PSO), Differential Evolution (DE) e Optimization by Particle Swarm Using Surrogates (OPUS). Ciascun metodo è stato applicato allo stesso problema di ottimizzazione, con l'obiettivo di valutare le loro prestazioni in termini di accuratezza della soluzione, tempo di convergenza e numero di valutazioni di funzione costosa. La funzione su cui sono testati i tre metodi è la funzione di **Ackley**, rappresentata in figura 1.1a, una funzione non lineare multi-modale, tale funzione, infatti, è caratterizzata da molti minimi locali e un minimo globale, localizzato nell'origine. Queste caratteristiche la rendono utile per valutare la capacità di un algoritmo di trovare soluzioni ottimali in spazi di ricerca complessi, con più minimi locali. Tutti i metodi sono stati implementati utilizzando il linguaggio di programmazione Python.

Nelle tabelle [2.1, 2.2, 2.3] sono riportate tutte le combinazioni di dominio, dimensione dello spazio di ricerca e numero di particelle utilizzate per i tre metodi analizzati. In ciascuna tabella, sono indicati i risultati associati a ogni combinazione testata, includendo:

- il punto di minimo trovato;
- il valore della funzione obiettivo associato al minimo;
- il numero di iterazioni necessarie affinché il metodo converga;
- il numero di valutazioni della funzione obiettivo;
- il tempo computazionale impiegato dal metodo.

In particolare, i risultati associati al metodo PSO, sono stati ottenuti utilizzando i seguenti valori di parametri:

- Il numero di particelle s è stato impostato in base alla dimensione del problema e all'ampiezza dello spazio di ricerca;
- il fattore di inerzia è stato impostato a un valore costante $i(t) = i = 0.72984$
- il parametro cognitivo è stato impostato come $\mu = 1.496172$:
- il parametro sociale è stato imposto pari a $\nu = 1.496172$;
- le velocità minima e massima sono state impostate rispettivamente pari a $v_{\min,i} = -(b_i - a_i)/4$ e $v_{\max,i} = (b_i - a_i)/4$ per ogni componente $i = 1, \ldots, d$, dove $[\mathbf{a}, \mathbf{b}] \subseteq \mathbb{R}^d$ denota lo spazio di ricerca.

Per quanto riguarda il metodo DE, invece, sono stati utilizzati i seguenti parametri:

- Il numero di individui nella popolazione è stato impostato nella maggior parte dei casi a $NP = 15$, ovvero al valore standard utilizzato solitamente. Tuttavia, in alcuni casi, in funzione della dimensione del problema e dell'ampiezza dello spazio di ricerca, è stato necessario aumentare tale valore per garantire la convergenza del metodo;
- la probabilità di crossover $CR \in [0,1]$ è stata impostata a $CR = 0.5$;
- il fattore di mutazione $F \in [0,2]$ è stato impostato a $F = 0.7$, ovvero al valore standard associato a questo parametro.

Per quanto riguarda il metodo OPUS, i parametri condivisi con il metodo PSO sono stati impostati nello stesso modo. Tuttavia, nella Tabella 2.3, invece di riportare unicamente la dimensione del problema, nella seconda colonna è indicata una coppia di valori: il primo rappresenta il numero di punti iniziali campionati k, mentre il secondo indica la dimensione del problema d.

Relativamente ai parametri aggiuntivi specifici del metodo OPUS — ovvero il numero di particelle di prova r, la lunghezza del lato del box per il raffinamento locale ξ e la distanza soglia δ utilizzata per determinare quando due punti sono troppo vicini tra loro — essi sono stati impostati come descritto nel paragrafo 2.2.2.

In tutti e tre i metodi, i punti iniziali sono stati campionati utilizzando la tecnica del *Latin Hypercube Sampling* all'interno dello spazio di ricerca. Il numero massimo di iterazioni è stato fissato in tutti i casi a $T_{max} = 1000$. Tuttavia, in alcuni dei casi in dimensione più alta ($d = 15$ e $d = 30$) analizzati con il metodo OPUS, il numero massimo di iterazioni è stato ridotto al fine di ottenere un giusto compromesso tra il tempo computazionale e l'efficacia del metodo.

Dai risultati sperimentali ottenuti, si sono potute trarre le seguenti considerazioni ed effettuare confronti tra i tre metodi.

- **Accuratezza:** I metodi PSO e DE ottengono valori leggermente più accurati rispetto a OPUS in termini di valore della funzione obiettivo. Tuttavia, tutti e tre i metodi forniscono risultati di elevata qualità, poiché in ogni caso i punti ottenuti coincidono con il minimo globale della funzione di Ackley, situato nell'origine.
- **Valutazioni di Funzione:** OPUS richiede un numero significativamente minore di valutazioni di funzione rispetto a PSO e DE, il che diventa particolarmente utile quando una singola valutazione della funzione è molto costosa, come nel caso di simulazioni complesse. Come evidenziato nelle tabelle 2.1, 2.2 e 2.3, un caso significativo in cui è stato utilizzato lo stesso numero massimo di iterazioni, pari a $T_{max} = 1000$, per tutti e tre i metodi e lo stesso criterio di terminazione anticipata, è rappresentato dalla combinazione di uno spazio di dimensione $d = 5$ e un dominio $[a_i, b_i] = [-15, 20]$ per ogni variabile i nello spazio. In questo caso, per raggiungere la convergenza:
  - il metodo **PSO** richiede **3318** valutazioni della funzione obiettivo;
  - il metodo **DE** richiede **4290** valutazioni della funzione obiettivo;

  - il metodo **OPUS**, invece, richiede solamente **759** valutazioni della funzione obiettivo, ossia circa un quinto e un sesto, rispettivamente, del numero di valutazioni effettuate da PSO e DE.

- **Tempo Computazionale:** Con l'aumentare della complessità del problema, il tempo computazionale richiesto da OPUS per convergere tende ad aumentare più rapidamente rispetto a PSO e DE. Tuttavia, l'analisi del tempo computazionale deve considerare due fattori contrastanti:

  - da un lato, il numero di valutazioni della funzione obiettivo diminuisce drasticamente rispetto a PSO e DE, comportando una riduzione del tempo computazionale;
  - dall'altro, aumenta la complessità del modello surrogato utilizzato per approssimare la funzione obiettivo costosa, determinando un incremento del tempo computazionale, come evidenziato nella colonna $Time(s)$ della Tabella 2.3.

  Nel contesto dell'ottimizzazione di un modello di simulazione, in cui la funzione obiettivo non è espressa in forma analitica ma implementata tramite un simulatore, ridurre il numero di valutazioni della funzione è cruciale, poiché queste risultano essere estremamente costose. Di conseguenza, è altamente probabile che il primo dei due fattori sopra menzionati prevalga sul secondo, determinando una riduzione complessiva del tempo computazionale.

In conclusione, considerando gli aspetti evidenziati precedentemente, OPUS si presenta come la scelta più vantaggiosa nel contesto in cui una singola valutazione della funzione è particolarmente costosa dal punto di vista computazionale, come nel caso della *Simulation-Based Optimization*. OPUS offre, infatti, un netto vantaggio in termini di utilizzo delle risorse e tempo computazionale, senza compromettere la precisione e l'efficienza del metodo.

Risultati per Particle Swarm Optimization (PSO)

s	*Dim.*	Domain	Best_x	Best_f	Iterations	Func. Evals	Time (s)
5	3	[-15, 20]	$[-9.9446e-16, -1.1075e-15, -8.3701e-16]$	$3.9968e-15$	375	1880	0.1300
5	3	[-5, 5]	$[2.0900e-15, 6.4928e-15, 6.1559e-16]$	$1.4655e-14$	347	1740	0.1100
5	3	[-3, 3]	$[8.4668e-16, 3.1411e-16, -2.0624e-15]$	$3.9968e-15$	353	1770	0.1100
7	5	[-15, 20]	$[1.9101e-16, 1.7770e-16, 3.3607e-16, 1.2310e-16, 2.1590e-16]$	$4.4409e-16$	473	3318	0.3600
7	5	[-5, 5]	$[-1.0822e-15, 1.5337e-15, -8.1975e-16, -1.3655e-15, -6.0471e-16]$	$3.9968e-15$	471	3304	0.3200
7	5	[-3, 3]	$[-1.8863e-15, -1.2507e-15, -4.3949e-16, 1.5355e-15, 8.4623e-16]$	$3.9968e-15$	463	3248	0.3400
15	15	[-15, 20]	$[-1.2541e-15, 3.6665e-15, 1.9024e-16, -7.7142e-15, 4.1567e-15, -1.1116e-15, 5.8141e-15, 8.5458e-16, -1.7081e-15, -2.3447e-15, -1.9497e-15, 1.3548e-15, 6.7540e-15, -2.8632e-15, 6.2105e-15]$	$1.4655e-14$	837	12570	3.3800
20	15	[-5, 5]	$[-2.8518e-15, -5.1220e-15, 9.4557e-16, 1.7536e-15, -4.8653e-16, 4.7223e-15, -6.3561e-15, -3.9095e-15, 1.5357e-14, 1.4153e-15, 2.9801e-15, 3.8153e-16, 2.2920e-15, -6.4751e-15, -2.9253e-15]$	$2.1760e-14$	777	15560	4.7400
15	15	[-3, 3]	$[2.1133e-15, 8.5941e-14, 4.8527e-15, 2.9152e-15, 3.0088e-15, 1.4922e-14, -7.9197e-15, -1.6024e-14, 3.4258e-14, 6.8839e-15, 4.4863e-15, -1.2061e-14, 1.3151e-14, 2.1217e-15, -4.5690e-15]$	$9.9920e-14$	847	12720	4.1900
100	30	[-15, 20]	$[2.1990e-15, 3.3434e-15, 3.9601e-15, -1.1506e-15, 1.6937e-15, -3.4460e-15, 6.7142e-15, 3.9468e-15, -9.3375e-15, 4.3611e-15, 8.4778e-15, -1.7897e-15, 6.2730e-16, -6.3257e-16, 4.5793e-15, -6.1679e-15, 2.2508e-16, 3.7096e-16, 1.2123e-14, -1.4815e-14, 2.8121e-15, -2.0297e-15, -7.1485e-15, 5.7747e-16, 8.2071e-15, 4.2089e-15, 8.1057e-15, 3.7834e-15, -7.0987e-15, 3.2769e-15]$	$2.1760e-14$	947	94800	46.1500
70	30	[-5, 5]	$[1.1829e-14, 1.3684e-14, 8.3316e-15, 4.7444e-15 -7.6951e-17, 3.7146e-16, -1.3880e-14, 8.2244e-15 -3.5715e-15, -7.6630e-15, -6.7587e-15, 8.5881e-15 1.4602e-14, -2.9692e-15, -9.4881e-15, 1.3680e-14 -3.3468e-15, -2.3213e-15, -3.9160e-15, 2.4846e-15 2.5141e-14, 6.4583e-15, -1.2563e-14, -2.2664e-15 2.2923e-14, -6.8258e-15, -3.3677e-15, 2.0187e-15 5.9419e-16, 2.1838e-15]$	$3.9524e-14$	1000	70070	34.08
30	30	[-3, 3]	$[-4.0143e-10, 3.5588e-10, 5.1082e-10, -2.4807e-09 -1.1827e-09, -2.1825e-09, 8.8171e-10, 1.8302e-09 -1.0261e-09, 1.9768e-09, -7.1200e-11, 2.6878e-09 1.0177e-09, 2.5062e-10, 5.9642e-10, -8.3365e-10 -6.6975e-09, 3.8586e-10, 2.1084e-09, 8.3065e-12 1.4125e-08, -1.3333e-09, 2.8548e-10, -1.4863e-09 1.5992e-09, -4.6968e-10, -1.8392e-09, -6.5431e-10 1.6204e-10, 6.4892e-11]$	$1.2458e-08$	1000	30030	13.95

Tabella 2.1: Risultati per **PSO**.

Risultati per Differential Evolution (DE)

NP	*Dim.*	Domain	Best_x	Best_f	Iterations	Func. Evals	Time (s)
15	3	[-15, 20]	$[1.2199e-16,\ 2.2460e-16,\ 1.6788e-16]$	$4.4409e-16$	196	2955	0.41
15	3	[-5, 5]	$[-1.0589e-16,\ 1.6383e-16,\ -2.2118e-16]$	$4.4409e-16$	195	2940	0.35
15	3	[-3, 3]	$[-1.9193e-15,\ -1.1690e-15,\ 1.7364e-16]$	$3.9968e-15$	190	2865	0.33
15	5	[-15, 20]	$[4.3049e-16,\ 1.2804e-15,\ 3.9296e-16,$ $-2.4674e-15,\ -3.3511e-16]$	$3.9968e-15$	285	4290	0.58
15	5	[-5, 5]	$[-7.6420e-16,\ 1.0240e-15,\ 1.9645e-16,$ $1.8117e-15,\ 2.4073e-16]$	$3.9968e-15$	276	4155	0.57
15	5	[-3, 3]	$[-1.2474e-16,\ -1.1921e-15,\ 6.5296e-16,$ $-7.5959e-16,\ 2.4254e-15]$	$3.9968e-15$	266	4005	0.55
15	15	[-15, 20]	$[-2.4003e-15,\ -4.6450e-16,\ 4.3037e-15,$ $-8.0902e-16,\ 2.4424e-15,\ -1.0178e-15,$ $1.6275e-15,\ 2.0105e-16,\ -3.4788e-15,$ $3.0682e-15,\ 2.2299e-15,\ 3.9965e-15,$ $2.9780e-15,\ 2.3075e-15,\ 6.5580e-16]$	$7.5495e-15$	703	10560	2.77
16	15	[-5, 5]	$[2.4895e-15,\ 9.0459e-16,\ 2.5886e-15,$ $9.5632e-16,\ 9.0474e-16,\ 2.9856e-15,$ $3.8053e-15,\ 3.1217e-17,\ 1.0864e-15,$ $-5.2963e-16,\ -3.1208e-15,\ -4.4009e-16,$ $-2.2522e-15,\ -2.7067e-15,\ 9.2226e-16]$	$7.5495e-15$	650	10416	2.71
15	15	[-3, 3]	$[-4.3388e-15,\ -5.5860e-16,\ -6.4667e-15,$ $-1.8490e-15,\ -1.2372e-16,\ 9.8311e-16,$ $-1.0229e-15,\ -6.6057e-15,\ 2.7182e-15,$ $-6.3295e-15,\ 4.9849e-16,\ -3.5137e-15,$ $-6.0321e-15,\ 2.8990e-16,\ -5.6672e-15]$	$1.4655e-14$	665	9990	2.36
30	30	[-15, 20]	$[2.1143e-11,\ -2.6590e-11,\ 1.0954e-10,\ -1.4176e-11,$ $5.3857e-11,\ 2.6437e-10,\ -2.8128e-12,\ -1.8971e-10,$ $8.3906e-11,\ 1.1699e-10,\ -4.5627e-11,\ -3.2539e-12,$ $9.2824e-12,\ -4.9532e-11,\ -8.6767e-11,\ 3.5165e-11,$ $2.4962e-11,\ -1.4052e-11,\ -1.9063e-10,\ 1.4735e-10,$ $-7.0463e-11,\ 1.1996e-10,\ 4.5382e-11,\ -5.9482e-11,$ $-2.5700e-11,\ -7.5759e-11,\ -2.7698e-10,\ -2.5583e-12,$ $-1.3134e-10,\ 2.5217e-10]$	$4.6422e-10$	1000	30030	17.51
30	30	[-5, 5]	$[-1.8309e-11,\ 1.4208e-11,\ -8.5175e-13,\ 8.5704e-12,$ $3.2187e-11,\ 2.3558e-12,\ 1.6287e-11,\ -3.4928e-12,$ $-1.9032e-11,\ -6.0867e-12,\ 2.5599e-11,\ 1.2345e-11,$ $-2.1416e-11,\ -1.4823e-11,\ -1.0541e-11,\ -1.3798e-11,$ $-1.1661e-11,\ -3.1388e-11,\ 7.4123e-12,\ -1.4949e-11,$ $1.5827e-12,\ 1.1661e-11,\ 6.2415e-12,\ 6.4197e-12,$ $3.7070e-12,\ 3.6896e-12,\ -1.3050e-11,\ 7.8233e-12,$ $2.0544e-11,\ 3.3204e-12]$	$5.8549e-11$	1000	30030	18.23
30	30	[-3, 3]	$[3.4992e-12,\ 1.6195e-11,\ 5.6453e-12,\ 4.3806e-12,$ $-3.4885e-12,\ -5.5905e-12,\ 6.6877e-12,\ 4.7340e-12,$ $-2.3106e-12,\ -2.9411e-12,\ -2.9325e-12,\ -7.1354e-12,$ $-7.7392e-12,\ 3.8288e-12,\ -3.2942e-13,\ -1.4593e-12,$ $1.0662e-12,\ 2.3985e-12,\ 3.3998e-12,\ 1.6464e-12,$ $-1.1226e-11,\ -3.2722e-12,\ -2.8453e-12,\ 1.1405e-11,$ $3.2781e-13,\ -2.8728e-12,\ -7.3641e-12,\ -1.6878e-12,$ $5.3639e-13,\ 2.1617e-13]$	$4.2197e-11$	1000	30030	17.43

Tabella 2.2: Risultati per **DE**.

Risultati per Optimization by Particle Swarm Using Surrogates (OPUS)

s	z_shape	Domain	Best_x	Best_f	Iterations	Func. Evals	Time (s)
5	(20, 3)	[-15, 20]	$[2.2180e-08, -1.1082e-08, -2.6102e-08]$	$8.31e-08$	112	677	26.64
5	(20, 3)	[-5, 5]	$[9.8554e-07, 1.0673e-06, 3.1297e-07]$	$3.43e-06$	67	407	11.12
20	(20, 3)	[-3, 3]	$[-4.2990e-08, -8.1435e-09, -9.4831e-09]$	$1.03e-07$	100	2120	81.71
7	(20, 5)	[-15, 20]	$[-4.4479e-07, -3.5271e-07, 1.5478e-07, 7.5357e-07, -1.2402e-06]$	$2.80e-06$	94	759	35.09
7	(20, 5)	[-5, 5]	$[2.8257e-08, 4.9594e-08, -1.8830e-07, -2.0609e-07, -2.5275e-07]$	$6.81e-07$	84	679	28.41
20	(20, 5)	[-3, 3]	$[-1.9703e-07, -3.2119e-07, -2.9048e-07, 1.2221e-07, 1.7603e-07]$	$9.33e-07$	69	1469	44.36
15	(30, 15)	[-15, 20]	$[-4.6409e-06, -1.7563e-06, 4.2644e-06, 8.1482e-07, 1.6954e-06, 4.6620e-06, -1.0271e-05, -3.0205e-06, 4.7424e-06, -2.0995e-06, -4.7400e-06, -2.8882e-06, 2.4801e-06, -1.0402e-06, 8.2095e-06]$	$1.83e-05$	100	1616	283.68
15	(30, 15)	[-5, 5]	$[1.4178e-07, 1.3484e-06, 7.0699e-07, -9.6807e-07, -1.1044e-06, 7.3478e-07, 1.4469e-06, 2.4256e-07, -4.6879e-07, -4.7286e-07, -9.3235e-07, -7.3823e-08, -2.7682e-06, 1.1859e-06, 4.9786e-08]$	$4.33e-06$	100	1616	286.97
30	(30, 15)	[-3, 3]	$[-8.8939e-07, -1.8479e-05, -9.6836e-06, -1.3666e-06, -2.1609e-06, 4.9452e-06, 2.4929e-09, -1.4295e-05, -5.7935e-06, -1.0016e-06, 1.2601e-05, 1.3882e-05, -1.5524e-05, -2.7266e-05, 2.0263e-05]$	$5.12e-05$	50	1580	142.71
30	(31, 30)	[-15, 20]	$[4.9238e-06, -5.0213e-07, -1.3939e-06, -8.0193e-06, -1.6556e-05, 1.1112e-05, -6.4107e-06, 1.8453e-05, 3.5857e-06, 1.2717e-05, 1.4299e-05, 5.8089e-06, 2.1447e-05, -8.7921e-06, -6.0509e-08, -1.2566e-05, 6.3974e-06, 1.6603e-05, -3.2697e-05, -7.0196e-06, -1.1730e-06, -5.8397e-07, 1.0703e-05, -7.3934e-06, 1.5939e-06, -5.6150e-06, -2.9233e-06, 4.7608e-06, -1.8270e-05, -8.8716e-06]$	$4.65e-05$	100	3131	1406.34
30	(31, 30)	[-5, 5]	$[-1.0664e-06, -2.4126e-06, 4.3157e-06, 9.2006e-07, 9.7502e-07, -5.9439e-06, -1.4755e-06, 2.0436e-06, -4.2349e-08, 3.5259e-06, -5.0001e-06, -9.9807e-07, -7.9262e-07, 3.5545e-06, 1.2453e-06, -2.8803e-06, -3.7855e-06, -2.8217e-06, 4.7436e-06, 1.8574e-06, -7.1392e-07, -4.2455e-07, -3.9200e-06, 3.6861e-07, -3.8349e-07, 2.2016e-09, -1.0566e-07, -1.5893e-06, -1.2943e-06, 5.8750e-07]$	$1.03e-05$	100	3131	1392.47
50	(50, 30)	[-3, 3]	$[7.8101e-05, -1.3192e-04, -2.3084e-05, -4.9948e-06, 2.1214e-05, -1.0875e-05, -9.9306e-05, 4.7878e-05, -1.1631e-04, 8.5497e-05, 2.8143e-05, -8.6891e-05, 5.5940e-05, 2.0809e-05, -1.7871e-04, 1.4113e-04, -1.1389e-04, -8.7933e-06, -1.8632e-04, 7.4359e-05, -5.8293e-05, 5.9317e-05, 9.8540e-05, 1.0724e-05, 9.6683e-05, -9.5972e-05, -7.6211e-05, -2.4248e-05, -1.1523e-04, -2.3282e-05]$	$3.51e-04$	50	2600	664.00

Tabella 2.3: Risultati per **OPUS**.

Capitolo 3

Applicazione di OPUS a un modello semplificato di supply-chain: risultati e performance

3.1 Struttura e dinamiche del modello di simulazione della supply-chain: black-box a parametri noti

Il simulatore rappresenta un modello complesso di gestione delle scorte all'interno di una catena di supply-chain, operando come una black-box che accetta in input una serie di parametri predefiniti e restituisce un output significativo per il processo decisionale.
Il modello di supply chain adottato coinvolge una rete di punti vendita (PdV) interconnessi a un unico centro di distribuzione (DC), in cui ogni prodotto è associato a un unico fornitore. Ogni PdV può decidere se approvvigionarsi direttamente dai fornitori o tramite il centro di distribuzione, in funzione delle previsioni di vendita, della domanda e dei costi operativi.

Le variabili di input del modello includono:

- **Parametri continui:** questi includono valori essenziali per la gestione ottimale delle scorte di ogni prodotto. Per ogni prodotto all'interno del Centro di Distribuzione (DC) è necessario decidere tre valori associati alle seguenti variabili:
  - **Safety stock DC (scorta di sicurezza):** è il livello minimo di inventario che un centro di distribuzione deve mantenere per far fronte a eventuali fluttuazioni nella domanda o ritardi di approvvigionamento. Garantisce che ci sia sempre una quantità sufficiente di prodotto disponibile per evitare esaurimenti delle scorte (stockout).

  - **Reorder level DC (livello di riordino):** è il livello critico sotto il quale viene attivato il processo di riordino. Quando l'inventario di un prodotto scende al di sotto di questo livello, viene effettuato un nuovo ordine per rifornire le scorte, evitando il raggiungimento di livelli critici.
  - **Max level DC (livello massimo):** rappresenta la quantità massima di scorte che può essere mantenuta per un determinato prodotto. Serve a evitare costi eccessivi di giacenza o deterioramento del prodotto, mantenendo un livello di inventario efficiente. In particolare, quando le scorte scendono sotto il livello di riordino, viene ordinata una quantità pari alla differenza tra il *max level* e il *reorder level*.

  Inoltre, per ogni combinazione di prodotto e punto di vendita (PdV), è necessario scegliere il valore associato alla seguente variabile:

  - **Safety stock item-store (scorta di sicurezza per prodotto-PdV):** rappresenta il livello minimo di inventario che, per ogni coppia prodotto-PdV, deve essere mantenuto per far fronte a eventuali fluttuazioni nella domanda o ritardi di approvvigionamento. Garantisce che ci sia sempre una quantità sufficiente di prodotto disponibile per evitare stockout.

- **Variabili binarie/intere:** per ogni punto di vendita (PdV), è necessaria una decisione binaria riguardo alla fonte di approvvigionamento dei prodotti. In particolare, è necessario stabilire se il punto di vendita debba ordinare i prodotti direttamente dal fornitore o tramite il centro di distribuzione.

L'output del simulatore è composto dai seguenti elementi:

- **f:** il valore della funzione obiettivo, rappresentante i costi totali da minimizzare, inclusi i costi di approvvigionamento, di giacenza, di *stockout* e di gestione delle scorte all'interno della supply chain.

- **g:** un valore che rappresenta la percentuale totale di *stockout* relativa ai punti vendita e funge da indicatore per verificare il rispetto di specifici vincoli, come il mantenimento di un livello minimo di servizio. Generalmente, si impone che tale percentuale rimanga inferiore al 5%, al fine di garantire un adeguato livello di disponibilità delle scorte. Per garantire il rispetto rigoroso di tale vincolo, è possibile adottare la strategia di includere nella funzione obiettivo un termine aggiuntivo, che venga penalizzato con alti coefficienti di penalità qualora la percentuale di *stockout* superi la soglia del 5%, come mostrato nell'equazione (1.3) del Capitolo 1.

3.1.1 Interazione tra modello di simulazione e ottimizzatore (OPUS)

L'obiettivo principale del processo di ottimizzazione è minimizzare i costi totali, garantendo al contempo il rispetto di tutti i vincoli. In particolare, vengono considerati i vincoli di tipo *box* associati alle variabili e i vincoli di servizio, che implicano il mantenimento di un livello minimo di disponibilità delle scorte, corrispondente al vincolo di mantenere

la percentuale totale di *stockout* inferiore al 5%. L'algoritmo di ottimizzazione (OPUS) interagisce iterativamente con il simulatore, come illustrato nella figura 3.1. In particolare, l'algoritmo fornisce nuovi punti al simulatore, che li valuta restituendo le performance associate a ciascuna configurazione, permettendo così un continuo aggiornamento dei parametri di input.

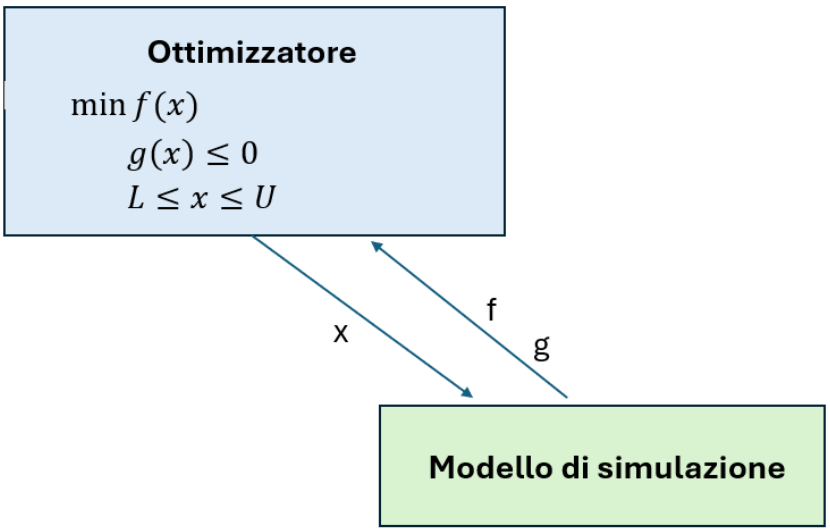


Figura 3.1: Interazione tra modello di simulazione e ottimizzatore.

É importante sottolineare che sulle variabili di input presentate precedentemente, sono imposti dei vincoli di tipo *box*, che limitano i possibili valori che tali variabili possono assumere:

- **Variabili binarie:** queste variabili sono trattate come variabili continue nel processo di ottimizzazione. Tuttavia, al momento in cui vengono passate al simulatore, i valori vengono arrotondati all'intero più vicino, garantendo che esse assumano esclusivamente i valori {0,1}.
- **Variabili continue:** per quanto riguarda i tre livelli (*max level*, *reorder level*, e *safety stock*) per ogni prodotto all'interno del Centro di Distribuzione (DC) e le *safety stock* per ogni prodotto all'interno dei punti vendita (PdV), i vincoli sono i seguenti:
  - **Lower bound:** per tutte queste variabili, il lower bound è fissato a 0, rappresentando il valore minimo che ogni variabile continua può assumere.
  - **Upper bound:** l'upper bound per ciascuna variabile è determinato dalla capacità di magazzino per quel determinato prodotto, sia nel centro di distribuzione (DC) che nel punto vendita (PdV).

Inoltre, esistono vincoli impliciti per i tre livelli (*max level*, *reorder level*, e *safety stock*) per ogni prodotto all'interno del DC, in particolare essi devono soddisfare la seguente condizione:

$$safety\ stock \leq reorder\ level \leq max\ level$$

ovvero il livello di riordino deve essere minore o uguale al livello massimo e contemporaneamente, il livello delle scorte di sicurezza deve essere a sua volta minore o uguale del livello di riordino.

Questi vincoli vengono gestiti internamente dal simulatore, che provvede a ripristinarli qualora non siano rispettati, modificando di conseguenza i valori di input forniti. Al fine di evitare che il simulatore intervenga autonomamente nella gestione dei vincoli, i punti iniziali vengono generati rispettando sia i vincoli di tipo *box* sia le relazioni tra i tre livelli del centro di distribuzione (DC), utilizzando il metodo *Latin Hypercube Sampling* (LHS) con vincoli integrati, come descritto nel paragrafo 1.2.3 del Capitolo 1.

Per quanto riguarda gli step successivi dell'ottimizzazione, in particolare:

- dopo la generazione di ogni particella di prova nello *step 6* dell'Algoritmo 3;
- durante la fase di ottimizzazione locale del modello surrogato nello *step 11* dell'Algoritmo 4;

tutti i punti generati vengono inizialmente riportati all'interno dei limiti definiti dai vincoli di tipo box e successivamente viene utilizzata una funzione che, per ogni nuovo punto, modifica l'input in modo da garantire il rispetto dei vincoli sulle scorte. Tale correzione viene effettuata attraverso i seguenti passaggi:

1. Se il *reorder level* supera il *max level*, il *reorder level* viene ridotto al valore del *max level*.
2. Successivamente, se il *safety stock* supera il *reorder level*, il *safety stock* viene impostato pari al valore del *reorder level*.

In conclusione, grazie all'interazione continua tra l'algoritmo di ottimizzazione e il simulatore, si è in grado di migliorare progressivamente le decisioni operative nella catena di supply-chain.

3.2 Modello semplificato di supply-chain: struttura e risultati sperimentali

In questa sezione vengono presentati i principali risultati ottenuti applicando l'algoritmo di ottimizzazione OPUS a un modello semplificato.

Il modello semplificato è stato progettato per ridurre la complessità della *supply chain*, limitando l'analisi a un singolo prodotto e a un unico punto di vendita (denominato *store*), in presenza di un solo centro di distribuzione (*DC*). Questa configurazione permette di focalizzare l'attenzione sugli aspetti fondamentali della gestione delle scorte e delle

decisioni logistiche, eliminando le difficoltà derivanti dalla presenza di molteplici prodotti e punti vendita.

I parametri ottimizzati all'interno di questo modello sono i seguenti:

1. Una variabile binaria che indica la modalità di rifornimento per l'unico store:
   - valore 1, se il rifornimento avviene direttamente dal fornitore;
   - valore 0, se il rifornimento avviene tramite il centro di distribuzione (DC).
2. Il parametro *item_store_ss*, che rappresenta la scorta di sicurezza (*safety stock*) del prodotto presso lo *store*.
3. I parametri *safety stock*, *reorder level* e *max level* relativi al DC. Questi parametri vengono utilizzati esclusivamente quando il rifornimento avviene tramite il DC; in caso di rifornimento diretto dal fornitore, tali valori sono impostati a zero.

Una versione generale di un modello di simulazione della *supply chain* può incorporare elementi stocastici, come la distribuzione di probabilità dell'errore sulla previsione della domanda o l'incertezza nei tempi di consegna dei prodotti, la cui conoscenza non è disponibile a priori. Nonostante il modello includa elementi stocastici, l'output della simulazione rimane invariato quando l'esecuzione viene ripetuta utilizzando gli stessi valori dei parametri da ottimizzare.

Nella versione semplificata adottata, il modello è invece trattato come deterministico, trascurando le incertezze relative alla domanda e ai tempi di consegna. L'obiettivo dell'ottimizzazione è minimizzare i costi totali della *supply chain*, includendo i costi di giacenza, di trasporto e quelli associati alla rottura dello stock, ossia all'esaurimento delle scorte. La natura deterministica del modello consente di concentrarsi sugli effetti diretti delle decisioni di ottimizzazione, semplificando l'analisi ed evitando la complessità introdotta dalla gestione delle variabili stocastiche.

Per valutare l'efficacia dell'algoritmo di ottimizzazione adottato, sono stati condotti diversi esperimenti, variando sia i parametri del modello che le condizioni operative. Le principali configurazioni analizzate sono le seguenti:

- **Variazione dei costi di giacenza**: sono stati effettuati test modificando i costi di giacenza sia presso il centro di distribuzione (DC) sia presso lo *store*. L'obiettivo è stato analizzare l'impatto di tali variazioni sulle politiche di approvvigionamento ottimali.
- **Variazione dei costi di ordinazione**: sono stati eseguiti test per valutare come differenti configurazioni dei costi fissi di ordinazione influenzano le decisioni di rifornimento, sia nel caso di approvvigionamento dal DC che direttamente dal fornitore.
- **Assenza di vincoli di riordino**: al fine di semplificare l'interpretazione dei risultati, si è deciso di escludere eventuali vincoli di riordino, come l'obbligo di riempire pallet interi o strati di pallet.

I risultati dei test sono riportati nella Tabella 3.1, che fornisce una sintesi delle configurazioni ottimali individuate e dei costi totali associati a ciascun test.

3.2.1 Analisi dettagliata dei risultati sperimentali

Di seguito viene presentata un'analisi dettagliata di ciascun test, con particolare attenzione alle scelte ottimali effettuate e alle relative implicazioni. Tutti gli esperimenti sono stati condotti utilizzando l'algoritmo di ottimizzazione OPUS e il linguaggio di programmazione Python.

In merito alla selezione del modello surrogato e alla configurazione dei parametri dell'algoritmo, sono state adottate le stesse impostazioni descritte nella sottosezione 2.2.2 del Capitolo 2.

Per quanto riguarda i parametri iniziali del modello semplificato, essi sono stati configurati in modo da riflettere in maniera realistica i valori tipicamente utilizzati nelle *supply chain* reali. Tuttavia, mentre i parametri reali si riferiscono a *supply chain* caratterizzate da una complessità significativamente maggiore, con numerosi prodotti e punti vendita, in questo caso, come descritto in precedenza, l'analisi è stata limitata a un singolo prodotto e un unico punto vendita.

Successivamente, sono state introdotte modifiche significative ai parametri e ulteriori assunzioni sul modello, al fine di essere in grado di prevedere un risultato e confrontarlo con quello effettivamente ottenuto dall'algoritmo di ottimizzazione.

Le assunzioni relative al modello ed i parametri iniziali, successivamente modificati, sono i seguenti:

1. **Modello deterministico**: Il modello è stato configurato in modo da eliminare tutte le incertezze relative alla domanda e ai tempi di consegna, trasformandolo in un modello deterministico. Ciò consente di trovare soluzioni che assicurano una percentuale di stockout pari a zero, garantendo la disponibilità continua del prodotto.

2. **Costo di giacenza al DC**: Il costo di giacenza iniziale al centro di distribuzione è pari a 0.001 euro/unità, riflettendo un valore realistico per il mantenimento delle scorte.

3. **Costo di giacenza allo store**: Il costo di giacenza iniziale presso lo store è pari a 0.0015 euro/unità, leggermente superiore a quello del DC.

4. **Costi fissi di ordinazione e di trasporto**: A differenza del caso di approvvigionamento tramite il DC, in cui i costi totali includono sia i costi di ordinazione che quelli di trasporto, nel caso di rifornimento diretto dal fornitore il costo di trasporto è pari a zero. Questo perché il costo di trasporto è già incluso nel costo fisso di ordinazione dal fornitore. I costi fissi di ordinazione sono pari a 40 euro per gli ordini diretti dallo store al fornitore e a 15 euro per gli ordini dallo store al DC, mentre per quanto riguarda i costi fissi di ordinazione da DC verso fornitore, questi sono pari a 125 euro. I costi fissi di trasporto ammontano a 14,4 euro per ciascun camion che trasporta merce dal DC allo store, questi costi sono calcolati in base alla distanza in chilometri (km) percorsa dal DC allo store e ad un costo fisso di trasporto espresso in euro per chilometro (euro/km).

5. **Vincoli di riempimento di pallet e layer**: Nel caso realistico, sono attivi sia il vincolo di riempimento del pallet che quello dello strato del pallet. In generale,

quando entrambi i vincoli sono presenti, prevale il vincolo di riempimento del pallet completo, poiché esso include implicitamente il vincolo di riempimento dello strato. Questi vincoli si applicano solo nel caso in cui il DC effettui ordini verso il fornitore; non sono invece rilevanti per gli ordini effettuati dallo store verso il DC o direttamente verso il fornitore. Relativamente al vincolo di riempimento del pallet, il numero necessario per completare un pallet è di 24 colli, corrispondenti a 360 unità del prodotto considerato. Per quanto riguarda, invece, il vincolo di riempimento del singolo layer del pallet, la quantità richiesta è di 6 colli, pari a 90 unità del prodotto. Tuttavia, in tutti i test riportati di seguito, tali vincoli sono stati disattivati al fine di rendere le soluzioni più facilmente interpretabili.

6. **Costi fissi di ordinazione**: I costi fissi di ordinazione, già accennati nel punto 4, sono stati definiti come segue:

   - Costo fisso di ordinazione dal DC verso il fornitore: 125 euro
   - Costo fisso di ordinazione dallo store verso il DC: 15 euro
   - Costo fisso di ordinazione dallo store verso il fornitore: 40 euro

   Questi costi fissi influenzano le scelte ottimali di rifornimento; nei test successivi, infatti, sono state effettuate variazioni nei costi di fissi di ordinazione per analizzare l'impatto sulle politiche di approvvigionamento.

7. **Periodo di warm up:** Il modello di simulazione adottato prevede un periodo di warm up della durata di una settimana all'interno di un periodo totale di 35 giorni. Questo periodo iniziale ha lo scopo di portare il sistema dallo stato iniziale a uno stato di regime, eliminando eventuali effetti transitori. È importante sottolineare che il periodo di warm up non viene considerato nel calcolo delle statistiche utilizzate per determinare i costi, il numero di ordini e altre metriche. Infatti, il suo unico scopo è preparare il sistema per l'analisi successiva. Pertanto, per calcolare queste statistiche, si considerano solo i 28 giorni successivi al periodo di warm up. Durante il warm up, viene sempre effettuato almeno un ordine dal centro di distribuzione (DC) verso il fornitore per rifornire il magazzino, che all'inizio è vuoto.

Questi parametri iniziali e le loro modifiche consentono di creare scenari significativi per valutare la capacità dell'ottimizzatore di trovare soluzioni efficaci per la gestione della supply chain. Infine, per i vincoli di tipo box, tutti i limiti inferiori (*lower bound*) sono fissati a zero, come già descritto in 3.1.1. I limiti superiori (*upper bound*) sono invece definiti come segue:

1. L'*upper bound* della variabile binaria è fissato a 1.
2. L'*upper bound* della *safety stock* presso lo store è fissato a 70 unità, valore corrispondente alla capacità massima di magazzino dell'unico store considerato per l'unico prodotto analizzato.
3. Gli *upper bound* relativi ai tre livelli del DC, ovvero *safety stock*, *reorder level* e *max level*, sono tutti fissati a 2000 unità, valore che rappresenta la capacità massima di magazzino del DC per l'unico prodotto considerato.

A differenza dei parametri precedenti, i limiti inferiori (*lower bound*) e superiori (*upper bound*) rimangono invariati durante le sperimentazioni. Essi sono stati definiti con l'obiettivo di facilitare l'interpretazione e l'analisi dei risultati ottenuti.

I test descritti di seguito sono sintetizzati nella Tabella 3.1. Tutti gli esperimenti sono stati condotti utilizzando la stessa configurazione adottata nel test di OPUS con dimensione dello spazio di ricerca $d = 5$ sulla funzione di Ackley (vedi Tabella 2.3), ovvero con $k = 20$ particelle iniziali e $s = 7$ particelle migliori mantenute.
Il test 0 costituisce la base di tutti i test successivi. Un esempio di come l'algoritmo opera nel contesto del caso base è illustrato nella figura 3.2.

Test	Rifornimento	SS Store	SS DC	RL DC	ML DC	Costo Totale (€)	N. Iter.
0	DC	20	11	13	647	479.81	58
1	Fornitore	19	0	0	0	601.85	80
2	DC	6	13	13	658	617.41	100
3	Fornitore	4	0	0	0	721.83	25
4	Fornitore	0	0	0	0	0.97	21
5	Fornitore	0	0	0	0	0.97	21
6	Fornitore	4	0	0	0	601.22	61
7	DC	35	89	94	94	385.19	53

Tabella 3.1: Risultati degli esperimenti sul modello semplificato

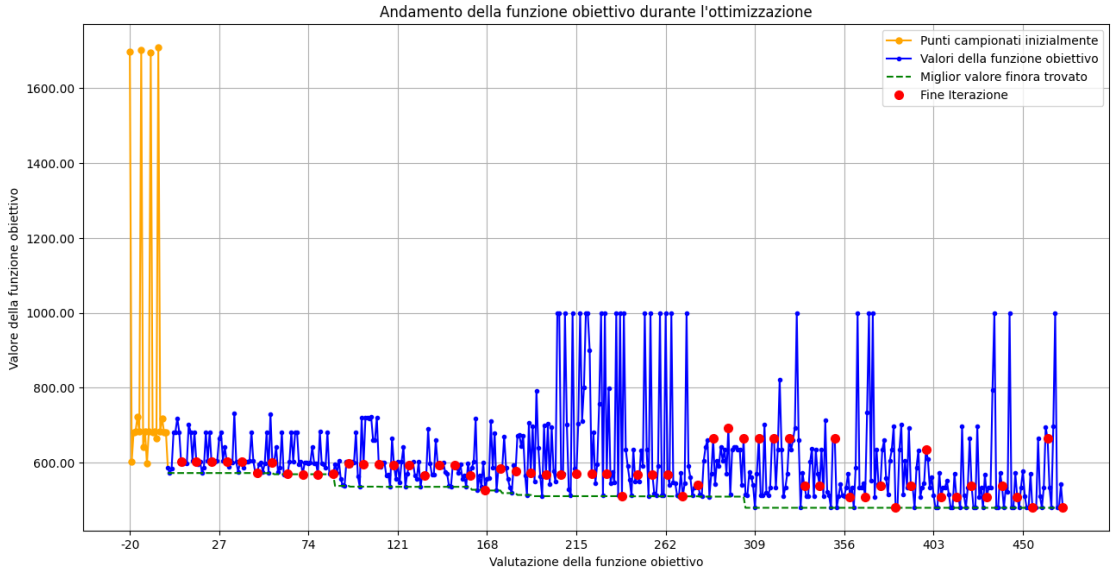


Figura 3.2: Andamento della funzione obiettivo al variare delle iterazioni per il caso base, ovvero per il Test 0. Il grafico è riportato per illustrare il funzionamento del metodo OPUS e fornire una comprensione visiva della sua evoluzione durante il processo iterativo. Si è scelto di rappresentare solo il caso del Test 0 poiché gli altri casi presentano un andamento simile a quello mostrato.

- **Test 0: Caso base**
  L'elenco riportato di seguito rappresenta l'output finale del processo di ottimizzazione e riassume i risultati ottenuti. Ogni voce corrisponde a un parametro chiave che è stato ottimizzato, fornendo così una panoramica delle scelte effettuate dall'algoritmo per minimizzare i costi nella gestione della supply chain.

  – **Rifornimento**: DC
  – **Scorta di sicurezza (SS) Store**: 20
  – **Scorta di sicurezza (SS) DC**: 11
  – **Livello di riordino (RL) DC**: 13
  – **Livello massimo (ML) DC**: 647
  – **Costo Totale**: 479.81
  – **Numero di Iterazioni**: 58

  Il risultato ottimale suggerisce di rifornirsi dal DC, con una scorta di sicurezza per lo store pari a 20 unità, con valori di *safety stock*, *reorder level* molto bassi pari rispettivamente a 11 e 13 unità e *max level* pari a 647 unità.
  Il costo totale è di 479.81 euro, suddiviso in:

  – Costo di giacenza allo store: 1.67
  – Costo di giacenza al DC: 7.75
  – Costo totale degli ordini: 240.0
  – Costo di trasporto: 230.4
  – Costo di stockout: 0.0
  – Num. ordini effettuati dal DC: 0
  – Num. ordini effettuati dallo store: 16

  La componente principale del costo totale è rappresentata dai costi di trasporto e degli ordini che costituiscono la parte preponderante, mentre i costi di giacenza al DC e allo store sono considerevolmente minori (2 ordini di grandezza in meno). I livelli bassi di *safety stock*, *reorder level* e il *max level* pari a 647 unità fanno si che non venga effettuato nessun ordine dal DC verso il fornitore (escludendo l'ordine iniziale nel periodo di warm up), questo è coerente con quanto ci si poteva aspettare, in quanto un ordine aggiuntivo comporterebbe il pagamento di un costo fisso di ordinazione pari a 125 euro. Questa configurazione di parametri, allo stesso tempo, permette di mantenere bassi i costi di giacenza al DC. L'assenza di stockout indica che la politica di scorte è sufficiente a evitare interruzioni nella disponibilità del prodotto.

- **Test 1: Aumento del costo di giacenza al DC**
  Il costo di giacenza presso il DC è stato incrementato di due ordini di grandezza, passando da 0,001 euro/unità a 0,1 euro/unità, mentre gli altri parametri sono rimasti invariati. Con un costo di giacenza così elevato, è ragionevole aspettarsi che l'ottimizzatore opti per un rifornimento diretto dal fornitore. In caso contrario, scegliendo di ordinare tramite il DC, si presenterebbero due scenari principali:

1. Mantenere scorte molto basse al DC, il che richiederebbe frequenti ordini al fornitore, comportando un aumento significativo dei costi operativi.
2. Conservare livelli più elevati di scorte al DC, in particolare per i parametri *safety stock*, *reorder level* e *max level*, ma ciò genererebbe un costo di giacenza proibitivo. Ad esempio, mantenendo valori di *safety stock*, *reorder level* e *max level* simili a quelli del caso precedente, il costo di giacenza al DC, che in precedenza era dell'ordine delle unità, crescerebbe notevolmente con l'attuale configurazione dei costi di giacenza, raggiungendo un ordine di grandezza pari alle centinaia.

L'output finale del processo di ottimizzazione è il seguente:

- **Rifornimento**: Fornitore
- **Scorta di sicurezza (SS) Store**: 19
- **Scorta di sicurezza (SS) DC**: 0
- **Livello di riordino (RL) DC**: 0
- **Livello massimo (ML) DC**: 0
- **Costo Totale**: 601.85
- **Numero di Iterazioni**: 80

Il costo totale è di 601.85 euro, suddiviso in:

- Costo di giacenza allo store: 1.85
- Costo di giacenza al DC: 0.0
- Costo totale degli ordini: 600.0
- Costo di trasporto: 0.0
- Costo di stockout: 0.0
- Num. ordini effettuati dal DC: 0
- Num. ordini effettuati dallo store: 15

In questo scenario, la voce che contribuisce consistentemente al costo totale è il costo degli ordini, mentre i costi di trasporto sono assenti poiché il rifornimento è avvenuto direttamente dal fornitore. Il risultato finale, come ci si aspettava, ha portato a una scelta di rifornimento diretto.

- **Test 2: Aumento del costo di giacenza allo store**
  In questo scenario, il costo di giacenza presso lo store è stato incrementato di due ordini di grandezza, passando da 0,0015 euro/unità a 0,15 euro/unità. Con questa nuova configurazione, ci si aspetta un livello di *safety stock* presso lo store inferiore rispetto al caso del Test 0. Questa scelta consentirebbe di minimizzare il contributo del costo di giacenza allo store, il quale ora assume un ordine di grandezza comparabile con quello del costo totale degli ordini e del costo totale di trasporto.

  L'output finale del processo di ottimizzazione è il seguente:

- **Rifornimento**: DC
- **Scorta di sicurezza (SS) Store**: 6
- **Scorta di sicurezza (SS) DC**: 13
- **Livello di riordino (RL) DC**: 13
- **Livello massimo (ML) DC**: 658
- **Costo Totale**: 617.41
- **Numero di Iterazioni**: 100

Come previsto, l'aumento significativo del costo di giacenza presso lo store ha portato l'algoritmo a ridurre il livello di *safety stock* dello store a 6 unità. Tuttavia, la scorta di sicurezza non è stata abbassata a zero: infatti, riducendola a tale livello, lo store avrebbe probabilmente dovuto effettuare un ordine aggiuntivo verso il DC, comportando così un aumento dei costi di approvvigionamento.

Mantenere un livello minimo risulta necessario per bilanciare i costi di giacenza allo store con quelli associati agli ordini e al trasporto. Ridurre ulteriormente il livello di *safety stock* allo store, quindi, diminuirebbe il costo di giacenza, ma comporterebbe un incremento nei costi totali degli ordini e del trasporto necessari per garantire un adeguato approvvigionamento. Questi effetti risultano contrastanti: da un lato, una scorta di sicurezza troppo bassa aumenta i costi operativi; dall'altro, una scorta troppo elevata comporta un incremento del costo di giacenza.

Il livello di scorta determinato dall'algoritmo rappresenta quindi un punto di equilibrio ottimale tra questi costi contrapposti.

Il costo totale è di 617.41 euro, suddiviso in:

- Costo di giacenza allo store: 109.48
- Costo di giacenza al DC: 8.13
- Costo totale degli ordini: 255.0
- Costo di trasporto: 244.8
- Costo di stockout: 0.0
- Num. ordini effettuati dal DC: 0
- Num. ordini effettuati dallo store: 17

L'aumento del costo di giacenza presso lo store ha portato a una riduzione della scorta di sicurezza e a un incremento significativo del costo di giacenza, che in questo caso rappresenta una delle principali voci di spesa.

- **Test 3: Aumento dei costi di giacenza sia al DC che allo store**
  In questo scenario, il costo di giacenza presso lo store è stato aumentato di due ordini di grandezza, passando da 0,0015 euro/unità a 0,15 euro/unità, così come il costo di giacenza presso il DC, incrementato da 0,001 euro/unità a 0,1 euro/unità. Questo test è una combinazione dei test 1 e 2, quindi a partire dai risultati di questi due

test ci si aspetta che, visto l'alto costo di giacenza del DC, l'ottimizzatore preferisca l'opzione di rifornimento diretto; allo stesso tempo dato l'alto costo di giacenza allo store, ci si aspetta anche che la scorta di sicurezza allo store sia molto bassa, in modo tale da minimizzare la voce di costo corrispondente.
L'output finale del processo di ottimizzazione è il seguente:

- **Rifornimento**: Fornitore
- **Scorta di sicurezza (SS) Store**: 4
- **Scorta di sicurezza (SS) DC**: 0
- **Livello di riordino (RL) DC**: 0
- **Livello massimo (ML) DC**: 0
- **Costo Totale**: 721.83
- **Numero di Iterazioni**: 25

I risultati sono coerenti con quanto ci si aspetta, infatti l'algoritmo opta per il rifornimento diretto dal fornitore, con un livello di *safety stock* molto basso, pari a 4 unità.
Il costo totale è di 721.83 euro, suddiviso in:

- Costo di giacenza allo store: 121.83
- Costo di giacenza al DC: 0.0
- Costo totale degli ordini: 600.0
- Costo di trasporto: 0.0
- Costo di stockout: 0.0
- Num. ordini effettuati dal DC: 0
- Num. ordini effettuati dallo store: 15

Il costo totale, pari a 721.83 euro, riflette i costi di giacenza aumentati.

I test successivi sono effettuati con i valori di costi di giacenza reali, quindi quelli iniziali pari a 0.001 euro/unità per il DC e 0.0015 euro/unità per lo store.

- **Test 4: Eliminazione dei costi fissi di ordinazione da store a DC e da store a fornitore**
  In questo test, sono stati eliminati i costi fissi di ordinazione dallo store, azzerando sia il costo di ordinazione diretta dallo store al fornitore sia quello per le ordinazioni dallo store al DC. Il costo di ordinazione dal DC al fornitore, invece, rimane invariato e pari a 125 euro.

  In questo caso ci si aspetta che l'ottimizzatore scelga di rifornirsi direttamente da fornitore, infatti se si ordinasse tramite DC, si avrebbe una voce di spesa in più, data dai costi di trasporto, che invece nel caso di rifornimento diretto sono assenti, in quanto già inclusi nei costi di ordinazione, che però sono stati impostati a zero.

Inoltre, in questo scenario non è necessario applicare il compromesso tra costi operativi e costi di giacenza, come era stato fatto nel Test 2, poiché i costi di ordinazione e di trasporto sono nulli. Di conseguenza, la scorta di sicurezza dello store è stata impostata a zero.

L'output finale del processo di ottimizzazione è il seguente:

- **Rifornimento**: Fornitore
- **Scorta di sicurezza (SS) Store**: 0
- **Scorta di sicurezza (SS) DC**: 0
- **Livello di riordino (RL) DC**: 0
- **Livello massimo (ML) DC**: 0
- **Costo Totale**: 0.97
- **Numero di Iterazioni**: 21

Il costo totale è di 0.97 euro, suddiviso in:

- Costo di giacenza allo store: 0.97
- Costo di giacenza al DC: 0.0
- Costo totale degli ordini: 0.0
- Costo di trasporto: 0.0
- Costo di stockout: 0.0
- Num. ordini effettuati dal DC: 0
- Num. ordini effettuati dallo store: 17

In questo scenario, l'eliminazione dei costi di ordinazione ha determinato una drastica riduzione del costo totale, portando il livello di *safety stock* presso lo store a zero. Questo risultato si verifica poiché la percentuale di *stockout* è pari a zero, rendendo superfluo mantenere una scorta di sicurezza nello store.

Il costo di giacenza presso lo store, tuttavia, non è completamente nullo: esso ammonta a 0,97 euro, valore che rappresenta il costo medio di giacenza calcolato sull'intero periodo considerato.

- **Test 5: Azzeramento del costo fisso di ordinazione da store a fornitore**

  In questo test il costo fisso di ordinazione dallo store verso il fornitore è stato azzerato, mentre il costo fisso di ordinazione dallo store al DC è stato mantenuto al valore standard di 15 euro.
  In tale configurazione, ci si aspetta risultati molto simili a quelli del Test 4.
  È ragionevole prevedere che l'ottimizzatore opti per il rifornimento diretto dal fornitore, poiché l'azzeramento del costo fisso di ordinazione dallo store al fornitore, combinato con l'assenza di costi di trasporto, comporta che l'unica voce di costo rilevante sia quella relativa allo giacenza presso lo store. Al contrario, rifornendosi dal DC, si incorrerebbe sia in costi di trasporto che in costi di ordinazione.

  L'output finale del processo di ottimizzazione è il seguente:

  - **Rifornimento**: Fornitore
  - **Scorta di sicurezza (SS) Store**: 0
  - **Scorta di sicurezza (SS) DC**: 0
  - **Livello di riordino (RL) DC**: 0
  - **Livello massimo (ML) DC**: 0
  - **Costo Totale**: 0.97
  - **Numero di Iterazioni**: 21

  Il costo totale è di 0.97 euro, suddiviso in:

  - Costo di giacenza allo store: 0.97
  - Costo di giacenza al DC: 0.0
  - Costo totale degli ordini: 0.0
  - Costo di trasporto: 0.0
  - Costo di stockout: 0.0
  - Num. ordini effettuati dal DC: 0
  - Num. ordini effettuati dallo store: 17

  Come previsto infatti, l'algoritmo ha scelto il rifornimento diretto dal fornitore, con una scorta di sicurezza pari a 0 unità, poichè anche in questo caso, come nel caso del Test 4, la percentuale di stockout è zero, la scorta di sicurezza allo store risulta superflua ed è quindi impostata a zero.
  In questo test, così come nel caso del Test 4, vengono effettuati due ordini in più rispetto ai casi di rifornimento diretto con costi di ordinazione (Test 1 e Test 3). Ciò avviene poiché, in presenza di un costo di ordinazione nullo e di un costo di giacenza positivo, risulta vantaggioso aumentare il numero di ordini, mantenendo quantità ridotte di merce nello store e riducendo di conseguenza i costi di giacenza.

- **Test 6: Aumento del costo fisso di ordinazione dallo store al DC**
  In questo test, il costo fisso di ordinazione dallo store al DC è stato incrementato di un ordine di grandezza, raggiungendo il valore di 150 euro, mentre il costo fisso di ordinazione dallo store al fornitore è stato mantenuto al valore standard di 40 euro. È ragionevole supporre che, in questo scenario, l'ottimizzatore preferisca il rifornimento diretto dal fornitore. Infatti, ordinando tramite il DC, il costo totale degli ordini risulterebbe significativamente più elevato rispetto al caso base del Test 0, dove il costo fisso di ordinazione era pari a 15 euro.
  Inoltre, il rifornimento diretto dal fornitore consente di evitare i costi aggiuntivi di trasporto che sarebbero invece sostenuti nel caso di approvvigionamento tramite il DC.

  L'output finale del processo di ottimizzazione è il seguente:

  - **Rifornimento**: Fornitore

  - **Scorta di sicurezza (SS) Store**: 4
  - **Scorta di sicurezza (SS) DC**: 0
  - **Livello di riordino (RL) DC**: 0
  - **Livello massimo (ML) DC**: 0
  - **Costo Totale**: 601.22
  - **Numero di Iterazioni**: 61

  La configurazione ottimale individuata prevede un rifornimento diretto dal fornitore, come era prevedibile, con una scorta di sicurezza presso lo store pari a 4 unità.
  Il costo totale è di 601.22 euro, suddiviso in:

  - Costo di giacenza allo store: 1.22
  - Costo di giacenza al DC: 0.0
  - Costo totale degli ordini: 600.0
  - Costo di trasporto: 0.0
  - Costo di stockout: 0.0
  - Num. ordini effettuati dal DC: 0
  - Num. ordini effettuati dallo store: 15

- **Test 7: Eliminazione del costo fisso di ordinazione dal DC al fornitore**
  In questo test, il costo fisso di ordinazione dal DC al fornitore è stato azzerato, mentre i costi fissi di ordinazione dallo store al fornitore e dallo store al DC sono stati impostati ai valori standard, pari rispettivamente a 40 euro e 15 euro. In questa configurazione, ci si aspetta che, in assenza di costi di ordinazione dal DC al fornitore, l'ottimizzatore tenda a mantenere livelli di scorta molto bassi presso il DC per ridurre al minimo il costo di giacenza. Questo permette, infatti, di aumentare la frequenza degli ordini dal DC al fornitore rispetto a un contesto in cui il costo fisso di ordinazione tra DC e fornitore è pari a 125 euro, poiché, in questo caso, ogni ordine ha costo nullo.

  La scelta dell'ottimizzatore di preferire il rifornimento tramite DC anziché direttamente dal fornitore è supportata dal fatto che, nel Test 0, una configurazione identica aveva già indicato il DC come opzione più conveniente, nonostante un costo fisso di ordinazione pari a 125 euro. In questo scenario, con tale costo azzerato, la convenienza del rifornimento tramite DC risulta ulteriormente rafforzata.

  L'output finale del processo di ottimizzazione è il seguente:

  - **Rifornimento**: DC
  - **Scorta di sicurezza (SS) Store**: 35
  - **Scorta di sicurezza (SS) DC**: 89
  - **Livello di riordino (RL) DC**: 94
  - **Livello massimo (ML) DC**: 94

- **Costo Totale**: 385.19
- **Numero di Iterazioni**: 53

Come previsto, l'algoritmo ha scelto di rifornirsi tramite il DC, con una scorta di sicurezza allo store pari a 35 unità. I parametri di gestione delle scorte al DC sono: *safety stock* pari a 89 unità, *reorder level* pari a 94 unità e *max level* pari a 94 unità. Il costo totale è di 385.19 euro, suddiviso in:

- Costo di giacenza allo store: 2.02
- Costo di giacenza al DC: 0.97
- Costo totale degli ordini: 195.0
- Costo di trasporto: 187.2
- Costo di stockout: 0.0
- Num. ordini effettuati dal DC: 8
- Num. ordini effettuati dallo store: 13

Rispetto al caso del Test 0, il *max level* risulta significativamente ridotto, passando da 647 unità nel Test 0 a 94 unità in questo scenario. Di conseguenza, il numero di ordini effettuati dal DC è aumentato notevolmente: in questo test sono stati effettuati 8 ordini, mentre nel Test 0 non si è registrato alcun ordine (escludendo gli ordini effettuati durante il periodo di warm-up).

Conclusioni sull'analisi dei risultati sperimentali

L'analisi condotta evidenzia come l'algoritmo di ottimizzazione sia in grado di rispondere in modo coerente e razionale ai cambiamenti nei parametri di costo. In particolare, l'algoritmo adatta le politiche di approvvigionamento al fine di minimizzare i costi totali, bilanciando in maniera efficace le diverse componenti di costo, quali i costi di ordinazione, di trasporto e di giacenza. I risultati ottenuti dimostrano, quindi, che l'approccio adottato è robusto e in grado di individuare configurazioni ottimali anche in presenza di significative variazioni dei parametri, confermando la validità della strategia di ottimizzazione implementata.

Capitolo 4

Decomposizione di Problemi Black-Box: Sfide e Approccio Risolutivo

4.1 Necessità e importanza di adottare tecniche di decomposizione

In contesti reali, ci si trova spesso ad affrontare problemi di ottimizzazione su larga scala, dove una delle principali sfide è legata alla crescita esponenziale della dimensione dello spazio di ricerca in relazione al numero di variabili decisionali. Il numero di variabili decisionali di un problema svolge un ruolo fondamentale nel determinare la sua complessità. Come evidenziato in L. Grippo [2011], infatti, se il numero di variabili decisionali n è molto elevato, risolvere il problema può diventare estremamente difficile, se non addirittura impossibile, a causa delle limitazioni delle risorse computazionali, come ad esempio la memoria di lavoro del calcolatore. Di conseguenza, è necessario decomporre il problema originale, su larga scala, in sottoproblemi di dimensioni più contenute, più semplici da risolvere. Successivamente, si procederà all'ottimizzazione dei sottoproblemi in modo separato.

4.1.1 Difficoltà legate alla decomposizione di problemi black-box

Nel caso di problemi analitici, dove sono note sia la struttura del problema che la funzione obiettivo, esistono in letteratura numerose tecniche di decomposizione, le quali possono essere adottate in base alla natura del problema. Tuttavia, per problemi di tipo black-box, dove queste informazioni non sono disponibili, è necessario ricorrere a metodi di decomposizione alternativi rispetto a quelli tradizionalmente utilizzati.

In un problema di ottimizzazione black-box, infatti, le informazioni sulle interazioni tra le variabili decisionali non sono accessibili, e pertanto è necessario considerare i seguenti aspetti principali:

- Poiché la struttura del problema non è conosciuta, non è possibile decomporre manualmente il problema stesso. È quindi indispensabile sviluppare un metodo che sia in grado di sfruttare la struttura nascosta del problema per determinare automaticamente una decomposizione efficace.
- A causa della mancanza di informazioni sulla struttura, un dato problema potrebbe essere decomposto in modi differenti, senza che sia possibile stabilire a priori quale decomposizione sia più vantaggiosa rispetto ad altre.

Pertanto, risulta essenziale sviluppare una tecnica in grado di individuare e sfruttare la struttura intrinseca del problema, al fine di ottenere una decomposizione efficace e ottimale. A tal proposito, è necessario definire cosa si intenda per una buona decomposizione di un problema. Una decomposizione può essere considerata di qualità elevata quando è caratterizzata da una minima dipendenza tra le sue componenti. Per comprendere meglio questo concetto, è necessario introdurre le seguenti definizioni di separabilità della funzione obiettivo.

Definizione 4.1.1 (Separabilità Parziale, M.N. Omidvar [2017])**.** *Una funzione $f(\mathbf{x})$ è parzialmente separabile con m componenti indipendenti se e solo se:*

$$\arg\min_{\mathbf{x}} f(\mathbf{x}) = \left(\arg\min_{\mathbf{x}_1} f(\mathbf{x}_1, \dots), \dots, \arg\min_{\mathbf{x}_m} f(\dots, \mathbf{x}_m)\right)$$

dove $\mathbf{x} = (\mathbf{x}_1, \dots, \mathbf{x}_n)^\top$ è il vettore delle variabili decisionali di dimensione n, $\mathbf{x}_1, \dots, \mathbf{x}_m$ sono sotto-vettori disgiunti di $\mathbf{x}$, e $2 \leq m \leq n$.

Se tutte le sottofunzioni componenti sono 1-dimensionali, la funzione è detta completamente separabile. Un caso particolare di separabilità parziale è la separabilità additiva, definita come segue.

Definizione 4.1.2 (Separabilità Additiva Parziale, M.N. Omidvar [2017])**.** *Una funzione $f(\mathbf{x})$ è parzialmente separabile in modo additivo se può essere scritta nella forma generale:*

$$f(\mathbf{x}) = \sum_{i=1}^{m} f_i(\mathbf{x}_i), \quad m > 1$$

dove $f_i(\cdot)$ rappresenta una sottofunzione non separabile e m è il numero di componenti non separabili di f.

Nella sezione successiva viene descritto un metodo di decomposizione che soddisfa i requisiti necessari per garantire una decomposizione ottimale di un problema di tipo black-box, facendo uso delle definizioni e dei concetti introdotti in precedenza.

4.2 Differential Grouping (DG)

In M.N. Omidvar [2017] viene presentata una versione più efficiente ed efficace del *Differential Grouping*, denominata *Differential Grouping 2*. Questo metodo di decomposizione

automatica sfrutta esclusivamente la valutazione della funzione obiettivo in punti dati per esaminare tutte le possibili coppie di variabili e determinare se vi sia o meno interazione tra di esse. Tale analisi risulta fondamentale per identificare le componenti sovrapposte della funzione obiettivo, permettendo una decomposizione accurata ed efficiente.

Il seguente teorema costituisce il fondamento per la rilevazione delle interazioni nel metodo di *Differential Grouping* (DG).

Teorema 1 (Rilevazione delle Interazioni, M.N. Omidvar [2014])**.** *Sia $f(\mathbf{x})$ una funzione additivamente separabile. Per ogni a, $b_1 \neq b_2$, $\delta \in \mathbb{R}$, con $\delta \neq 0$, le variabili x_p e x_q interagiscono se e solo se la seguente condizione è soddisfatta:*

$$\Delta_{\delta,x_p}[f](\mathbf{x})\big|_{x_p=a,x_q=b_1} \neq \Delta_{\delta,x_p}[f](\mathbf{x})\big|_{x_p=a,x_q=b_2}, \tag{4.1}$$

dove

$$\Delta_{\delta,x_p}[f](\mathbf{x}) = f(...,x_p+\delta,...) - f(...,x_p,...) \tag{4.2}$$

è la differenza forward della funzione f rispetto alla variabile x_p, con passo δ.

Il Teorema 4.1 afferma che due variabili x_p e x_q interagiscono se l'equazione (4.2), calcolata per due diversi valori di x_q, produce risultati differenti. Una dimostrazione formale del teorema è disponibile in M.N. Omidvar [2014].

Per semplicità, si denota il lato sinistro della (4.1) con $\Delta^{(1)}$ e il lato destro con $\Delta^{(2)}$. Pertanto, la condizione di interazione può essere espressa come:

$$\Delta^{(1)} \neq \Delta^{(2)} \iff |\Delta^{(1)} - \Delta^{(2)}| \neq 0. \tag{4.3}$$

Tuttavia, questa verifica di uguaglianza non è pratica nei dispositivi di calcolo a causa della precisione limitata dei numeri in virgola mobile. Per questo motivo, la condizione di uguaglianza può essere convertita in una verifica di disuguaglianza nella forma:

$$\lambda = |\Delta^{(1)} - \Delta^{(2)}| > \epsilon, \tag{4.4}$$

introducendo il parametro di controllo ϵ, che determina la sensibilità del metodo DG2 nella rilevazione delle interazioni.

4.2.1 Struttura Dettagliata del Metodo DG2

Si procede ora a fornire una descrizione dettagliata del funzionamento del metodo *DG2*, presentato in M.N. Omidvar [2017], illustrando i principali passaggi e lo pseudocodice degli algoritmi necessari per la sua implementazione.

Calcolo della Matrice delle Interazioni Λ (Funzione ISM)

La funzione `ISM` ha il compito di costruire la matrice delle interazioni Λ, che rappresenta la forza di interazione tra le variabili decisionali di una funzione obiettivo f. L'algoritmo si basa sull'analisi della risposta di f a variazioni singole e congiunte delle variabili.

Per ogni coppia di variabili (i,j), la funzione f viene valutata in tre configurazioni distinte:

- **Variazione singola di** i: La variabile i viene modificata, mentre tutte le altre rimangono fissate al loro valore di riferimento definito in $\mathbf{x}_{\text{lbar}}$. La valutazione corrispondente della funzione obiettivo è indicata con $\boldsymbol{f_{hat}}[i]$.
- **Variazione singola di** j: La variabile j viene modificata, mantenendo invariati gli altri parametri rispetto a $\mathbf{x}_{\text{lbar}}$. La valutazione corrispondente della funzione obiettivo è indicata con $\boldsymbol{f_{hat}}[j]$.
- **Variazione congiunta di** i **e** j: Entrambe le variabili i e j vengono modificate simultaneamente. La valutazione della funzione obiettivo in questa configurazione è indicata con $F[i, j]$.

Le differenze nelle valutazioni della funzione f tra le varie configurazioni sono calcolate come segue:

$$\Delta^{(1)} = \boldsymbol{f_{hat}}[i] - f_{base},$$

dove f_{base} è il valore della funzione obiettivo calcolato nella configurazione di riferimento $\mathbf{x}_{\text{lbar}}$. Analogamente, si definisce:

$$\Delta^{(2)} = F[i, j] - \boldsymbol{f_{hat}}[j].$$

La forza di interazione tra le variabili i e j è quantificata tramite il valore assoluto della differenza tra $\Delta^{(1)}$ e $\Delta^{(2)}$:

$$\Lambda[i, j] = |\Delta^{(1)} - \Delta^{(2)}|.$$

Il valore $\Lambda[i, j]$ è registrato nella matrice Λ, fornendo una misura diretta della relazione tra le due variabili. L'intero processo consente di costruire una rappresentazione delle interazioni, utile per analisi di dipendenze o strutture decisionali.

Gestione delle variabili binarie Un aspetto distintivo dell'implementazione della funzione `ISM` descritta in questa tesi è l'introduzione di una gestione esplicita delle variabili binarie. Questa modifica è stata progettata per garantire una costruzione corretta della matrice delle interazioni (Λ), anche in presenza di variabili decisionali discrete che assumono valori binari (0 o 1). Tale adattamento costituisce un'estensione dell'implementazione originale descritta in M.N. Omidvar [2017].

Le variabili binarie sono trattate durante il calcolo del vettore $\mathbf{m}$, che rappresenta un punto medio tra due configurazioni, $\mathbf{x}_{\text{lbar}}$ e $\mathbf{x}_{\text{ubar}}$, selezionate nel dominio della funzione obiettivo. È importante sottolineare che $\mathbf{x}_{\text{lbar}}$ e $\mathbf{x}_{\text{ubar}}$ non corrispondono ai limiti del dominio della funzione f, ma sono piuttosto due punti distinti scelti appositamente per il calcolo. In particolare:

- $\mathbf{x}_{\text{lbar}}$ è la configurazione di riferimento o base, utilizzata per generare nuove configurazioni attraverso la variazione di una o due variabili alla volta.
- $\mathbf{x}_{\text{ubar}}$ è un secondo punto preso nel dominio della funzione obiettivo f, utilizzato esclusivamente per il calcolo del vettore $\mathbf{m}$.

Il vettore $\mathbf{m}$ è definito come il punto medio tra le due configurazioni, calcolato componente per componente. Le componenti di $\mathbf{m}$ vengono quindi impiegate per modificare $\mathbf{x}_{\text{lbar}}$ e ottenere nuove configurazioni da utilizzare nel calcolo delle interazioni tra le variabili. Affinché $\mathbf{m}$ sia coerente con la natura binaria delle variabili, i valori binari di $\mathbf{m}$ vengono esplicitamente aggiornati, invertendo i valori rispetto a quelli presenti in $\mathbf{x}_{\text{lbar}}$. Questo processo viene implementato come segue:

- Se una variabile binaria i assume il valore 0 in $\mathbf{x}_{\text{lbar}}$, il corrispondente valore in $\mathbf{m}$ è impostato a 1.
- Analogamente, se una variabile binaria i assume il valore 1 in $\mathbf{x}_{\text{lbar}}$, il corrispondente valore in $\mathbf{m}$ è impostato a 0.

Lo pseudocodice relativo alla funzione `ISM` è riportato in 5.

Determinazione della Struttura di Interazione Θ (Funzione DSM)

La funzione `DSM` ha lo scopo di costruire una matrice binaria Θ che rappresenta la struttura delle interazioni tra le variabili decisionali, partendo dalla matrice delle interazioni Λ. Ogni elemento $\Theta[i, j]$ indica se esiste o meno una relazione significativa tra le variabili i e j.
Il processo si articola in due cicli principali:

Primo ciclo: Calcolo delle soglie e classificazione preliminare. Durante il primo ciclo, vengono calcolate due soglie per ciascuna coppia di variabili (i, j):

- La soglia inferiore $E_{\text{inf}}[i, j]$, calcolata come funzione di gamma_func(2) e del massimo tra le somme dei valori della funzione obiettivo calcolati nelle configurazioni di riferimento: $f_{\text{base}} + F[i, j]$ e $\boldsymbol{f_{hat}}[i] + \boldsymbol{f_{hat}}[j]$. Questa soglia rappresenta un limite sotto il quale le variabili i e j sono considerate *non interattive*.
- La soglia superiore $E_{\text{sup}}[i, j]$, calcolata come funzione di gamma_func($\sqrt{n}$) e del massimo valore tra $f_{\text{base}}, F[i, j]$, e le valutazioni per singole variabili $\boldsymbol{f_{hat}}[i]$ e $\boldsymbol{f_{hat}}[j]$. Questa soglia rappresenta il limite oltre il quale le variabili i e j sono considerate *interattive*.

La classificazione preliminare delle variabili è effettuata confrontando $\Lambda[i, j]$ con le soglie calcolate:

- Se $\Lambda[i, j] < E_{\text{inf}}[i, j]$, le variabili i e j sono classificate come *non interattive* ($\Theta[i, j] = 0$).
- Se $\Lambda[i, j] > E_{\text{sup}}[i, j]$, le variabili i e j sono classificate come *interattive* ($\Theta[i, j] = 1$).

Inoltre, vengono aggiornati i contatori η_0 e η_1, che registrano rispettivamente il numero di coppie di variabili classificate come non interattive o interattive.

Secondo ciclo: Calcolo delle soglie adattive. Per le coppie di variabili che appartengono al cosiddetto *range grigio*, definito dall'intervallo:

$$E_{\text{inf}}[i,j] \leq \Lambda[i,j] \leq E_{\text{sup}}[i,j],$$

e che quindi non possono essere classificate durante il primo ciclo (cioè, per cui $\Theta[i,j] = \text{NaN}$), viene calcolata una soglia adattiva $E[i,j]$. Tale soglia è determinata come una combinazione pesata delle soglie $E_{\text{inf}}[i,j]$ ed $E_{\text{sup}}[i,j]$, in funzione dei contatori η_0 e η_1:

$$E[i,j] = \frac{\eta_0}{\eta_0 + \eta_1} \cdot E_{\text{inf}}[i,j] + \frac{\eta_1}{\eta_0 + \eta_1} \cdot E_{\text{sup}}[i,j].$$

La soglia adattiva tiene conto del bilancio tra le interazioni rilevate e consente una classificazione più precisa delle variabili:

- Se $\Lambda[i,j] > E[i,j]$, le variabili i e j sono classificate come *interattive* ($\Theta[i,j] = 1$).
- Altrimenti, le variabili i e j sono classificate come *non interattive* ($\Theta[i,j] = 0$).

Al termine dell'algoritmo, la funzione `DSM` restituisce la matrice binaria Θ, che indica la presenza ($\Theta[i,j] = 1$) o l'assenza ($\Theta[i,j] = 0$) di interazioni significative tra le variabili. Questa metodologia consente di rappresentare la struttura delle interazioni in modo rigoroso, adattando la classificazione alla natura delle interazioni rilevate nel sistema.
Lo pseudocodice relativo alla funzione `DSM` è presentato in 6.

Definizione della funzione `gamma_func(n)` utilizzata nel DSM per calcolare i valori delle soglie

Innanzitutto è necessario definire il valore `muM`, che è calcolato come segue:

$$\texttt{muM} = \frac{\texttt{np.finfo(float).eps}}{2}$$

dove `muM` rappresenta la metà della precisione numerica della macchina. Questo valore è utilizzato per esprimere la sensibilità agli errori computazionali nelle operazioni in virgola mobile, e gioca un ruolo fondamentale nell'analisi delle interazioni tra le variabili, stabilendo i limiti di tolleranza per determinare la presenza o l'assenza di interazioni.

La funzione `gamma_func`(n) utilizza il valore di `muM` per calcolare un margine di errore accettabile che dipende dal parametro di input n. La funzione è definita come segue:

$$\texttt{gamma\_func}(n) = \frac{n \cdot \texttt{muM}}{1 - n \cdot \texttt{muM}}$$

La funzione `gamma_func`(n) viene utilizzata per calcolare due soglie principali:

- `gamma_func`(2), che corrisponde al fattore moltiplicativo usato in `DSM` per calcolare la soglia inferiore E_{inf}, sotto la quale le variabili sono considerate *non interattive*.
- `gamma_func`($\sqrt{n}$), che corrisponde al fattore moltiplicativo usato in `DSM` per calcolare la soglia superiore E_{sup}, al di sopra della quale le variabili sono considerate *interattive*.

Questo approccio consente di definire un intervallo, noto come il *range grigio*, in cui l'interazione tra le variabili è incerta, e pertanto vengono applicate soglie adattive per una classificazione più precisa.
Lo pseudocodice relativo alla funzione `gamma_func(n)` è presentato in 8.

Individuazione dei Gruppi di Variabili Connesse (Funzione `ConnComp`)

La funzione `ConnComp` ha lo scopo di analizzare la matrice di struttura binaria Θ per identificare i gruppi di variabili connesse. Ogni gruppo rappresenta un insieme di variabili che sono direttamente o indirettamente collegate tra loro, mentre le variabili non collegate sono considerate indipendenti.

L'algoritmo utilizza la tecnica di ricerca in profondità (*Depth-First Search*, DFS) per esplorare la matrice Θ. Partendo da una variabile non ancora visitata, l'algoritmo esplora ricorsivamente tutte le variabili collegate a essa, formando un gruppo di variabili connesse. Una volta completata l'esplorazione di un gruppo, l'algoritmo passa a una nuova variabile non ancora visitata e ripete il processo, fino a esplorare tutte le variabili.

Il risultato dell'algoritmo è una lista di gruppi di variabili connesse, dove le variabili che presentano interazioni significative tra di loro formano dei gruppi di variabili *non separabili*, mentre le variabili isolate o che non hanno interazioni vengono considerate come variabili *separabili*.
Lo pseudocodice relativo alla funzione `ConnComp` è presentato in 7.

Definizione dei Gruppi di Variabili (Funzione DG2)

La funzione `DG2` si occupa di coordinare l'intero processo:

1. Costruisce Λ tramite `ISM`.
2. Costruisce Θ tramite `DSM`.
3. Usa `ConnComp` per identificare gruppi connessi di variabili.

Alla fine, `DG2` classifica:

- `nonsep_groups`: gruppi non separabili di variabili interattive.
- `xsep`: gruppi di variabili separabili.

Lo pseudocodice relativo alla funzione `DG2` è presentato in 9.

4.2.2 Analisi del Metodo *DG2*: Numero di Valutazioni della Funzione e Soglia Automatica ϵ

Il metodo *DG2*, sviluppato come estensione del metodo *DG*, affronta due problematiche fondamentali che limitano l'efficienza e l'accuratezza della decomposizione delle funzioni complesse in componenti indipendenti. Questi problemi, identificati nella formulazione originale, sono i seguenti:

1. Trovare un'implementazione efficiente per la funzione `ISM`, al fine di costruire la matrice delle interazioni (*interaction structure matrix*) utilizzando il minor numero possibile di valutazioni della funzione obiettivo.
2. Sviluppare una tecnica efficace di *thresholding* che permetta una decomposizione accurata della funzione in componenti, assicurando una buona generalizzazione su una vasta gamma di funzioni.

Nel seguito, vengono descritte le soluzioni introdotte in *DG2* (M.N. Omidvar [2017]) per affrontare queste sfide. In particolare, viene illustrato come il metodo riesca ad ottimizzare il numero di valutazioni della funzione obiettivo necessarie per costruire la matrice delle interazioni e come l'introduzione di una soglia automatica ϵ consenta di migliorare la precisione della decomposizione, rendendo il metodo applicabile a un ampio spettro di problemi.

Numero di Valutazioni di Funzione per la Decomposizione Come spiegato in M.N. Omidvar [2017], per identificare le interazioni tra le variabili decisionali, è necessario esaminare tutte le coppie di variabili per individuare eventuali sovrapposizioni nelle funzioni. Considerando una funzione con n variabili decisionali, il numero totale di interazioni da esaminare è dato da $\binom{n}{2} = \frac{n(n-1)}{2}$.

In particolare, per determinare l'interazione tra la i-esima e la j-esima variabile, sono necessarie quattro valutazioni di funzione:

$$\Delta^{(1)} = f(\dots, x_i, \dots) - f(x_1, \dots, x_n),$$
$$\Delta^{(2)} = f(\dots, x_i, \dots, x_j, \dots) - f(\dots, x_j, \dots).$$

Seguendo questo schema, il numero totale di valutazioni richieste risulta essere $4 \cdot \binom{n}{2}$, ovvero pari a $2n(n-1)$.
Tuttavia, il numero effettivo di valutazioni uniche della funzione obiettivo richieste è significativamente inferiore rispetto al numero totale di valutazioni calcolato inizialmente. Questo perché alcune valutazioni risultano ridondanti a causa della natura stessa del metodo e degli assunti fatti durante la costruzione della matrice delle interazioni. In particolare, molte configurazioni della funzione vengono valutate più volte nel corso del processo. Ad esempio:

- La configurazione di riferimento $f(x_1, \dots, x_n)$, che rappresenta la valutazione della funzione obiettivo nella configurazione iniziale, viene riutilizzata più volte durante i calcoli relativi alle interazioni tra variabili.
- Le configurazioni in cui si varia una singola variabile $f(\dots, x_i, \dots)$ vengono calcolate ripetutamente per diverse coppie di variabili durante l'analisi delle interazioni.

Di conseguenza, il numero totale di valutazioni inizialmente stimato ($2n(n-1)$) include queste valutazioni duplicate. In particolare:

- Le valutazioni ridondanti della configurazione di riferimento $f(x_1, \dots, x_n)$ ammontano a $\binom{n}{2} - 1 = \frac{n(n-1)}{2} - 1$.
- Le valutazioni ridondanti delle configurazioni con una singola variabile modificata, $f(\dots, x_i, \dots)$, ammontano a $n(n-2)$.

Pertanto, sottraendo il numero di valutazioni ridondanti dal totale, il numero minimo di valutazioni uniche della funzione obiettivo richieste per costruire la matrice delle interazioni risulta essere:

$$\frac{n(n+1)}{2} + 1.$$

riducendo così in modo significativo il costo computazionale del metodo. Questo valore rappresenta il numero minimo di valutazioni necessarie per costruire la matrice della struttura delle interazioni.

Utilizzo di una Soglia automatica ϵ Come illustrato in M.N. Omidvar [2017], un elemento distintivo del metodo *DG2*, rispetto al suo predecessore *DG*, è la capacità di migliorare significativamente l'accuratezza nella suddivisione delle variabili in gruppi (*grouping accuracy*), in particolare quando applicato a problemi ad ampia scala. Un ulteriore vantaggio chiave di *DG2* è la sua natura completamente automatica, grazie all'eliminazione di qualsiasi parametro esterno (*parameter-free*). Questo risultato è ottenuto incorporando gli errori di arrotondamento computazionale nel processo di calcolo di una soglia adattiva (ϵ), utilizzata per rilevare le interazioni più deboli. Rispetto a *DG*, *DG2* offre due miglioramenti principali:

- Invece di applicare un unico valore globale di soglia (ϵ) a tutte le interazioni, *DG2* determina un valore di soglia specifico per ogni coppia di variabili. Questo viene fatto stimando gli errori di arrotondamento associati a ciascuna interazione e adattando dinamicamente la soglia di conseguenza. Tale flessibilità risulta particolarmente vantaggiosa nel caso di funzioni non bilanciate, in cui gli errori di arrotondamento possono variare sensibilmente tra le diverse componenti.
- Il metodo non richiede alcun intervento manuale da parte dell'utente per specificare parametri esterni. Ciò rende *DG2* un approccio completamente autonomo, semplificando l'applicazione pratica e aumentando la sua capacità di adattarsi a una vasta gamma di funzioni.

Limitazioni della Soglia Automatica nell'Ottimizzazione Black-Box L'utilizzo di una soglia automatica (ϵ), però, potrebbe non rappresentare la scelta ottimale nel contesto di problemi di ottimizzazione black-box. In tali scenari, la funzione obiettivo potrebbe essere affetta da rumore, rendendo meno efficace l'approccio basato su una soglia calcolata automaticamente. In alternativa, si potrebbe optare per l'adozione di una soglia manuale, scelta in base alle caratteristiche specifiche del problema. Questa soluzione consentirebbe di rilassare il processo di decomposizione, superando le limitazioni imposte dalla soglia automatica, che in alcuni casi potrebbe risultare troppo stringente e condurre a una decomposizione non ottimale. La possibilità di impostare una soglia manuale garantisce una maggiore flessibilità, adattandosi meglio alle esigenze di contesti complessi come l'ottimizzazione black-box.

Algorithm 5: Interaction Structure Matrix (ISM) M.N. Omidvar [2017]

Input: f: funzione obiettivo,
n: numero di variabili decisionali,
$\mathbf{x_{ubar}}$, $\mathbf{x_{lbar}}$: limiti superiore e inferiore presi all'interno del dominio di f,
num_var_bin: numero variabili binarie
Output: Λ: matrice delle interazioni,
F: matrice delle valutazioni della funzione obiettivo,
$\boldsymbol{f_{hat}}$: vettore delle valutazioni variando le singole variabili,
f_{base}: valutazione della configurazione di base,
Γ: numero totale di valutazioni.

Inizializza: $\Lambda \leftarrow 0^{n \times n}$
$F \leftarrow NaN^{n \times n}$
$\mathbf{f_{hat}} \leftarrow NaN^{n}$
$\Delta^{(1)}, \Delta^{(2)} \leftarrow NaN^{n \times n}$
Imponi $\mathbf{x}^{(1)} \leftarrow \mathbf{x_{lbar}}$ (copia di $\mathbf{x_{lbar}}$)
Valuta: f($\mathbf{x}^{(1)}$) e assegna il valore a f_{base} e inizializza $\Gamma \leftarrow 1$
Calcola il vettore $\mathbf{m}$ come punto medio di $\mathbf{x_{lbar}}$ e $\mathbf{x_{ubar}}$
if $num\_var\_bin \neq 0$ **then**
    Aggiorna le variabili binarie in $\mathbf{m}$, in modo tale che assumano il valore opposto rispetto ai valori che assumono in $\mathbf{x_{lbar}}$:
    for $i \leftarrow 1$ ***to*** num_var_bin **do**
        if $\boldsymbol{x_{lbar}}[i] == 0$ **then**
            $\mathbf{m}[i] \leftarrow 1$
        else if $\boldsymbol{x_{lbar}}[i] == 1$ **then**
            $\mathbf{m}[i] \leftarrow 0$
for $i \leftarrow 1$ **to** $n - 1$ **do**
    if $\boldsymbol{f_{hat}}[i]$ *è* NaN **then**
        $\mathbf{x}^{(2)} \leftarrow \mathbf{x}^{(1)}$ (copia di $\mathbf{x}^{(1)}$)
        Modifica $\mathbf{x}^{(2)}[i] \leftarrow \mathbf{m}[i]$
        Valuta f($\mathbf{x}^{(2)}$) e aggiorna $\boldsymbol{f_{hat}}[i]$
        $\Gamma \leftarrow \Gamma + 1$
    for $j \leftarrow i + 1$ **to** n **do**
        if $\boldsymbol{f_{hat}}[j]$ *è* NaN **then**
            $\mathbf{x}^{(3)} \leftarrow \mathbf{x}^{(1)}$
            Modifica $\mathbf{x}^{(3)}[j] \leftarrow \mathbf{m}[j]$
            Valuta f($\mathbf{x}^{(3)}$) e aggiorna $\boldsymbol{f_{hat}}[j]$
            $\Gamma \leftarrow \Gamma + 1$
        Imponi $\mathbf{x}^{(4)} \leftarrow \mathbf{x}^{(1)}$
        Modifica $\mathbf{x}^{(4)}[i] \leftarrow \mathbf{m}[i]$ e $\mathbf{x}^{(4)}[j] \leftarrow \mathbf{m}[j]$
        Valuta f($\mathbf{x}^{(4)}$) e assegna il valore a $F[i, j]$
        $F[j, i] \leftarrow F[i, j]$
        $\Gamma \leftarrow \Gamma + 1$
        $\Delta^{(1)} \leftarrow \boldsymbol{f_{hat}}[i] - f_{base}$
        $\Delta^{(2)} \leftarrow F[i, j] - \boldsymbol{f_{hat}}[j]$
        $\Lambda[i, j] \leftarrow |\Delta^{(1)} - \Delta^{(2)}|$
        $\Lambda[j, i] \leftarrow \Lambda[i, j]$
return $\Lambda, F, \boldsymbol{f_{hat}}, f_{base}, \Gamma$

Algorithm 6: Design Structure Matrix (DSM) M.N. Omidvar [2017]

Input: Λ: matrice delle interazioni,
F: matrice delle valutazioni della funzione obiettivo,
$\boldsymbol{f_{hat}}$: vettore delle valutazioni per singole variabili,
f_{base}: valore della funzione obiettivo base,
n: numero di variabili decisionali.
Output: Θ: matrice di struttura binaria,
E_{inf}: matrice delle soglie inferiori,
E_{sup}: matrice delle soglie superiori,
E: matrice delle soglie adattive.

Inizializza $\Theta \leftarrow NaN^{n\times n}$, $E_{\text{inf}}, E_{\text{sup}}, E \leftarrow NaN^{n\times n}$,
$\eta_0 \leftarrow 0$ (contatore per le non interazioni), $\eta_1 \leftarrow 0$ (contatore per le interazioni).
Primo ciclo: Valutazione delle soglie e assegnazione iniziale di Θ
for $i \leftarrow 1$ **to** $n-1$ **do**
  for $j \leftarrow i+1$ **to** n **do**
    Calcola $f_{\max} \leftarrow \max(f_{\text{base}}, F[i,j], \boldsymbol{f_{hat}}[i], \boldsymbol{f_{hat}}[j])$;
    Calcola $E_{\text{inf}}[i,j] \leftarrow \text{gamma\_func}(2) \cdot \max(f_{\text{base}} + F[i,j], \boldsymbol{f_{hat}}[i] + \boldsymbol{f_{hat}}[j])$;
    Calcola $E_{\text{sup}}[i,j] \leftarrow \text{gamma\_func}(\sqrt{n}) \cdot f_{\max}$;
    if $\Lambda[i,j] < E_{inf}[i,j]$ **then**
      $\Theta[i,j] \leftarrow 0$, $\Theta[j,i] \leftarrow 0$;
      $\eta_0 \leftarrow \eta_0 + 1$;
    else if $\Lambda[i,j] > E_{sup}[i,j]$ **then**
      $\Theta[i,j] \leftarrow 1$, $\Theta[j,i] \leftarrow 1$;
      $\eta_1 \leftarrow \eta_1 + 1$;
Secondo ciclo: Calcolo di soglie adattive e aggiornamento di Θ
for $i \leftarrow 1$ **to** $n-1$ **do**
  for $j \leftarrow i+1$ **to** n **do**
    if $\Theta[i,j]$ ***is*** NaN **then**
      Calcola $f_{\max} \leftarrow \max(f_{\text{base}}, F[i,j], \boldsymbol{f_{hat}}[i], \boldsymbol{f_{hat}}[j])$;
      Calcola
       $E_{\text{inf}}[i,j] \leftarrow \text{gamma\_func}(2) \cdot \max(f_{\text{base}} + F[i,j], \boldsymbol{f_{hat}}[i] + \boldsymbol{f_{hat}}[j])$;
      Calcola $E_{\text{sup}}[i,j] \leftarrow \text{gamma\_func}(\sqrt{n}) \cdot f_{\max}$;
      Calcola soglia adattiva:

$$E[i,j] \leftarrow \frac{\eta_0}{\eta_0+\eta_1} \cdot E_{\text{inf}}[i,j] + \frac{\eta_1}{\eta_0+\eta_1} \cdot E_{\text{sup}}[i,j]$$

       if $\Lambda[i,j] > E[i,j]$ **then**
        $\Theta[i,j] \leftarrow 1$, $\Theta[j,i] \leftarrow 1$;
      else
        $\Theta[i,j] \leftarrow 0$, $\Theta[j,i] \leftarrow 0$;
return $\Theta, E_{inf}, E_{sup}, E$

Algorithm 7: ConnComp

Input: Θ: matrice di struttura binaria ($n \times n$), dove $\Theta[i, j] = 1$ indica un'interazione tra le variabili i e j.

Output: **Groups**: Lista di gruppi di variabili connesse.

$n \leftarrow$ dimensione righe di Θ `// numero di variabili decisionali.`
visited $\leftarrow \mathbf{0}^n$ `// Vettore booleano per tracciare i nodi visitati.`
Groups $\leftarrow \emptyset$ `// Lista per memorizzare i gruppi connessi.`

Definizione della funzione di supporto *DFS*:
    Funzione ricorsiva di ricerca in profondità: visita tutti i nodi collegati al nodo corrente, aggiorna il gruppo corrente e traccia i nodi visitati.
    Input: *node*, **current_group**
    Segna *node* come visitato: **visited**[*node*] $\leftarrow$ True;
    Aggiungi *node* a **current_group**;
    for *neighbor* $\leftarrow 0$ **to** $n - 1$ **do**
        if $\Theta[node, neighbor] = 1$ ***and visited***$[neighbor] = False$ **then**
            Richiama DFS(neighbor, **current_group**);
        end
    end

Algoritmo principale:
for $i \leftarrow 0$ **to** $n - 1$ **do**
    if ***visited***$[i] = False$ **then**
        current_group $\leftarrow \emptyset$;
        Richiama DFS(i, **current_group**);
        Aggiungi **current_group** a **Groups**;
    end
end
return *Groups*

Algorithm 8: Definizione della funzione gamma_func

Input: n: parametro per calcolare la sensibilità numerica (può rappresentare la dimensione del problema o la radice quadrata del numero di variabili).

Output: gamma_func(n): valore del margine di errore numerico basato sulla precisione della macchina e su n.

Definizione della funzione gamma_func(n):
$\mu_M \leftarrow \frac{\texttt{np.finfo(float).eps}}{2}$ `// Precisione numerica di macchina.`
gamma_func $= \frac{n \cdot \mu_M}{1 - n \cdot \mu_M}$ `// Calcola margine di errore basato su` μ_M `e` n.

Commenti sull'uso di gamma_func:
gamma_func(2): limite inferiore per stabilire l'assenza di interazione;
gamma_func($\sqrt{n}$): limite superiore per identificare la presenza di interazione;
return *gamma_func*

Algorithm 9: Differential Grouping (DG2) M.N. Omidvar [2017]

Input: n: numero di variabili decisionali,
$\mathbf{x_{lbar}}, \mathbf{x_{ubar}}$: limiti superiore e inferiore presi all'interno del dominio di f,
num_var_bin: numero di variabili binarie.
Output: g: numero di gruppi non separabili,
nonsep_groups: lista di gruppi di variabili interagenti,
$\mathbf{x_{sep}}$: lista di variabili separabili,
$\Lambda, F, \Gamma, \mathbf{f_{hat}}, f_{\text{base}}, \Theta, E_{\text{inf}}, E_{\text{sup}}, E$.

Passo 1: Calcolo della matrice di interazione Λ e strutture ausiliarie con ISM
$\Lambda, F, \mathbf{f_{hat}}, f_{\text{base}}, \Gamma \leftarrow \text{ISM}(n, \mathbf{x_{ubar}}, \mathbf{x_{lbar}}, num\_var\_bin)$;
Passo 2: Calcolo della matrice di struttura Θ con DSM
$\Theta, E_{\text{inf}}, E_{\text{sup}}, E \leftarrow \text{DSM}(\Lambda, F, \mathbf{f_{hat}}, f_{\text{base}}, n)$;
Passo 3: Identificazione dei gruppi connessi con ConnComp
groups $\leftarrow \text{ConnComp}(\Theta)$;
Passo 4: Classificazione dei gruppi in separabili e non separabili
$\mathbf{x_{sep}} \leftarrow \emptyset$ `// Variabili separabili (senza interazione).`
nonsep_groups $\leftarrow \emptyset$ `// Gruppi di variabili interagenti.`
$g \leftarrow 0$ `// Contatore dei gruppi non separabili.`
foreach ***group*** $\in$ ***groups*** **do**
  if $len($***group***$) = 1$ **then**
    Aggiungi **group** a $\mathbf{x_{sep}}$;
  else
    Aggiungi **group** a **nonsep_groups**;
    $g \leftarrow g + 1$;
return g, ***nonsep_groups***, $\boldsymbol{x_{sep}}, \Gamma, \Lambda, F, f_{hat}, f_{base}, \Theta, E_{inf}, E_{sup}, E$

4.3 Cooperative Coevolution (CC) con Gruppi di Variabili Identificati da DG2

Questa sezione illustra come l'algoritmo di raggruppamento differenziale (Differential Grouping, DG2) venga integrato in un framework di **Cooperative Coevolution (CC)**, illustrato nel lavoro M.N. Omidvar [2014], per risolvere problemi di ottimizzazione globale su larga scala.

L'algoritmo 10 descrive il framework *CC* utilizzato. Tale approccio prevede due fasi principali: una fase di raggruppamento (*grouping stage*) e una fase di ottimizzazione (*optimization stage*).

Nella **fase di raggruppamento** (Fase 1), viene individuata la struttura delle interazioni tra le variabili decisionali utilizzando il metodo DG2 descritto nell'algoritmo 9. Sebbene il framework consenta l'uso di qualsiasi procedura di raggruppamento, in questo caso viene impiegata la procedura basata su DG2.

Durante la **fase di ottimizzazione** (Fase 2-3), il framework esegue l'ottimizzazione delle sottocomponenti, che sono state identificate nella fase di raggruppamento. Questo processo avviene attraverso un approccio ciclico *round-robin*, in cui ciascuna sottocomponente viene ottimizzata singolarmente. Durante l'ottimizzazione di una sottocomponente, tutte le variabili decisionali al di fuori della sottocomponente corrente vengono trattate come fissate ai loro valori attuali. Ciò consente di ridurre significativamente la dimensionalità del problema di ottimizzazione gestito in ciascun passo, migliorando l'efficienza computazionale.

Il processo di ottimizzazione delle sottocomponenti viene ripetuto per un numero predeterminato di iterazioni, denominate *cycles*. In ogni ciclo, tutte le sottocomponenti vengono ottimizzate una alla volta, seguendo l'ordine definito nella fase di raggruppamento. Per ogni sottocomponente, viene utilizzato un algoritmo di ottimizzazione numerica specifico. Nel caso in questione, l'algoritmo scelto è OPUS.

Un aspetto cruciale di questo framework è la sua flessibilità e generalità. Il metodo di *Cooperative Coevolution (CC)* non è vincolato a un particolare algoritmo di ottimizzazione, ma consente l'integrazione di qualsiasi ottimizzatore numerico in grado di sfruttare le informazioni di raggruppamento derivate nella fase precedente. Questo rende il framework un approccio estremamente versatile, applicabile a una vasta gamma di problemi di ottimizzazione globale su larga scala. In questi contesti, la decomposizione del problema in sottocomponenti indipendenti o debolmente interconnesse risulta fondamentale per gestire la complessità computazionale. A tal fine, viene utilizzato il metodo DG2 come tecnica di decomposizione in gruppi, grazie alla sua capacità di individuare in modo efficace le interazioni tra le variabili decisionali e di suddividere il problema in sottocomponenti ottimizzabili separatamente.

È importante evidenziare che, come discusso in M.N. Omidvar [2014], assegnare uguale enfasi a tutte le sottocomponenti durante la fase di ottimizzazione potrebbe non rappresentare un uso ottimale del budget computazionale. Per affrontare queste inefficienze, l'algoritmo *CC* può essere configurato per integrare tecniche di ottimizzazione mirate, progettate per sfruttare efficacemente la struttura delle interazioni tra le variabili decisionali.

4.3.1 Applicazione dell'algoritmo OPUS nel Contesto del Cooperative Coevolution (CC)

Per integrare l'approccio di decomposizione, l'algoritmo OPUS è stato adattato per ottimizzare sottogruppi di variabili decisionali, mantenendo fisse le altre. Questo adattamento garantisce una gestione coerente tra le variabili ottimizzabili e quelle fissate, permettendo di affrontare problemi complessi mediante la suddivisione in sottoproblemi più semplici.

I dettagli tecnici dell'adattamento di OPUS, incluse le tre fasi principali (esplorazione, costruzione del modello surrogato e ottimizzazione), sono descritti nell'Appendice A1.

Algorithm 10: Cooperative Coevolution (CC) con OPUS

Input: $\mathbf{x_{lbar}}, \mathbf{x_{ubar}}$: llimiti superiore e inferiore presi all'interno del dominio di f,
n: numero di variabili decisionali,
num_var_bin: numero di variabili binarie,
cycles: numero di iterazioni del ciclo di ottimizzazione.
Output: **best**: miglior soluzione trovata,
best_val: valore della funzione obiettivo associato a **best**.

Fase 1: Identificazione dei gruppi di variabili
g, **nonsep_groups**, $\mathbf{x_{sep}} \leftarrow$ DG2($n, \mathbf{x_{lbar}}, \mathbf{x_{ubar}}$, num_var_bin);
groups $\leftarrow$ **nonsep_groups** $+ \mathbf{x_{sep}}$;
Inizializza **pop** $\leftarrow$ LHS_constr();
$\mathbf{f_{values}} \leftarrow$ calcola i valori obiettivo iniziali per ogni soluzione in **pop**;

Fase 2: Cooperative Coevolution se trovato un singolo gruppo
if *len(**groups**)* $= 1$ **then**
    `// Ottimizzazione diretta con OPUS se c'è solo un gruppo`
    Esegui OPUS su tutte le variabili;
    return ***best***, *best_val*;

Fase 2: Cooperative Coevolution per gruppi multipli
for *cycle* $\leftarrow 1$ **to** *cycles* **do**
    foreach ***group*** $\in$ ***groups*** **do**
        `// Definisce le variabili da ottimizzare con OPUS e quelle fisse`
        total_indices $\leftarrow \{0, \ldots, n-1\}$;
        fixed_indices $\leftarrow$ **total_indices** $\setminus$ **group**;
        opt_vars $\leftarrow$ **group**;
        `// Ottimizza il gruppo corrente con OPUS;`
        subpop $\leftarrow$ **pop**[:, **opt_vars**];
        subpop' $\leftarrow$ `optimizer_OPUS(subpop, OPUS_parameters)`;
        Aggiorna **pop**, $\mathbf{f_{values}}$ con i risultati del gruppo ottimizzato;
        pop[:, **group**] $\leftarrow$ **subpop'**;
        Valuta la nuova popolazione **pop** e salva i risultati in $\mathbf{f_{values}}$
    `// Salva il miglior valore corrente globale best_current`
    current_best_val $\leftarrow \min(\mathbf{f_{values}})$;
    best_current $\leftarrow$ soluzione corrispondente in **pop**;
    `// Aggiorna il minimo globale best`
    if *current_best_val* $<$ *best_val* **then**
        best_val $\leftarrow$ *current_best_val*;
        best $\leftarrow$ **best_current**;

return ***best***, *best_val*

4.4 Risultati Sperimentali: Approccio con Decomposizione vs. Approccio Senza Decomposizione

In questa sezione vengono presentati i risultati sperimentali ottenuti per confrontare l'approccio con decomposizione rispetto a quello senza decomposizione, applicati a un modello con 16 variabili decisionali. I parametri utilizzati per l'algoritmo OPUS sono stati mantenuti invariati nei due approcci e configurati come descritto nella Sottosezione 2.2.2 del Capitolo 2.
Il modello considerato rappresenta un caso realistico, basato su dati reali, ma con una semplificazione del problema originale per ridurre la complessità computazionale. Sono stati analizzati due store e tre prodotti (con codici "2445270", "3366370" e "6273515"), per un totale di 16 variabili decisionali, suddivise come segue:

- 2 **variabili binarie**, ciascuna associata a uno store, utilizzate per decidere se approvvigionarsi dal centro di distribuzione (DC) o direttamente dal fornitore.
- 5 **variabili continue**, rappresentanti le *safety stock* per le coppie prodotto-store. Una coppia non è stata considerata poiché uno dei due store non vende un determinato prodotto.
- 9 **variabili continue** relative al DC, suddivise in *safety stock*, *reorder level*, e *max level* per ciascuno dei tre prodotti.

4.4.1 Considerazioni sui Parametri k ed s

Durante i test, i parametri k (numero di punti iniziali) ed s (numero di particelle mantenute) hanno dimostrato di influenzare significativamente le performance dell'algoritmo OPUS. Le osservazioni principali sono:

1. **Numero di punti iniziali** k: Un aumento del numero di punti iniziali k richiede necessariamente un incremento proporzionale del numero di particelle mantenute s, al fine di garantire risultati attendibili. In caso contrario, l'algoritmo rischia di concentrarsi esclusivamente su una regione promettente dello spazio di ricerca, portando a una convergenza verso un minimo locale. Per questo motivo, nei test successivi riportati, è stato preferito utilizzare un valore ridotto di k, pari a 17, corrispondente al minimo numero necessario di punti iniziali $(n + 1)$, così da ridurre al minimo il costo computazionale.
2. **Numero di particelle mantenute** s: Il valore di s potrebbe essere scelto in base al caso d'uso specifico, determinando il valore ottimale come un trade-off tra il costo computazionale e l'efficacia della soluzione.

4.4.2 Considerazioni sui Test con Decomposizione

L'approccio con decomposizione, che utilizza una soglia automatica, si è rivelato troppo restrittivo, come già preannunciato in 4.2.2. In questo caso, infatti, il problema con $n = 16$

variabili viene suddiviso in un unico gruppo non separabile di $n = 14$ variabili e 2 variabili separabili.

Per ovviare a questo problema, si è scelto di rilassare la soglia utilizzata per determinare la decomposizione delle variabili in gruppi. Basandosi sulla matrice delle interazioni Λ, è stata adottata una soglia pari a 50: le variabili con un valore di interazione superiore a questa soglia sono considerate interagenti, mentre quelle con un valore inferiore sono considerate indipendenti. Tale configurazione ha permesso di ottenere un unico gruppo non separabile costituito da 10 variabili, mentre le restanti variabili sono tutte trattate come separabili.
In particolare la suddivisione trovata è la seguente:

1. **Gruppo 1:** Variabili [0, 1, 4, 8, 9, 11, 12, 14, 15]
   - Variabili binarie (store):
     - Variabile 0: Scelta di approvvigionamento per il primo store (DC o fornitore).
     - Variabile 1: Scelta di approvvigionamento per il secondo store (DC o fornitore).
   - Variabili continue (store e DC):
     - Variabile 4: *Safety stock* per la terza coppia prodotto-store (secondo store e prodotto "2445270").
     - Variabile 8: *Reorder level* per il prodotto "2445270" presso il DC.
     - Variabile 9: *Max level* per il prodotto "2445270" presso il DC.
     - Variabile 11: *Reorder level* per il prodotto "3366370" presso il DC.
     - Variabile 12: *Max level* per il prodotto "3366370" presso il DC.
     - Variabile 14: *Reorder level* per il prodotto "6273515" presso il DC.
     - Variabile 15: *Max level* per il prodotto "6273515" presso il DC.
2. **Gruppo 2:** Variabile [2]
   - Variabile continua (*safety stock*): *Safety stock* per la prima coppia prodotto-store (primo store e prodotto "2445270").
3. **Gruppo 3:** Variabile [3]
   - Variabile continua (*safety stock*): *Safety stock* per la seconda coppia prodotto-store (primo store e prodotto "3366370").
4. **Gruppo 4:** Variabile [5]
   - Variabile continua (*safety stock*): *Safety stock* per la quarta coppia prodotto-store (secondo store e prodotto "3366370").
5. **Gruppo 5:** Variabile [6]
   - Variabile continua (*safety stock*): *Safety stock* per la quinta coppia prodotto-store (primo store e prodotto "6273515").

6. **Gruppo 6:** Variabile [7]
   - Variabile continua (*safety stock*): *Safety stock* per il prodotto "2445270" presso il DC.
7. **Gruppo 7:** Variabile [10]
   - Variabile continua (*safety stock*): *Safety stock* per il prodotto "3366370" presso il DC.
8. **Gruppo 8:** Variabile [13]
   - Variabile continua (*safety stock*): *Safety stock* per il prodotto "6273515" presso il DC.

La matrice Λ ottenuta è la seguente:

$\Lambda =$

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
0	NaN	7.10	13.49	23.35	58.70	0.00	7.18	0.00	47.30	122.31	26.79	58.40	56.07	0.00	136.29	86.05
1	7.10	NaN	48.50	13.11	2.11	0.00	5.24	0.00	93.20	74.50	0.51	16.87	30.86	0.00	13.38	6.28
2	13.49	48.50	NaN	11.20	2.95	0.00	3.32	0.00	18.49	7.63	1.25	17.04	16.49	0.00	14.33	3.75
3	23.35	13.11	11.20	NaN	4.88	0.00	2.00	0.00	4.71	1.17	21.33	22.88	8.22	0.00	18.49	2.89
4	58.70	2.11	2.95	4.88	NaN	0.00	2.09	0.00	2.05	13.35	5.04	1.71	8.55	0.00	0.57	15.25
5	0.00	0.00	0.00	0.00	0.00	NaN	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
6	7.18	5.24	3.32	2.00	2.09	0.00	NaN	0.00	4.22	5.27	1.80	0.30	1.61	0.00	9.02	8.43
7	0.00	0.00	0.00	0.00	0.00	0.00	0.00	NaN	0.00	0.00	17.71	0.00	0.00	0.00	0.00	0.00
8	47.30	93.20	18.49	4.71	2.05	0.00	4.22	0.00	NaN	61.76	24.74	24.42	12.28	0.00	29.89	3.01
9	122.31	74.50	7.63	1.17	13.35	0.00	5.27	0.00	61.76	NaN	7.57	17.27	37.22	0.00	9.98	0.00
10	26.79	0.51	1.25	21.33	5.04	0.00	1.80	17.71	24.74	7.57	NaN	13.53	18.24	0.00	19.47	1.98
11	58.40	16.87	17.04	22.88	1.71	0.00	0.30	0.00	24.42	17.27	13.53	NaN	45.45	0.00	27.39	4.95
12	56.07	30.86	16.49	8.22	8.55	0.00	1.61	0.00	12.28	37.22	18.24	45.45	NaN	0.00	12.79	10.44
13	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	NaN	0.00	0.00
14	136.29	13.38	14.33	18.49	0.57	0.00	9.02	0.00	29.89	9.98	19.47	27.39	12.79	0.00	NaN	63.14
15	86.05	6.28	3.75	2.89	15.25	0.00	8.43	0.00	3.01	0.00	1.98	4.95	10.44	0.00	63.14	NaN

Questa suddivisione riflette la struttura e le interazioni tra le variabili decisionali, con un unico gruppo contenente la maggior parte delle variabili correlate di cardinalità pari a 10, mentre le rimanenti variabili, con interazioni più deboli, sono considerate separabili.

4.4.3 Risultati Sperimentali con e senza Decomposizione

Dettagli sul numero di particelle e il parametro *patience*

Nel corso del processo di ottimizzazione e per tutti gli esperimenti computazionali con decomposizione testati, sono state adottate diverse strategie per la gestione del numero di particelle, al fine di ottimizzare le prestazioni computazionali e migliorare l'efficacia dell'algoritmo. In particolare, il numero di particelle è stato ridotto dopo il primo ciclo di ottimizzazione per ciascun gruppo, nonché durante il primo ciclo per i gruppi costituiti da una sola variabile. La riduzione è avvenuta nei seguenti modi:

- **Problemi di dimensione unitaria ($n = 1$):** Per i gruppi con una sola variabile e in tutti i cicli di ottimizzazione, il numero di particelle è stato ridotto da 16 o 10 (a seconda della configurazione iniziale di s impostata) a 5. Questa riduzione è stata ritenuta adeguata in quanto, anche con il numero ridotto di particelle, il valore rimane maggiore del numero di variabili da ottimizzare.

- **Cicli di ottimizzazione successivi al primo** (*cycles* > 1)**:** Per ridurre il costo computazionale, nei cicli successivi al primo, il numero di particelle è stato diminuito anche per il gruppo di dimensione maggiore. Dopo una prima ottimizzazione con 16 o 10 particelle, sono state mantenute solo le 5 migliori particelle, selezionate in base alla funzione obiettivo.

Per quanto riguarda il parametro *patience*, sono state adottate configurazioni differenti in base alla dimensione dei gruppi ottimizzati:

- **Gruppo con più variabili:** Il parametro *patience* è stato impostato a 10, al fine di consentire un numero maggiore di iterazioni per migliorare l'ottimizzazione, dato che i gruppi più complessi necessitano di più tempo per convergere.
- **Gruppi con variabili singole:** Il parametro *patience* è stato ridotto a 5, poiché il problema risulta più semplice da risolvere e richiede un minor numero di iterazioni per raggiungere una soluzione ottimale.

Si procede ora a illustrare in dettaglio i principali risultati sperimentali ottenuti.

1. **Caso senza decomposizione**

   Il primo test è stato effettuato con $k = 17$ punti iniziali e $s = 16$ particelle mantenute. L'algoritmo OPUS è stato eseguito senza decomposizione, utilizzando l'intero set di 16 variabili decisionali. Di seguito vengono riportati i risultati ottenuti:

   - **Configurazione ottimale (Best_x):**


```
{
    "stores": {
        "1": 0,
        "2": 0
    },
    "item_store_ss": {
        "1": 70,
        "2": 47,
        "3": 120,
        "4": 59,
        "5": 39
    },
    "dc_items": {
        "2445270": {
            "safety_stock": 97,
            "reorder_level": 187,
            "max_level": 1708
        },
        "3366370": {
```

```
            "safety_stock": 0,
            "reorder_level": 2,
            "max_level": 1500
        },
        "6273515": {
            "safety_stock": 179,
            "reorder_level": 179,
            "max_level": 992
        }
    }
}
```


- **Valore ottimale della funzione obiettivo** ($f_{\mathbf{best}}$)**:** 1269.01
- **Evoluzione della** $f_{\mathbf{best}}$ **durante le iterazioni:**

  $[1681.91, 1667.91, 1641.55, 1641.55, 1618.65, 1580.0, 1448.28, 1448.28,$
  $1448.28, 1448.28, 1448.28, 1448.28, 1448.28, 1448.28, 1419.47, 1419.47,$
  $1386.63, 1386.63, 1367.14, 1367.14, 1342.67, 1342.67, 1342.67, 1342.67,$
  $1326.48, 1274.35, 1269.03, 1269.03, 1269.03, 1269.03, 1269.03, 1269.03,$
  $1269.03, 1269.03, 1269.03, 1269.01]$

- **Numero di iterazioni totali:** 35
- **Numero di valutazioni di funzione:** 612
- **Tempo totale (in secondi):** 7726.57

2. **Caso con decomposizione,** $s = 16$ **e** $cycles = 5$

   Nel secondo test, l'approccio con decomposizione è stato applicato suddividendo le variabili in gruppi. La configurazione utilizzata prevedeva $k = 17$ punti iniziali e $s = 16$ particelle mantenute durante il primo ciclo di ottimizzazione. Per i gruppi di variabili singole, il numero di particelle è stato impostato a $s = 5$ in tutti i cicli di ottimizzazione. Per il gruppo di variabili, invece, si è utilizzato $s = 5$ a partire dal secondo ciclo di ottimizzazione. I risultati ottenuti sono i seguenti:

   - **Configurazione ottimale (Best_x):**


```
{
    "stores": {
        "1": 0,
        "2": 0
    },
    "item_store_ss": {
        "1": 70,
```

```
            "2": 56,
            "3": 72,
            "4": 46,
            "5": 41
        },
        "dc_items": {
            "2445270": {
                "safety_stock": 0,
                "reorder_level": 278,
                "max_level": 1979
            },
            "3366370": {
                "safety_stock": 2,
                "reorder_level": 2,
                "max_level": 1500
            },
            "6273515": {
                "safety_stock": 24,
                "reorder_level": 24,
                "max_level": 1000
            }
        }
    }
```


- **Valore ottimale della funzione obiettivo** ($f_{\mathbf{best}}$)**:** 1266.24
- **Evoluzione della** $f_{\mathbf{best}}$ **durante i cicli di ottimizzazione:**

$$[1517.56, 1437.62, 1355.81, 1266.24, 1266.24]$$

  - **Numero di Iterazioni per ciascun gruppo all'interno di ogni ciclo:**

$$\begin{bmatrix} 20 & 5 & 5 & 5 & 5 & 5 & 5 & 5 \\ 14 & 5 & 5 & 5 & 5 & 5 & 5 & 5 \\ 17 & 5 & 5 & 5 & 5 & 5 & 5 & 5 \\ 13 & 5 & 5 & 5 & 5 & 5 & 5 & 5 \\ 10 & 5 & 5 & 5 & 5 & 5 & 5 & 5 \end{bmatrix}$$

  - **Valutazioni di funzione per ciascun gruppo all'interno di ogni ciclo:**

$$\begin{bmatrix} 340 & 30 & 30 & 30 & 30 & 30 & 30 & 30 \\ 84 & 30 & 30 & 30 & 30 & 30 & 30 & 30 \\ 102 & 30 & 30 & 30 & 30 & 30 & 30 & 30 \\ 78 & 30 & 30 & 30 & 30 & 30 & 30 & 30 \\ 60 & 30 & 30 & 30 & 30 & 30 & 30 & 30 \end{bmatrix}$$

- **Numero di iterazioni totali:** 249
- **Numero di valutazioni di funzione:** 1731
- **Tempo totale (in secondi):** 22703.80

3. **Caso con decomposizione,** $s = 10$ **e** $cycles = 5$

   Nel terzo test, l'approccio con decomposizione è stato applicato suddividendo le variabili in gruppi. La configurazione utilizzata prevedeva $k = 17$ punti iniziali e $s = 10$ particelle mantenute durante il primo ciclo di ottimizzazione. Per i gruppi di variabili singole, il numero di particelle è stato impostato a $s = 5$ in tutti i cicli di ottimizzazione. Per il gruppo di variabili, invece, si è utilizzato $s = 5$ a partire dal secondo ciclo di ottimizzazione. I risultati ottenuti sono i seguenti:

   - **Configurazione ottimale (Best_x):**


```
{
    "stores": {
        "1": 0,
        "2": 0
    },
    "item_store_ss": {
        "1": 66,
        "2": 44,
        "3": 68,
        "4": 9,
        "5": 43
    },
    "dc_items": {
        "2445270": {
            "safety_stock": 0,
            "reorder_level": 209,
            "max_level": 1054
        },
        "3366370": {
            "safety_stock": 0,
            "reorder_level": 271,
            "max_level": 1222
        },
        "6273515": {
            "safety_stock": 76,
            "reorder_level": 182,
            "max_level": 1000
        }
    }
}
```


   - **Valore ottimale della funzione obiettivo** ($f_{\mathbf{best}}$)**:** 1420.16
   - **Evoluzione della** $f_{\mathbf{best}}$ **durante i cicli di ottimizzazione:**

$$[1594.75, 1422.1, 1420.16, 1420.16, 1420.16]$$

– **Numero di Iterazioni per ciascun gruppo all'interno di ogni ciclo:**

$$\begin{bmatrix} 13 & 5 & 5 & 5 & 5 & 5 & 5 & 5 \\ 21 & 5 & 5 & 5 & 5 & 5 & 5 & 5 \\ 10 & 5 & 5 & 5 & 5 & 5 & 5 & 5 \\ 10 & 5 & 5 & 5 & 5 & 5 & 5 & 5 \\ 10 & 5 & 5 & 5 & 5 & 5 & 5 & 5 \end{bmatrix}$$

– **Valutazioni di funzione per ciascun gruppo all'interno di ogni ciclo:**

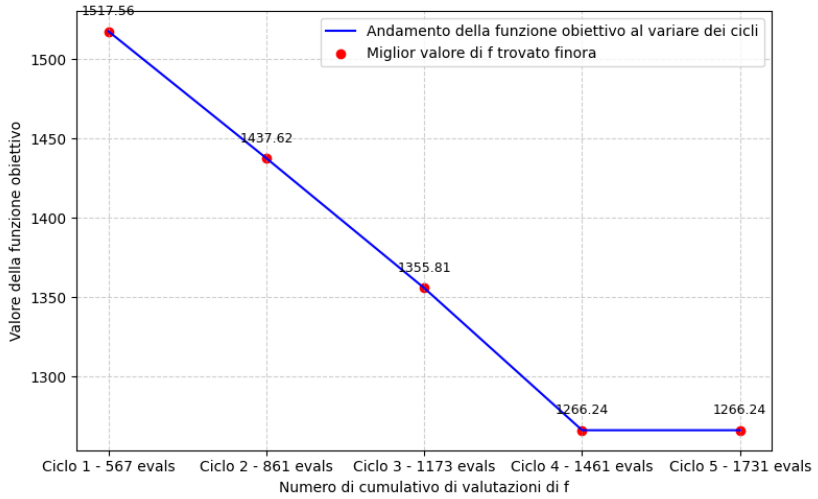
$$\begin{bmatrix} 143 & 30 & 30 & 30 & 30 & 30 & 30 & 30 \\ 126 & 30 & 30 & 30 & 30 & 30 & 30 & 30 \\ 60 & 30 & 30 & 30 & 30 & 30 & 30 & 30 \\ 60 & 30 & 30 & 30 & 30 & 30 & 30 & 30 \\ 60 & 30 & 30 & 30 & 30 & 30 & 30 & 30 \end{bmatrix}$$

- **Numero di iterazioni totali:** 239
- **Numero di valutazioni di funzione:** 1516
- **Tempo totale (in secondi):** 18324.90

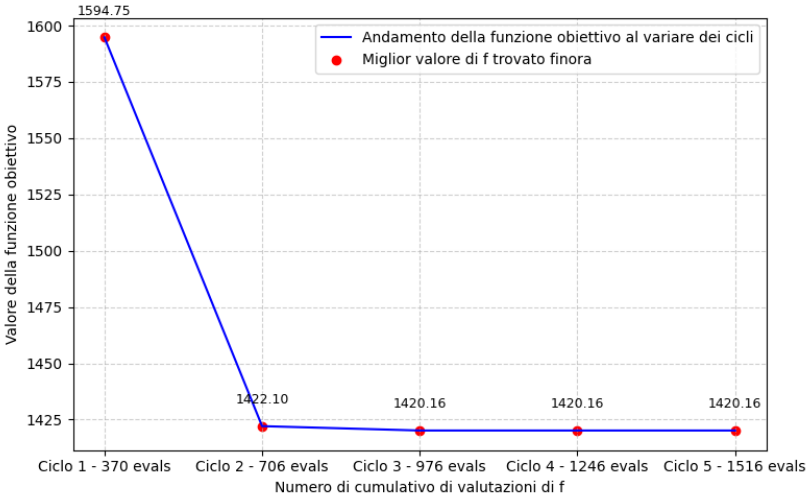
4.4.4 Analisi dei Risultati

I migliori risultati sono stati ottenuti con $s = 16$ e $cycles = 5$. Dai test effettuati emergono le seguenti osservazioni:

- **Dimensionalità e numero di particelle:** Il caso con $n = 10$ rappresenta la dimensionalità del sotto-problema di ottimizzazione più grande, ovvero il gruppo con 10 variabili. Questa configurazione è stata testata per valutare la possibilità di ridurre il numero di particelle mantenute dall'algoritmo, passando dal valore pari alla dimensionalità del problema originario ($s = 16$) a quello del sotto-problema di cardinalità maggiore ($n = 10$), con l'obiettivo di ridurre il tempo computazionale. Dai risultati sperimentali emerge che, con $s = 10$ particelle, l'algoritmo ottiene un risultato accettabile, pari a $f_{\text{best}} = 1420.16$ ma non riesce a raggiungere l'ottimo globale. Al contrario, quando $s = 16$, il numero maggiore di particelle consente di ottenere un risultato migliore, sebbene l'algoritmo converga più lentamente.
- **Influenza del parametro *cycles*:** Incrementando il valore del parametro *cycles* (numero di cicli di ottimizzazione su tutte le variabili), si osserva un miglioramento nei risultati. Questo accade perché, nei cicli successivi al primo, i gruppi di variabili vengono ottimizzati tenendo conto dei livelli o gruppi già ottimizzati. Tuttavia, il miglioramento si arresta superata una certa soglia:
  – Con $s = 16$, non si osservano ulteriori miglioramenti dopo il quarto ciclo di ottimizzazione, come si può notare in 4.1a.
  – Con $s = 10$, i miglioramenti si fermano dopo il terzo ciclo di ottimizzazione, come si può notare in 4.1b.

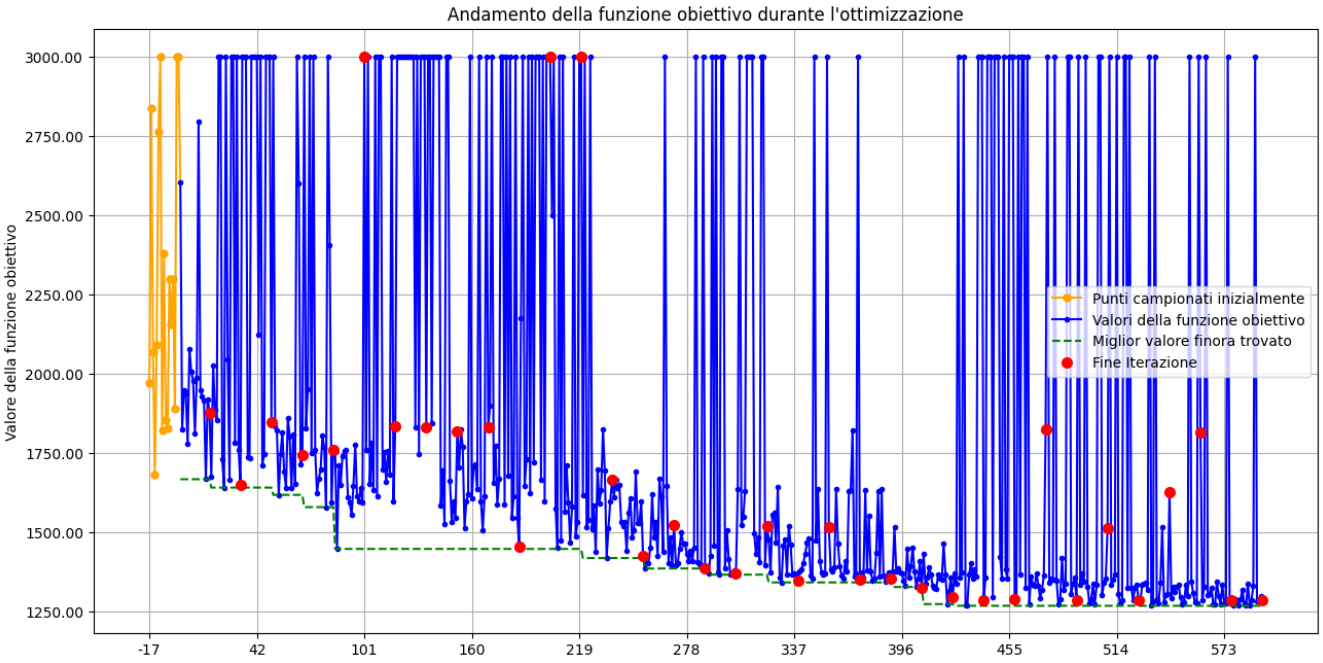


(a) Andamento del miglior valore f_{best} trovato al variare dei cicli di ottimizzazione per il caso 2 con decomposizione, con s = 16.

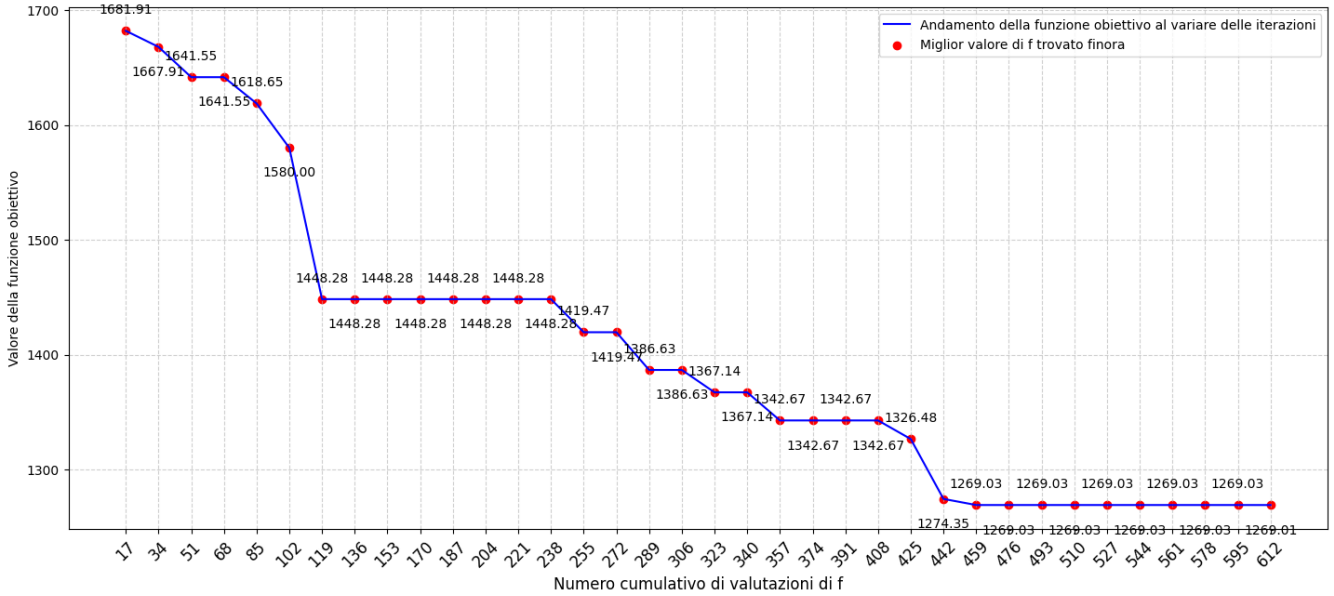


(b) Andamento del miglior valore f_{best} trovato al variare dei cicli di ottimizzazione per il caso 3 con decomposizione, con s = 10.

- **Confronto con l'approccio senza decomposizione:** Con $s = 16$ e $cycles = 5$, si ottiene un risultato molto simile a quello del caso senza decomposizione, dimostrando l'efficacia dell'approccio proposto. Nel caso senza decomposizione, infatti, si ottiene un valore di $f_{\text{best}} = 1269.01$, mentre nel caso con decomposizione il valore raggiunto è $f_{\text{best}} = 1266.24$. In questo caso specifico, la dimensione originaria del problema ($s = 16$) è ancora relativamente bassa, consentendo all'algoritmo OPUS di gestire il problema senza necessità di decomposizione. In Regis [2014], infatti, è stato dimostrato sperimentalmente che OPUS è in grado di gestire efficacemente problemi di ottimizzazione con una dimensionalità fino a 30–32 variabili.



(a) Andamento della funzione obiettivo rispetto al numero di valutazioni della funzione. Il grafico è stato ottenuto utilizzando l'algoritmo OPUS nel caso 1 senza decomposizione con $k = 17$ ed $s = 16$.



(b) Andamento del miglior valore di f_{best} trovato al variare delle iterazioni per il caso 1 senza decomposizione, con $k = 17$ ed $s = 16$, estrapolato da 4.2a.

- **Convenienza computazionale:** Osservando i risultati in termini di costo computazionale, emerge che risulta conveniente utilizzare OPUS senza decomposizione, come si può notare in 4.2a e 4.2b. Questa scelta è supportata dai tempi computazionali riportati nei risultati sperimentali, considerando che nei casi con decomposizione occorre tenere conto anche del costo computazionale aggiuntivo associato alla suddivisione delle variabili in gruppi. Tuttavia, per problemi di dimensioni più elevate, l'approccio con decomposizione risulta più conveniente in termini di efficacia della soluzione. Questo avviene perché, in presenza di problemi ad alta dimensionalità, come discusso in 4.1, non è garantito che OPUS converga. Decomponendo il problema, si riduce la dimensionalità, semplificando il processo di ottimizzazione e riuscendo a raggiungere soluzioni di qualità.

In conclusione, per problemi ad alta dimensionalità affrontati senza decomposizione, non solo la convergenza dell'algoritmo non è garantita, ma risulta spesso impraticabile ottimizzare direttamente su tutte le variabili decisionali, a causa dell'eccessivo numero di variabili coinvolte; quindi, l'approccio basato sulla decomposizione si rivela particolarmente vantaggioso ed efficace per affrontare e risolvere tali problemi.

D'altro canto, come mostrato dai risultati sperimentali, l'approccio senza decomposizione risulta più adatto per problemi di bassa dimensionalità, grazie al suo minor costo computazionale, che lo rende una scelta preferibile in contesti meno complessi.

Conclusioni

La presente tesi si è focalizzata sull'ottimizzazione dei parametri di un modello di simulazione di una supply-chain, affrontando sfide metodologiche e computazionali legate ai problemi black-box. È stato sviluppato un framework basato sull'algoritmo OPUS, integrato con tecniche di decomposizione, con l'obiettivo di migliorare l'efficienza computazionale e la qualità delle soluzioni ottenute.

Nel Capitolo 1 sono state introdotte le basi teoriche dell'ottimizzazione basata su simulazione, con un focus sui problemi black-box e sui modelli surrogati, come le Radial Basis Functions, oltre a tecniche di campionamento come il Latin Hypercube Sampling, fondamentali per la costruzione di modelli accurati.

Nel Capitolo 2 è stato analizzato l'algoritmo OPUS, confrontandolo con metodi di ricerca diretta come PSO e DE. I risultati mostrano che l'uso di un modello surrogato riduce significativamente le valutazioni necessarie della funzione obiettivo, mantenendo però un elevato livello di precisione, il che rende OPUS particolarmente vantaggioso in contesti ad alto costo computazionale.

Nel Capitolo 3, l'algoritmo OPUS è stato applicato a un modello semplificato di supply-chain, dimostrando la sua capacità di adattarsi efficacemente ai cambiamenti dei parametri di costo e bilanciare le componenti del sistema. I risultati sperimentali confermano la validità del framework, evidenziando la robustezza dell'approccio anche in scenari caratterizzati da variazioni nei parametri.

Il Capitolo 4 ha analizzato la decomposizione di problemi black-box, esplorando l'approccio Differential Grouping (DG2) combinato con il Cooperative Coevolution (CC). I risultati mostrano che, per una dimensionalità di $s = 16$, l'approccio con decomposizione raggiunge prestazioni comparabili a quelle senza decomposizione, sebbene con un costo computazionale aggiuntivo. Tuttavia, la decomposizione offre vantaggi teorici per problemi caratterizzati da un numero elevato di variabili decisionali, che, in assenza di tale approccio, risulterebbero intrattabili dal punto di vista computazionale. Al contrario, per problemi a bassa dimensionalità, l'approccio senza decomposizione si dimostra più vantaggioso.

4.5 Sviluppi Futuri

Un possibile sviluppo futuro riguarda la progettazione e l'adozione di una strategia di decomposizione parallela, in alternativa a quella sequenziale utilizzata in questo lavoro. Tale approccio potrebbe ridurre significativamente i tempi computazionali necessari, migliorando l'efficienza anche in problemi caratterizzati da bassa dimensionalità. Questa evoluzione potrebbe ottimizzare ulteriormente il processo, mantenendo la capacità di gestire problemi complessi e aumentando l'efficacia complessiva dell'approccio.

In conclusione, il lavoro svolto contribuisce a esplorare nuove direzioni nell'ottimizzazione basata su simulazione, offrendo spunti per future ricerche e applicazioni. Ulteriori studi potrebbero essere utili per affinare le tecniche proposte e per validarle in contesti ancora più complessi.

Appendice

A1 Adattamento dell'Algoritmo OPUS al caso con decomposizione

L'approccio con decomposizione proposto nasce dall'esigenza di ottimizzare problemi complessi suddividendoli in sottoproblemi più semplici, ciascuno focalizzato su un sottoinsieme delle variabili decisionali. Tuttavia, questa metodologia richiede un adattamento specifico dell'algoritmo OPUS, poiché quest'ultimo, essendo progettato per lavorare come una funzione *black-box*, accetta in input un numero predefinito e invariabile di parametri, che corrisponde a tutte le variabili decisionali del problema.

Necessità di adattamento

Con l'approccio di decomposizione, l'obiettivo è ottimizzare solo un sottoinsieme delle variabili decisionali alla volta, mantenendo le restanti fissate a valori predeterminati. Per far funzionare correttamente OPUS in questo contesto, è necessario:

1. Passare come input al simulatore sia le variabili da ottimizzare che quelle fissate.
2. Garantire che, durante l'intero processo di ottimizzazione, l'algoritmo gestisca in modo coerente questa distinzione tra variabili ottimizzabili e variabili fissate.

Adattamento dell'algoritmo OPUS

L'adattamento di OPUS si articola in tre fasi principali:

1. **Fase di esplorazione:** Durante la fase di esplorazione dello spazio delle soluzioni, l'algoritmo genera punti candidati considerando solo le variabili da ottimizzare, mentre le variabili fissate sono mantenute costanti nei punti campionati.
2. **Costruzione del modello surrogato:** Il modello surrogato viene costruito tenendo conto della suddivisione tra variabili ottimizzabili e fissate. Ciò implica che il modello è addestrato su dati che includono entrambe le categorie di variabili, ma le variazioni avvengono solo sulle variabili da ottimizzare.

3. **Ottimizzazione del modello surrogato:** La ricerca della soluzione ottimale del modello surrogato è limitata allo spazio delle variabili ottimizzabili, mentre le variabili fissate restano invariate. Questo assicura che il modello approssimi correttamente la funzione obiettivo considerando la decomposizione adottata.

Vantaggi del riadattamento

Questo adattamento di OPUS consente di:

- Integrare l'approccio di decomposizione, permettendo di ottimizzare separatamente sottoinsiemi di variabili senza perdere di vista il problema globale.
- Garantire una gestione coerente tra variabili ottimizzabili e fissate in tutte le fasi dell'algoritmo, migliorando la robustezza del processo.
- Concentrare l'ottimizzazione solo sulle variabili rilevanti per ciascun sottoproblema.

I dettagli relativi all'implementazione sono presentati nel seguente codice Python, riportato nella listing A1.

```
def OPUS_optimizer(f_values, a, b, lower_b, upper_b, fixed_vars, z,
    check_constraints, s, i_t, mu, nu, v_min, v_max, T_max, r, eps,
    delta, patience, json_filepath, exe_path, num_var_bin):
    """
    Funzione OPUS adattata per ottimizzare solo le variabili
        specificate, mantenendo fisse le altre,
    e per restituire l'intera popolazione aggiornata delle migliori
        particelle trovate.
    """
    # Numero di variabili nella sottopopolazione da ottimizzare
    d = len(a)
    # Numero totale di variabili decisionali
    n = z.shape[1]
    print(f"s = {s}")

    # Identificazione degli indici delle variabili da ottimizzare e
        di quelle fisse
    all_indices = np.arange(n)
    non_fixed_indices = np.setdiff1d(all_indices, fixed_vars['
        group_indices'])

    # Trovare le posizioni iniziali dello sciame basate sui migliori
        punti
    indices_pop_best = np.argsort(f_values)[:s]  # Ordina i valori e
        prende i migliori
    pop_best = np.array(z[indices_pop_best])  # Le migliori posizioni
         iniziali
    y_pop_best = np.array(z[indices_pop_best])  # Copia delle
        posizioni migliori
    f_pop_best = f_values[indices_pop_best]  # I valori associati
        alle migliori posizioni

```

```
22      # Posizioni iniziali per l'ottimizzazione, solo variabili
            decisionali non fisse
23      x = np.array([z[i][non_fixed_indices] for i in indices_pop_best])
24      y = np.copy(x)
25      f_x = f_values.copy()
26      f_y = np.copy(f_pop_best)
27
28      # Trova la miglior posizione iniziale
29      y_best = z[np.argmin(f_x)][non_fixed_indices]
30      index_best = np.argmin(f_pop_best)
31      z_best = z[np.argmin(f_x)]
32      f_best = np.min(f_x)
33
34      # Inizializza la velocita' per le particelle
35      v = [0.5 * (np.random.uniform(a, b, d) - x_i) for x_i in x]
36
37      # Inizializzazione del training set per il modello surrogato
38      x_train = z.copy()
39      y_train = f_x.copy()
40
41      # Variabili per il controllo delle iterazioni
42      t = 0
43      no_improvement_count = 0
44
45      # Ciclo di ottimizzazione fino al massimo numero di iterazioni
46      while t < T_max:
47          # Liste per memorizzare le migliori posizioni e velocita' in
                ogni iterazione
48          x_best, v_best = [], []
49          print(f"Iterazione num. t: {t}")
50
51          # Ciclo per ogni particella
52          for i in range(s):
53              trial_positions, trial_velocities = [], []
54              for _ in range(r):
55                  # Generazione di coefficienti casuali per aggiornare
                        la velocita'
56                  omega1 = np.random.uniform(0, 1, d)
57                  omega2 = np.random.uniform(0, 1, d)
58
59                  # Aggiornamento della velocita' usando la formula di
                        PSO
60                  v_trial = (i_t * v[i] + mu * omega1 * (y[i] - x[i]) +
                         nu * omega2 * (y_best - x[i]))
61                  v_trial = np.clip(v_trial, v_min, v_max)  #
                        Limitazione della velocita'
62
63                  # Calcolo della posizione di prova (trial)
64                  x_trial = np.copy(pop_best[i])
65                  x_trial[non_fixed_indices] = np.clip(x[i] + v_trial,
                        a, b)  # Limitazione della posizione
66                  x_trial = check_constraints(x_trial)  # Verifica dei
                        vincoli
```

```
67                  trial_positions.append(x_trial)
68                  trial_velocities.append(v_trial)
69
70              # Uso di un surrogato RBF per valutare la posizione migliore tra le prove
71              trial_positions = np.array(trial_positions)
72              trial_velocities = np.array(trial_velocities)
73              f_surrogate = RBF.RBF_surrogate(x_train, y_train, trial_positions)
74              best_index = np.argmin(f_surrogate)  # Trova l'indice della posizione migliore
75              x_best.append(trial_positions[best_index])  # Salva la posizione migliore
76              v_best.append(trial_velocities[best_index])  # Salva la velocita' corrispondente
77
78          # Aggiorna la posizione migliore globale
79          for i in range(s):
80              pop_best[i] = x_best[i]
81
82          # Valutazione delle nuove posizioni tramite simulazione reale
83          results_best = [simulation_objective(x_i, json_filepath, exe_path) for x_i in pop_best]
84          f_x_best = np.array([r[0] for r in results_best])  # Prende i valori di fitness
85
86          # Aggiorna il training set per il surrogato
87          x_train = np.vstack([x_train, pop_best])
88          y_train = np.concatenate([y_train, f_x_best])
89          x_train, unique_indices = np.unique(x_train, axis=0, return_index=True)  # Rimuove duplicati
90          y_train = y_train[unique_indices]
91
92          # Controllo miglioramenti
93          improved = False
94          for i in range(s):
95              x[i] = x_best[i][non_fixed_indices]  # Aggiorna la posizione
96              v[i] = v_best[i]  # Aggiorna la velocita'
97              if f_x_best[i] < f_y[i]:  # Se la nuova posizione e' migliore
98                  y[i] = x_best[i][non_fixed_indices]
99                  f_y[i] = f_x_best[i]
100                 y_pop_best[i] = x_best[i]
101                 pop_best[i] = x_best[i]
102                 if f_x_best[i] < f_best:
103                     improved = True
104                     z_best = x_best[i]
105                     y_best = x_best[i][non_fixed_indices]
106                     f_best = f_x_best[i]
107                     index_best = i
108
109         # Se non ci sono miglioramenti, incrementa il contatore
```

```
110         no_improvement_count = 0 if improved else
                no_improvement_count + 1
111
112         # Se il contatore supera il limite di pazienza, interrompi
113         if no_improvement_count >= patience:
114             print(f"Interruzione anticipata dopo {t} iterazioni senza
                    miglioramenti.")
115             break
116         t += 1
117
118     # Aggiorna la popolazione con le migliori soluzioni trovate
119     z[indices_pop_best] = y_pop_best
120     f_values[indices_pop_best] = f_y
121     evals = (t + 1) * (s + 1)  # Numero di valutazioni effettuate
122
123     return z, f_values, t + 1, evals
```

Listing 1: Adattamento di OPUS per gestire il caso con decomposizione

Bibliografia

Q. Wang A. Kiuchi, H. Wang. Bayesian optimization algorithm with agent-based supply chain simulator for multi-echelon inventory management. *IEEE 16th International Conference on Automation Science and Engineering (CASE)*, 2020.

A.J. Keane A.I.J. Forrester, A. Sóbester. *Engineering Design via Surrogate Modelling*. 2008.

L. Yuanxiang B. Wei. Local stable mechanism for particle swarm optimization algorithm. *IEEE Transactions on Evolutionary Computation*, pages 466–473, 2012.

J.J. Liang B.Y. Qu. Niching particle swarm optimization with local search for multi-modal optimization. *Information Sciences*, 197:131–143, 2012.

H.M. Gutmann. A radial basis function method for global optimization. *Journal of Global Optimization*, 19:201–227, 2001.

Z. Xu J.R. Gardner, M.J. Kusner. Bayesian optimization with inequality constraints. *International Conference on Machine Learning*, 2014.

M. Sciandrone L. Grippo. *Metodi di ottimizzazione non vincolata*. 2011.

J. Jager M. Buhmann. On radial basis functions. *Cambridge University Press*, 9:1–38, 2001.

B. Iooss M. Petelet. Latin hypercube sampling with inequality constraints. *AStA Advances in Statistical Analysis*, 94:325–339, 2010.

Y. Mei M.N. Omidvar, M. Yang. Dg2: A faster and more accurate differential grouping for large-scale black-box optimization. *IEEE Transactions on Evolutionary Computation*, 21(6):929–942, 2017.

Y. Mei M.N. Omidvar, X. Li. Cooperative co-evolution with differential grouping for large scale optimization. *IEEE Transactions on Evolutionary Computation*, 18(3):378–393, 2014.

M.J.D. Powell. *The Theory of Radial Basis Function Approximation in 1990s*. 1992.

R.G. Regis. Particle swarm with radial basis function surrogates for expensive black-box optimization. *Journal of Computational Science*, 5:12–23, 2014.

C.A. Shoemaker R.G. Regis. Local function approximation in evolutionary algorithms for the optimization of costly functions. *IEEE Transactions on Evolutionary Computation*, 8(5):490–505, 2004.

J. Rudholm S. Jakobsson, M. Patriksson. A method for simulation based optimization using radial basis functions. *Optimization Engineering*, 11:501–532, 2010.

A. Seifi T. Foroud, A. Baradaran. A comparative evaluation of global search algorithms in black box optimization of oil production: A case study on brugge field. *Journal of Petroleum Science and Engineering*, 167:131–151, 2018.

O. Wang T. Ogura, H. Wang. Bayesian optimization methods for inventory control with agent based supply-chain simulator. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, E105.A(9):1348–1357, 2022.

J. Wei Y. Tang, J. Chen. A surrogate-based particle swarm optimization algorithm for solving optimization problems with expensive black box functions. *Taylor & Francis Online*, 2012.