Università degli Studi di Padova

DIPARTIMENTO DI MATEMATICA "TULLIO LEVI-CIVITA"
Corso di Laurea Magistrale in Matematica

Tesi di laurea magistrale

Un algoritmo senza derivate per problemi con variabili intere

Candidato:
Monica Bego
Matricola 1176512

Relatore:
Prof. Francesco Rinaldi

Indice

Abstract

Nella tesi seguente verrà presentata una modifica ad un algoritmo per la risoluzione di problemi black box a variabili intere. I *problemi black box a variabili intere* rappresentano un'importante sfida ancora aperta per la Ricerca Operativa. La caratteristica principale di tali problemi risiede nella non-analiticità delle funzioni che li descrivono. Tale non-analiticità porta all'impossibilità di una conoscenza delle derivate di queste funzioni. Di conseguenza i problemi black box a variabili intere non possono essere affrontati con i tradizionali algoritmi basati sulle derivate. L'insorgere di problemi black box a variabili intere come modellizzazione di problemi reali ha dato quindi origine all'esigenza di sviluppare opportuni algoritmi volti alla loro risoluzione.
In questa tesi i problemi black box a variabili intere verranno introdotti nel *Capitolo 1*, ponendo l'attenzione sulle loro peculiarità e sulle caratteristiche che un algoritmo volto alla loro risoluzione deve avere, nel *Capitolo 2* saranno presentati alcuni algoritmi che negli anni sono stati proposti per cercare soluzioni a questi problemi, mentre nel *Capitolo 3* verrà presentato un algoritmo di più recente costruzione: l'algoritmo NM-BBOA. Una modifica all'algoritmo NM-BBOA sarà presentata in dettaglio nel *Capitolo 4*, per poi concludere con il *Capitolo 5* dove verranno riportati i risultati ottenuti dal confronto tra l'algoritmo base e la sua versione modificata.

Capitolo 1

Introduzione ai metodi senza derivate per problemi black box con variabili intere

Si consideri un problema di ricerca di un punto di minimo di una funzione f su un insieme $X \subseteq \mathbb{R}^n$

$$\begin{aligned} &min\ f(x) \\ &x \in X \end{aligned} \tag{1.1}$$

con f differenziabile e $\nabla f(x)$ facilmente calcolabile. I metodi di ottimizzazione che vengono solitamente impiegati per la sua risoluzione sono metodi che fanno uso delle derivate della funzione f, i quali vengono detti *metodi basati sulle derivate*. Qualora invece $\nabla f(x)$ non risulti facilmente calcolabile i metodi basati sulle derivate sono inutilizzabili. Per ovviare a questo problema, dagli anni 50 del '900 [7], hanno cominciato a prendere piede diversi metodi sviluppati senza l'uso delle derivate, noti come *metodi senza derivate* o *DFO methods*.

Una classe importante di problemi in cui i metodi senza derivate hanno recentemente trovato impiego è rappresentata dai problemi di tipo black box con variabili intere, che ora introdurremo.

1.1 Problemi black box con variabili intere

Molti problemi riscontrabili nel mondo reale (per esempio in campo economico, ingegneristico, biologico,...) vengono modellati ricorrendo a funzioni di tipo *black box*. Le funzioni di tipo *black box* sono spesso il risultato di una procedura numerica o vengono ottenute mediante misure sperimentali, per questo motivo queste funzioni non sono note analiticamente e di conseguenza le loro derivate parziali non sono disponibili. Da un punto di vista concettuale un sistema *black box*, il cui funzionamento può essere riassunto nell'immagine Fig.1.1, riceve in ingresso il vettore delle variabili $x \in \mathbb{R}^n$ e restituisce in uscita il valore della funzione $f(x)$ da minimizzare.

Figura 1.1: Funzione "black box"

La mancanza di conoscenza delle derivate rende questi problemi impossibili da risolvere con i metodi "tradizionali" (ovvero con algoritmi basati sulle derivate) e richiede lo sviluppo di metodi alternativi. Questo però non è l'unico ostacolo che solitamente si incontra: spesso infatti bisogna considerare anche la possibile presenza di rumore o discontinuità nella funzione che si vuole ottimizzare, che aggiunge complessità al problema. Un altro aspetto che rende il problema difficile da risolvere è il fatto che la valutazione di una funzione di tipo black box in un dato punto è spesso un'operazione onerosa in termini di risorse computazionali e per questo motivo solo un numero finito di valutazioni di funzione sono permesse all'ottimizzatore. Il problema diventa ulteriormente complesso se a tutto ciò viene aggiunto il fatto che molti sistemi reali sono spesso descritti per mezzo di variabili intere non-rilassabili (per esempio: numero di infermieri in un reparto, numero di bobine in un magnete) che necessitano di essere propriamente trattate. Infatti quando si ha a che fare con problemi di quest'ultimo tipo, non esiste un modo semplice

per ottenere limiti inferiori per la valutazione della qualità delle soluzioni generate. Pertanto, non è possibile né eliminare parte dell'insieme ammissibile né ottenere un gap di ottimalità per le soluzioni (come gli algoritmi esatti nella programmazione intera di solito fanno).
In genere quando i metodi esatti non possono essere applicati entrano in gioco i metodi euristici. I *metodi di ricerca esplorativi* (*explorative search methods*), che cercano una nuova soluzione in alcuni intorni di un dato punto e perturbano il punto e/o l'intorno quando vengono soddisfatte specifiche condizioni, e i *metodi evolutivi basati su popolazioni di punti* (*population based methods*), che a ogni iterazione usano un insieme di punti per generare nuove soluzioni, rappresentano approcci euristici classici per problemi con variabili intere. Ciononostante, queste due classi di approcci non sono adatte ai problemi di ottimizzazione appena descritti in quanto sono richieste un numero elevato di valutazioni di funzione per trovare delle buone soluzioni. Sorge dunque l'esigenza di ricercare ulteriori metodi più adatti alla risoluzione di problemi con questo grado di difficoltà.

Lo scopo che si vuole perseguire è quello di creare un metodo di ottimizzazione in grado di esplorare il lattice degli interi nel miglior modo possibile sfruttando le informazioni raccolte fino a quel momento, ovvero un metodo capace di ridurre il più possibile il valore della funzione obiettivo senza sprecare il numero di valutazioni di funzione che gli sono permesse.
Il problema che viene affrontato è dunque il seguente

$$\begin{aligned} &min\ f(x) \\ &x \in C \cap \mathbb{Z}^n \end{aligned} \tag{1.2}$$

dove $f : \mathbb{R}^n \to \mathbb{R}$ è una funzione di tipo black box, $C \subset \mathbb{R}^n$, $C \neq \emptyset$ è un insieme compatto non vuoto, la cui descrizione può include l'uso di vincoli di tipo black box, e $C \cap \mathbb{Z}^n$ è non vuoto. Assumiamo inoltre che le funzioni black box siano computazionalmente onerose.
I metodi per affrontare questo tipo di problemi si dividono in due classi:

- **Direct search methods** [3]: ad ogni iterazione la funzione obiettivo viene campionata su uno specifico insieme di punti e il miglior punto (in termini di valore di funzione obiettivo) viene scelto come nuova iterata;

- **Model based methods** [7]: a ogni iterazione viene costruito un modello della funzione obiettivo e un nuovo punto che minimizza il modello viene eventualmente selezionato;

Osservazione. I metodi di ricerca diretta possono essere ulteriormente suddivisi a seconda di come la funzione obiettivo viene campionata sul lattice: esistono metodi che utilizzano un insieme predeterminato di direzioni di ricerca con un passo comune fisso (*stencil*), e metodi che utilizzano un insieme predeterminato di direzioni di ricerca con passi che variano dinamicamente (*skewed stencil*).

Nei Capitoli successivi verranno presentati vari algoritmi volti alla risoluzione di problemi di tipo (1.2).

Capitolo 2

Principali algoritmi per problemi black box con variabili intere

2.1 Definizioni

Raccogliamo in questa sezione tutte le definizioni utili per la comprensione dei paragrafi e dei capitoli successivi.

Definizione 2.1. (Stencil) Dato un punto $x \in \mathbb{R}^n$, un passo $\alpha \in \mathbb{R}_+$ e p direzioni $d_i \in \mathbb{R}^n$, $i = 1, ..., p$, si definisce *stencil* l'insieme di punti:

$$S(x, \alpha, d_1, ..., d_p) = \{x \pm \alpha d_i, i = 1, ..., p\}. \tag{2.1}$$

Un particolare stencil è, per esempio, lo stencil dei vettori coordinati definito nel seguente modo:

$$S(x, \alpha, e_1, ..., e_p) = \{x \pm \alpha e_i, i = 1, ..., p\}. \tag{2.2}$$

dove $e_i \in \mathbb{R}^n$ è l'i-esimo vettore coordinato. Successivamente verrà introdotto lo *skewed stencil* che si ottiene dalla Def 2.1 ammettendo passi diversi per ogni direzione.

Definizione 2.2. (Divisore) Dati due interi a e b, a divide b, o a è un divisore (o fattore) di b, o b è un multiplo di a, se esiste un intero c tale che $b = ca$ e si denota con $a|b$.

Definizione 2.3. (Skewed stencil) Dato un punto $x \in \mathbb{R}^n$, p passi $\alpha_i \in \mathbb{R}_+$ e p direzioni $d_i \in \mathbb{R}^n$, $i = 1, ..., p$, si definisce *skewed stencil* l'insieme di punti:

$$S(x, \alpha_1, ..., \alpha_p, d_1, ..., d_p) = \{x \pm \alpha_i d_i, i = 1, ..., p\}. \tag{2.3}$$

Sia $v \in \mathbb{Z}^n$. Chiamiamo $d \in \mathbb{Z}$ divisore comune di $v_1, ..., v_n$ se $d|v_i$, con $i = 1, ..., p$. Il *massimo comun divisore* di $v_1, ..., v_n$, denotato con $GCD(v_1, ..., v_n)$ è un divisore comune non negativo tale che tutti gli altri divisori comuni di $v_1, ..., v_n$ dividono d.

Definizione 2.4. (Vettore primitivo) Un vettore $v \in \mathbb{Z}^n$ è detto *primitivo* se $GCD(v_1, ..., v_n) = 1$.

Osservazione 1. Dati due punti $x, y \in \mathbb{Z}^n$, si ha che $x - y = \alpha d$, con $d \in \mathbb{Z}^n$ un vettore primitivo e $\alpha \in \mathbb{N}$. Quindi, partendo da un punto $x \in \mathbb{Z}^n$, ogni altro punto $y \in \mathbb{Z}^n$ può essere raggiunto scegliendo un opportuno passo $\alpha \in \mathbb{N}$ lungo una specifica direzione primitiva $d \in \mathbb{Z}^n$.

Introduciamo ora il concetto di direzione primitiva ammissibile, di fondamentale importanza per la struttura del nostro algoritmo.

Definizione 2.5. (Direzione primitiva ammissibile) Dato un punto $\bar{x} \in C \cap \mathbb{Z}^n$, una direzione primitiva d è *ammissibile* in $\bar{x}$ per C se esiste $\beta \in \mathbb{N}$ tale che

$$\bar{x} + \alpha d \in C \cap \mathbb{Z}^n, \text{ per ogni } \alpha \leq \beta, \alpha \in \mathbb{N}. \tag{2.4}$$

Denotiamo con $D(\bar{x})$ l'insieme di tutte le direzioni primitive ammissibili nel punto $\bar{x}$. Ricordando l' *Osservazione 1*, è facile capire che partendo da $\bar{x}$ tramite una scelta adeguata di direzioni nell'insieme $D(\bar{x})$ e un adeguato spostamento lungo le direzioni scelte, possono essere raggiunti tutti i punti ammissibili per il problema (1.2).

Definizione 2.6. (Intorno discreto) Dato un punto $\bar{x} \in C \cap \mathbb{Z}^n$ e un parametro $\beta \in \mathbb{N}$, l' *intorno discreto* di $\bar{x}$ è dato da

$$N(\bar{x}, \beta) = \{x \in C \cap \mathbb{Z}^n : x = \bar{x} + \alpha d, \text{ con } \alpha \leq \beta, \alpha \in \mathbb{N} \text{ e } d \in D(\bar{x})\}. \tag{2.5}$$

Osservazione 2. Si noti che l'intorno discreto $N(\bar{x}, \beta)$ può coincidere con l'intero insieme ammissibile del problema (1.2), prendendo un β sufficientemente grande.

Un esempio di intorno discreto è dato dalla Fig. 2.1. Ovviamente il concetto di intorno discreto è puramente ideale. Infatti costruire un tale intorno

risulterebbe molto dispensioso e nella pratica non sarebbe efficiente. Per questo motivo abbiamo bisogno di sostituire $D(\bar{x})$ nella definizione precedente con un adeguato sottoinsieme di direzioni, ottenendo la seguente definizione.

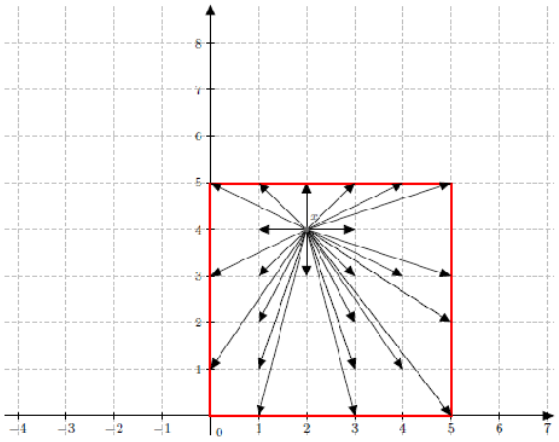

Figura 2.1: Esempio di intorno discreto con $\beta = 1$, $X = \{x \in \mathbb{R}^2 : 0 \leqslant x_1 \leqslant 5, 0 \leqslant x_2 \leqslant 5\}$

Definizione 2.7. (Intorno discreto debole) Dato un punto $\bar{x} \in C \cap \mathbb{Z}^n$, un parametro $\beta \in \mathbb{N}$ e un sottoinsieme $D \subset D(\bar{x})$, l' *intorno discreto debole* di $\bar{x}$ è dato da

$$N_w(\bar{x}, \beta, D) = \{x \in C \cap \mathbb{Z}^n : x = \bar{x} + \alpha d, \text{ con } \alpha \leq \beta, \alpha \in \mathbb{N} \text{ e } d \in D\}. \quad (2.6)$$

Infine diamo la definizione di minimo locale per il problema (1.2).

Definizione 2.8. (Punto di minimo locale) Un punto $x^* \in C \cap \mathbb{Z}^n$ è un *punto di minimo locale* per il problema (1.2), se esiste $\beta \in \mathbb{N}$ tale che

$$f(x^*) \leq f(x), \forall x \in N(x^*, \beta). \quad (2.7)$$

Allo stesso modo otteniamo la definizione di punto di minimo locale debole, più utile dal punto di vista pratico.

Definizione 2.9. (Punto di minimo locale debole) Un punto $x^* \in C \cap \mathbb{Z}^n$ è un *punto di minimo locale debole* per il problema (1.2), se esisteno $\beta \in \mathbb{N}$ e un sottoinsieme $D \subset D(x^*)$ tali che

$$f(x^*) \leq f(x), \forall x \in N_w(x^*, \beta, D). \quad (2.8)$$

2.2 Algoritmi nella letteratura

Come anticipato nel *Capitolo 1*, consideriamo un problema black box con variabili intere del tipo (1.2).

Quali algoritmi possono essere usati per risolvere problemi di questo tipo? Nella letteratura se ne possono trovare diversi, tutti di recente comparsa. Di seguito ne presenteremo alcuni, mettendo in evidenza gli aspetti più importanti nella loro implementazione. Successivamente dedicheremo il *Capitolo 3* alla descrizione più dettagliata di uno di questi algoritmi e il *Capitolo 4* a una modifica di quest'ultimo.

2.2.1 NOMAD

L'algoritmo NOMAD (Nonlinear Optimization by Mesh Adaptive Direct search) [12] nasce come algoritmo utile per la risoluzione di problemi black box che presentano un numero contenuto di variabili. L'algoritmo NOMAD è un'implementazione dell'algoritmo MADS (Mesh Adaptive Direct Search), nel cuore dell'algoritmo NOMAD risiede dunque la struttura dell'algoritmo MADS. Come il nome suggerisce, questo metodo genera iterate su una serie di mesh con dimensione variabile. Una mesh è una discretizzazione dello spazio delle variabili.
Il compito di ogni iterazione dell'algoritmo MADS è generare un punto di prova (*trial point*) sulla mesh che risulti migliore della soluzione corrente. Quando un'iterazione fallisce nel fare ciò, l'iterazione successiva viene svolta su una mesh più fine.
Ogni iterazione è composta da due step principali, chiamati SEARCH STEP e POLL STEP. Il SEARCH STEP (passo di ricerca) può restituire qualsiasi punto della mesh su cui sta lavorando, ma il suo scopo è quello di trovare un punto che migliori la soluzione corrente.
Il POLL STEP (passo di esplorazione) genera sulla mesh dei punti di prova, nelle vicinanze della miglior soluzione corrente. Siccome il POLL STEP è alla base dell'analisi della convergenza, è la parte dell'algoritmo su cui si è concentrata maggiormente la ricerca.
Di seguito presentiamo lo pseudocodice dell'algoritmo MADS.

Algorithm 1 MADS

Inizializzazione. Sia $x_0 \in \mathbb{R}^n$ un punto iniziale e poni $k \leftarrow 0$
Finché il Criterio d'Arresto è soddisfatto
  Cerca (SEARCH) nella mesh una soluzione migliore di x_k
  If la ricerca fallisce **then**
    Esplora (POLL) la mesh per cercare una soluzione migliore di x_k
  Endif
  If una soluzione migliore viene trovata con SEARCH o POLL **then**
    chiama la soluzione x_{k+1} e rendi meno fine la mesh
  Else
    Imposta $x_{k+1} = x_k$ e rendi più fine la mesh
  Endif
  Aggiorna i parametri e poni $k \leftarrow k + 1$

NOMAD include inoltre altri algoritmi, come per esempio:

- Un algoritmo MIXED VARIABLE PROGRAMMING (MVP) per l'ottimizzazione di varibili continua, dicrete e categoriali;
- Un algoritmo VARIABLE NEIGHBORHOOD SEARCH (VNS) per uscire dai minimi locali;
- Algoritmi per l'esecuzione parallela.

Per una trattazione più approfondita si rimanda a [11] e [12].

2.2.2 MISO

Un algoritmo che ha raggiunto risultati migliori rispetto all'algoritmo NOMAD è l'algoritmo MISO (Mixed-Integer Surrogate Optimization algorithm) [15]. Nell'algoritmo MISO la funzione obiettivo viene approssimata attraverso l'uso di modelli surrogati. Tali modelli decidono inoltre in quali punti della regione ammissibile la funzione obiettivo deve essere calcolata. Tra i vari modelli surrogati che possono essere usati con l'algoritmo MISO troviamo l'RBFs (Radial basis functions). Una funzione RBF interpolante è definita nel seguente modo:

$$s(z) = \sum_{i=1}^{n} \lambda_i \phi(||z - z_i||) + p(z) \tag{2.9}$$

dove $\phi(\cdot)$ è una funzione radial basis (per esempio $\phi(r) = r^3$), z_i, $i = 1, ..., n$ indica i punti in cui il valore della funzione obiettivo è noto e $p(\cdot)$ denota la tail polinomiale il cui ordine dipende dalla scelta effettuata per l'RBF (per RBF cubici è necessaria una tail polinomiale che sia almeno lineare $p(z) = a + b^T z$). I parametri $\lambda_i \in \mathbb{R}_+$, $i = 1, ..., n$, $a \in \mathbb{R}$ e $b = [b_1, ..., b_d]^T \in \mathbb{R}^d$ sono determinati attraverso il sistema lineare:

$$\begin{bmatrix} \Phi & \mathbf{P} \\ \mathbf{P}^T & \mathbf{0} \end{bmatrix} \begin{bmatrix} \lambda \\ \mathbf{c} \end{bmatrix} = \begin{bmatrix} \mathbf{F} \\ \mathbf{0} \end{bmatrix} \tag{2.10}$$

dove $\Phi_{i,j} = \phi(||z_i - z_j||)$, $i, j = 1, ..., n$, $\mathbf{0}$ è una matrice le cui entrate sono tutte nulle e

$$\mathbf{P} = \begin{bmatrix} z_1^T 1 \\ z_2^T 1 \\ \vdots \\ z_n^T 1 \end{bmatrix}, \lambda = \begin{bmatrix} \lambda_1 \\ \lambda_2 \\ \vdots \\ \lambda_n \end{bmatrix}, \mathbf{c} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_d \\ a \end{bmatrix}, \mathbf{F} = \begin{bmatrix} f(z_1) \\ f(z_2) \\ \vdots \\ f(z_n) \end{bmatrix} \tag{2.11}$$

la matrice in (2.10) è invertibile se e solo se $\text{rank}(\mathbf{P}) = d + 1$.

Come riportato in [15], un generico algoritmo di ottimizzazione basato su un modello surrogato segue in generale i seguenti passi:

Algorithm 2 General Surrogate Model Algorithm

1: Crea uno schema iniziale sperimentale e calcola la funzione obiettivo nei punti selezionati.
2: Adatta il modello surrogato scelto ai dati dello Step 1.
3: Usa le informazioni derivanti dal modello surrogato per selezionare il nuovo punto z_{new} in cui calcolare la funzione obiettivo.
4: Calcola la funzione obiettivo in z_{new}: $f_{new} = f(z_{new})$.
5: **If** il Criterio d'Arresto non è soddisfatto **then**
6: Aggiorna il modello surrogato e vai allo Step 3.
7: **else**
8: Ritorna la migliore soluzione trovata durante l'ottimizzazione.
9: **Endif**

Nello Step 1, uno schema iniziale viene creato e la funzione obiettivo viene valutata nei punti selezionati. In generale può essere usata qualsiasi strategia iniziale, ma è necessario essere sicuri che ci siano abbastanza punti per poter adattare il modello surrogato nello Step 2. Le informazioni finora

raccolte attraverso il calcolo della funzione obiettivo nei vari punti scelti e l'adattamento del modello surrogato sono usati nello Step 3 per trovare un nuovo punto. Dopo aver valutato la funzione obiettivo in tale punto nello Step 4, se il criterio d'arresto non è soddisfatto, il modello surrogato viene aggiornato nello Step 6 e viene selezionato un nuovo punto per la valutazione. Altrimenti l'algoritmo si arresta e ritorna la soluzione migliore trovata durante l'ottimizzazione.

Lo schema generale riportato nell'algoritmo 2 rappresenta la base per l'algoritmo MISO. Le uniche differenze sono negli Step 1 e 3. Di seguito riportiamo dunque lo pseudocodice dell'algoritmo MISO.

Algorithm 3 MISO

1: Crea uno schema iniziale sperimentale. Verifica che le variabili intere dei punti presenti nello schema assumano valori interi. Calcola la funzione obiettivo nei punti selezionati.
2: Adatta il modello surrogato scelto ai dati dello Step 1.
3: Usa le informazioni derivanti dal modello surrogato per selezionare il nuovo punto in cui calcolare la funzione obiettivo. Verifica che il nuovo punto z_{new} soddisfi i vincoli interi.
4: Calcola la funzione obiettivo in z_{new}: $f_{new} = f(z_{new})$.
5: **If** il Criterio d'Arresto non è soddisfatto **then**
6: Aggiorna il modello surrogato e vai allo Step 3.
7: **else**
8: Ritorna la migliore soluzione trovata durante l'ottimizzazione.
9: **Endif**

Nello Step 1 dell'algoritmo 3, nel generare lo schema iniziale vengono creati solo punti che soddisfano i vincoli interi. La funzione obiettivo viene poi valutata in tali punti e il modello surrogato viene adattato di conseguenza nello Step 2. Nel momento in cui il modello surrogato viene adattato, si assume che tutte le variabili siano continue in modo da ottenere una superficie liscia. Di conseguenza, nello Step 3, dobbiamo garantire nuovamente che il nuovo punto soddisfi i vincoli interi.

Varianti dell'algoritmo MISO sono consultabili in [15].

2.2.3 SO-MI

Un altro algoritmo basato su modelli surrogati è l'algoritmo SO-MI (Surrogate Optimization - Mixed-Integer) [18]. Come l'algoritmo MISO, anche

l'algoritmo SO-MI utilizza funzioni radial basis RBF.

Di seguito presentiamo molto brevemente lo schema dell'algoritmo SO-MI. Una trattazione più dettagliata si può trovare in [18].
Indichiamo con f la funzione obiettivo e con c_j le funzioni che descrivono i vincoli della regione ammissibile.

Algorithm 4 SO-MI

1: Genera ripetutamente uno schema iniziale di $2k+1$ punti attraverso il campionamento con ipercubo latino, arrotonda le variabili discrete all'intero più vicino e aggiungi un punto ammissibile allo schema finché la matrice $\mathbf{P}$ in (2.11) non avrà rango $k+1$. Denota tali punti con $w_1, ..., w_{n_0}$, dove $n_0 = 2(k+1)$.
2: Calcola il valore della funzione obiettivo e delle varie funzioni che descrivono la regione ammissibile nei punti w_i, con $i = 1, ..., n_0$.
3: Trova il miglior punto ammissibile w_{min} con il valore più basso di funzione obiettivo e determina il peggior valore ammissibile di funzione obiettivo $f_{max} = max\{f(w_i)$ dove $c_j(w_i) \leq 0, \forall j = 1, ..., m\}$.
4: Calcola $f_p(w_i)$, $i = 1, ..., n_0$ come riportato in (2.13).
5: Usa le coppie $(w_i, f_p(w_i))$, $i = 1, ..., n_0$ per calcolare i parametri del modello RBF risolvendo il sistema (2.10).
6: Itera le operazioni seguenti fintantoché non è raggiunto il massimo numero di calcoli di funzione possibili:
   a. Crea quattro gruppi di punti candidati attuando le seguenti operazioni randomicamente (i) perturba solo i valori delle variabili continue di w_{min}, (ii) perturba solo i valori delle variabili discrete di w_{min}, (iii) perturba i valori delle variabili discrete e continue di w_{min}, (iv) seleziona punti dal dominio in modo uniforme. Controlla che i vincoli interi siano soddisfatti. Elimina tutti i punti candidati che sono già stati campionati.
   b. Calcola i criteri di scoring per ogni punto candidato in ogni gruppo [18].
   c. Scegli da ogni gruppo il punto candidato con il miglior punteggio.
   d. Calcola (in parallelo) la funzione obiettivo in questi punti.
   e. Se necessario, aggiorna il miglior punto ammissibile trovato finora w_{min}. Se necessario, aggiorna anche il peggior valore ammissibile di funzione obiettivo f_{max} e modifica il valore di funzione obiettivo in accordo con (2.13) [18].
   f. Aggiorna i parametri del modello RBF usando i valori di funzione obiettivo dello Step 6(e).
7: Ritorna la migliore soluzione ammissibile w_{min} trovata.

Per la costurizione del SO-MI viene considerata una funzione che tiene

conto della violazione dei vincoli:

$$v(w) = \sum_{j=1}^{m} [max\{0, c_j(w)\}]^2 \tag{2.12}$$

Tale funzione viene poi utilizzata per la creazione della funzione di penalità:

$$f_p(w) = \begin{cases} f_{max} + pv(w) & \text{se } w \text{ è ammissibile} \\ f(w) & \text{altrimenti} \end{cases} \tag{2.13}$$

con p parametro di penalità e f_{max} peggior valore di funzione obiettivo finora calcolato.

2.2.4 BFO

L'algoritmo BFO [13] è un algoritmo senza derivate in grado di gestire problemi a variabili discrete e continue. Come la maggior parte degli algoritmi di questo tipo, l'algoritmo BFO lavora meglio con problemi di piccole dimensioni. La struttura del BFO è quella di un patter search algorithm: a ogni iterazione la funzione oggetto viene valutata su un numero finito di punti di una mesh, nelle vicinanze del punto corrente, nella speranza di trovare un nuovo punto con un valore di funzione obiettivo minore, che diventerà la nuova iterata.
Più precisamente ogni iterazione dell'algoritmo inizia con l'iterata corrente $\bar{x}$, il valore di funzione obiettivo corrente $\bar{f} = f(\bar{x})$ e un insieme numerabile D di direzioni di ricerca, sia per le variabili continue che per quelle discrete. Queste direzioni definiscono implicitamente la mesh corrente come l'insieme di tutti i punti in $C = C^c \cup C^d$ (con C^c dominio delle variabili continue, di dimensione n_c, e C^d dominio delle variabili discrete, di dimensione n_d) che possono essere raggiunti da $\bar{x}$ spostandosi lungo una direzione presente di D con passo fissato. Alla prima iterazione del BFO vengono poste una soluzione iniziale $\bar{x}$ e un vettore di passi $\delta = \delta_0 \in \mathbb{R}^{n_c} \times \mathbb{Z}^{n_d}$. L'algoritmo poi procede esplorando il lattice specificato da questi elementi fino a quando l'esplorazione risulta esaurita. Il passo continuo δ_C subisce dunque una variazione, la ricerca di direzioni per variabili continue viene aggiornata e il processo viene ripetuto. L'esplorazione di una data mesh avviene in due fasi. La prima fase è il POLL STEP (passo di esplorazione), dove viene eseguita una ricerca calcolando passi lungo tutte le direzioni partendo dall'iterata corrente x_k. Se

questa fase non ha successo e $n_d \neq 0$, inizia la seconda fase. In questa fase viene eseguita un'ulteriore ricerca attraverso l'esplorazione dei sottoproblemi definiti fissando ogni variabile discreta a un valore nell'intorno del punto $\bar{x}$. Questa fase, chiamata RECURSIVE STEP (passo ricorsivo), viene eseguita richiamando ricorsivamente l'algoritmo stesso per ogni sottoproblema. Queste due fasi sono poi seguite dalla terza fase, chiamata TERMINATION STEP (passo di terminazione) dove, se viene trovato un punto che riduce il valore di funzione obiettivo in entrambe le fasi, il punto trovato diventa la nuova iterata, in caso contrario l'iterata resta quella precedente.

Lo pseudocodice dell'algoritmo BFO viene presentato di seguito. Per una trattazione più approfondita si rimanda a [14].

Algorithm 5 BFO

1: **Inizializzazione.** Sia $\bar{x} \in C^c \cup C^d$ e $\bar{f} = f(\bar{x})$, passo iniziale $\delta = \delta_0 \in \mathbb{R}^{n_c} \times \mathbb{Z}^{n_d}$ e l'insieme delle direzioni iniziali D. Imposta il valore iniziale $x^{best} = \bar{x}$, $f^{best} = \bar{f}$.
2: **Finché la convergenza è soddisfatta**
3: a. POLL STEP. Esegui un ciclo di ricerca sulle variabili, spostandoti avanti e indietro lungo le direzioni di D con passo δ. Se viene trovato un poll point x^p costruito in questo modo e tale che $f(x^p) < f^{best}$, poni $x^{best} = x^p$ e $f^{best} = f(x^p)$. Se $f^{best} < \bar{f}$ o $n_d = 0$, vai a TERMINATION STEP.
4: b. RECURSIVE STEP. Se richiesto, applica l'algoritmo BFO per risolvere i sottoproblemi definiti fissando ogni variabile discreta a un valore che differisce da $\bar{x}$ per $\pm$ il passo. Se viene trovato un punto x^r costruito in questo modo e tale che $f(x^r) < f^{best}$, poni $x^{best} = x^r$ e $f^{best} = f(x^r)$.
5: c. TERMINATION STEP. Se $f^{best} < \bar{f}$ (l'iterazione ha successo), aggiorna $\bar{x} = x^{best}$ e $\bar{f} = f^{best}$, aumenta δ_C, aggiorna l'insieme D per le variabili continue e vai a POLL STEP. Altrimenti (l'iterazione non ha successo), controlla la convergenza. Se la convergenza non è soddisfatta, diminuisci δ_C, scegli un nuovo insieme D per le variabili continue e vai a POLL STEP.

2.2.5 DFL

L'algoritmo DFL (Derivative-Free Linesearch) [2] è un algoritmo per problemi misti interi. Gli ingredienti fondamentali di questo metodo sono una procedura di ricerca continua *Continuous search* e una procedura di ricerca discreta *Discrete search*, utilizzate per esplorare le direzioni coordinate associate alle variabili continue e discrete, rispettivamente. L'iterata corrente viene aggiornata non appena viene raggiunta una sufficiente riduzione della funzione obiettivo, da parte di una delle due procedure. La procedura *Continuous search* è una procedura abbastanza standard per il contesto in cui siamo e per una sua trattazione si rimanda a [16]. La procedura di ricerca discreta *Discrete search*, invece, è simile alla procedura di ricerca continua ma la sufficiente riduzione viene controllata da un parametro ξ, il quale subisce una riduzione durante il processo di ottimizzazione. Nello specifico, il parametro di controllo ξ viene ridotto ogniqualvolta nessuna delle variabili discrete è stata aggiornata dalla procedura *Discrete search* e il passo $\tilde{\alpha}^i$ risulta uguale a 1 per le variabili discrete. Il passo $\tilde{\alpha}^i$ viene, in generale, usato per campionare la funzione obiettivo lungo le direzioni d^i, con $i = 1, ..., n$. Ogni volta che il passo deve essere ridotto, viene adottato un fattore costante $\theta \in (0, 1)$. Il punto x_0 indica il punto iniziale per l'algoritmo.
Lo pseudocodice dell'algoritmo DFL [2] viene presentato qui sotto, insieme allo pseudocodice per la procedura continua *Continuous search* e allo pseudocodice per la procedura discreta *Discrete search*.
Assumiamo: $x \in \mathbb{R}^n$, $l, u \in \mathbb{R}^n$, $l_i < u_i$ per ogni $i = 1, ..., n$, $I_z \subset \{1, ..., n\}$ sottoinsieme di indici per le variabili discrete, $I_c = \{1, ..., n\} \setminus I_z$ sottoinsieme di indici per le variabili continue, $X = \{x \in \mathbb{R}^n : l \leq x \leq u\}$ insieme compatto, $Z = \{x \in \mathbb{R}^n : x_i \in \mathbb{Z}, i \in I_z\}$.

Ad ogni iterazione k l'algoritmo DFL, considera l'iterata corrente x_k, esplora tutte le direzioni coordinate e produce i punti y_k^i, $i = 1, ..., n$. Quando $i \in I_c$, ovvero nel caso delle variabili continue, viene calcolato il passo α_k^i mentre il passo $\tilde{\alpha}_k^i$ viene aggiornato come descritto in [16]. Quando $i \in I_z$, l'algoritmo attua una procedura discreta *Discrete search* che è molto simile alla procedura continua *Continuous search*, eccetto per il fatto che la sufficiente riduzione viene governata dal parametro ξ_k. La formula di aggiornamento del passo $\tilde{\alpha}_k^i$ è tale che $1 \leq \tilde{\alpha}_k^i \in \mathbb{Z}$.

Algorithm 6 DFL

Dati. $\theta \in (0,1)$, $\xi > 0$, $x_0 \in X \cap Z$, $\tilde{\alpha}_0^i > 0$, $i \in I_c$, $\tilde{\alpha}_0^i = 1$, $i \in I_z$ e imposta $d_0^i = e^i$ per ogni $i = 1, ..., n$.
For $k = 0, 1, ...$
  Imposta $y_k^1 = x_k$.
  For $i = 1, ..., n$
    If $i \in I_c$ **then** calcola α attraverso la *Continuous search*$(\tilde{\alpha}_k^i, y_k^i, d_k^i; \alpha)$
      If $\alpha = 0$ **then** imposta $\alpha_k^i = 0$ e $\tilde{\alpha}_{k+1}^i = \theta \tilde{\alpha}_k^i$.
      else imposta $\alpha_k^i = \alpha$, $\tilde{\alpha}_{k+1}^i = \alpha$.
      Endif
    else calcola α attraverso la *Discrete search*$(\tilde{\alpha}_k^i, y_k^i, d_k^i, \xi_k; \alpha)$
      If $\alpha = 0$ **then** imposta $\alpha_k^i = 0$ e $\tilde{\alpha}_{k+1}^i = max\{1, \lfloor \tilde{\alpha}_k^i / 2 \rfloor\}$.
      else imposta $\alpha_k^i = \alpha$, $\tilde{\alpha}_{k+1}^i = \alpha$.
      Endif
    Imposta $y_k^{i+1} = y_k^i + \alpha_k^i d_k^i$ e $d_{k+1}^i = d_k^i$.
    Endif
  Endfor
  If $(y_k^{n+1})_z = (x_k)_z$ e $\tilde{\alpha}_k^i = 1$, $i \in I_z$, **then** imposta $\xi_{k+1} = \theta \xi_k$
  else Imposta $\xi_{k+1} = \xi_k$.
  Endif
  Trova $x_{k+1} \in X \cap Z$ tale che $f(x_{k+1}) \leq f(y_k^{n+1})$.
Endfor

Algorithm 7 Continuous search$(\tilde{\alpha}, y, d; \alpha)$

Dati. $\gamma > 0$, $\delta \in (0,1)$.
Step 1. Calcola il più grande $\bar{\alpha}$ tale che $y + \bar{\alpha} d \in X \cap Z$. Imposta $\alpha = min\{\bar{\alpha}, \tilde{\alpha}\}$.
Step 2. **If** $\alpha > 0$ e $f(y + \alpha d) \leq f(y) - \gamma \alpha^2$ **then** vai a Step 6.
Step 3. Calcola il più grande $\bar{\alpha}$ tale che $y - \bar{\alpha} d \in X \cap Z$. Imposta $\alpha = min\{\bar{\alpha}, \tilde{\alpha}\}$.
Step 4. **If** $\alpha > 0$ e $f(y - \alpha d) \leq f(y) - \gamma \alpha^2$ **then** imposta $d \leftarrow -d$ e vai a Step 6.
Step 5. Imposta $\alpha = 0$. **Return**.
Step 6. **While** $(\alpha < \tilde{\alpha}$ e $f(y + \frac{\alpha}{\delta} d) \leq f(y) - \gamma \frac{\alpha^2}{\delta^2})$
  $\alpha \leftarrow \frac{\alpha}{\delta}$.
Step 7. Imposta $\alpha \leftarrow min\{\tilde{\alpha}, \alpha\}$. **Return**.

Algorithm 8 Discrete search$(\tilde{\alpha}, y, d, \xi; \alpha)$

Step 1. Calcola il più grande $\bar{\alpha}$ tale che $y + \bar{\alpha} d \in X \cap Z$. Imposta $\alpha = min\{\bar{\alpha}, \tilde{\alpha}\}$.
Step 2. **If** $\alpha > 0$ e $f(y + \alpha d) \leq f(y) - \xi$ **then** vai a Step 6.
Step 3. Calcola il più grande $\bar{\alpha}$ tale che $y - \bar{\alpha} d \in X \cap Z$. Imposta $\alpha = min\{\bar{\alpha}, \tilde{\alpha}\}$.
Step 4. **If** $\alpha > 0$ e $f(y - \alpha d) \leq f(y) - \xi$ **then** imposta $d \leftarrow -d$ e vai a Step 6.
Step 5. Imposta $\alpha = 0$. **Return**.
Step 6. **While** $(\alpha < \tilde{\alpha}$ e $f(y + 2\alpha d) \leq f(y) - \xi)$
  $\alpha \leftarrow 2\alpha$.
Step 7. Imposta $\alpha \leftarrow min\{\tilde{\alpha}, \alpha\}$. **Return**.

2.2.6 DFL$_{ord}$

Come per il DFL, l'algoritmo DFL$_{ord}$ [2] è un algoritmo senza derivate per problemi misti interi. Tale algoritmo è composto da due fasi. Nella prima le variabili continue sono aggiornate per mezzo di una ricerca di linea distributiva che restituisce un punto $\tilde{y}$. Nella seconda fase, partendo dal punto $\tilde{y}$, vengono investigate le direzioni collegate alle variabili discrete e viene restituito il punto che raggiunge il miglior valore di funzione obiettivo. A differenza dell'algoritmo DFL, la procedura di ricerca discreta richiede qui esclusivamente una semplice riduzione della funzione obiettivo. L'algoritmo DFL$_{ord}$ presenta proprietà di convergenza leggermente più forti dell'algoritmo DFL. Infatti è possibile mostrare che ogni punto limite della sequenza di iterate generate dall'algoritmo è stazionario per il nostro problema. Ciononostante, queste proprietà di convergenza forti sono bilanciate da una ridotta flessibilità nell'esplorazione delle variabili discrete.
Di seguito presentiamo lo pseudocodice dell'algoritmo [2]. Valgono le assunzioni fatte per il DFL.

Algorithm 9 DFL$_{ord}$

Dati. $\theta \in (0,1)$, $x_0 \in X \cap Z$, $\tilde{\alpha}_0^i > 0$, $i \in I_c$, $\tilde{\alpha}_0^i = 1$, $i \in I_z$ e imposta $d_0^i = e^i$ per ogni $i = 1, ..., n$. $I_c = \{1, ..., r\}$, $I_z = \{r+1, ..., n\}$.
For k = 0,1,...
  Imposta $y_k^1 = x_k$.
  For $i = 1, ..., r$
    Calcola α attraverso la *Continuous search*$(\tilde{\alpha}_k^i, y_k^i, d_k^i; \alpha)$
      If $\alpha = 0$ **then** imposta $\alpha_k^i = 0$ e $\tilde{\alpha}_{k+1}^i = \theta \tilde{\alpha}_k^i$.
      else imposta $\alpha_k^i = \alpha$, $\tilde{\alpha}_{k+1}^i = \alpha$.
      Endif
    Imposta $y_k^{i+1} = y_k^i + \alpha_k^i d_k^i$ e $d_{k+1}^i = d_k^i$.
  Endfor
  For $i = r+1, ..., n$
    Calcola α attraverso la *Discrete search*$(\tilde{\alpha}_k^i, y_k^i, d_k^i, 0; \alpha)$
      If $\alpha = 0$ **then** imposta $\alpha_k^i = 0$ e $\tilde{\alpha}_{k+1}^i = max\{1, \lfloor \theta \tilde{\alpha}_k^i / 2 \rfloor\}$.
      else imposta $\alpha_k^i = \alpha$, $\tilde{\alpha}_{k+1}^i = \alpha$.
      Endif
    Imposta $y_k^{i+1} = y_k^{r+1} + \alpha_k^i d_k^i$ e $d_{k+1}^i = d_k^i$.
  Endfor
  Imposta $\tilde{x}_{k+1} = arg\min_{y \in \{y_k^{r+1}, ..., y_k^{n+1}\}} f(y)$.
  Trova $x_{k+1} \in X \cap Z$ tale che $f(x_{k+1}) \leq f(\tilde{x}_{k+1})$.
Endfor

2.2.7 SDFL

Un altro algoritmo, simile agli algoritmi DFL e DFL$_{ord}$ è l'algoritmo SDFL [2]. L'algoritmo SDFL è anch'esso un algoritmo senza derivate per problemi misti a variabili intere e continue. Tale algoritmo è caratterizzato da una procedura di ricerca locale *Local search* la quale dapprima esegue una ricerca di linea lungo le direzioni collegate alle variabili discrete. Successivamente, se viene trovato un punto che permette una sufficiente decrescita della funzione obiettivo, tale punto diventa la soluzione corrente. Altrimenti, se viene trovato un punto z promettente, ovvero un punto che non è di molto peggiore in termini di valore di funzione obiettivo rispetto alla soluzione corrente, viene eseguita una ricerca distribuita a partire dal punto z. L'algoritmo SDFL con la procedura locale *Local search*, genera alcune sequenze e in particolare le sequenze $\{x_k\}$, $\{\xi_k\}$, $\{y_k^i\}$, $\{d_k^i\}$, $\{\alpha_k^i\}$, $\{\tilde{\alpha}_k^i\}$, $\{z_k^i\}$, per $i = 1, ..., n$. Ricordiamo inoltre che la procedura di ricerca locale *Local search* può essere vista come una procedura di ricerca discreta *Discrete search* arricchita attraverso una ricerca di griglia. Più precisamente una ricerca di griglia viene usata per esplorare meglio un intorno di un punto z promettente rispetto al punto corrente y, ovvero un punto z tale che $f(y) - \xi \leq f(z) \leq f(y) + \nu$.
Di seguito presentiamo lo psuedocodice dell'algoritmo SDFL [2] e quello della ricerca locale *Local search*. Assumiamo, come per gli algoritmi precedenti: $x \in \mathbb{R}^n$, $l, u \in \mathbb{R}^n$, $l_i < u_i$ per ogni $i = 1, ..., n$, $I_z \subset \{1, ..., n\}$ sottoinsieme di indici per le variabili discrete, $I_c = \{1, ..., n\} \setminus I_z$ sottoinsieme di indici per le variabili continue, $X = \{x \in \mathbb{R}^n : l \leq x \leq u\}$ insieme compatto, $Z = \{x \in \mathbb{R}^n : x_i \in \mathbb{Z}, i \in I_z\}$.

Algorithm 10 SDFL

Dati. $\theta \in (0,1)$, $\xi_0 > 0$, $x_0 \in X \cap Z$, $\tilde{\alpha}_0^i > 0$, $i \in I_c$, $\tilde{\alpha}_0^i = 1$, $i \in I_z$ e imposta $d_0^i = e^i$ per ogni $i = 1, ..., n$.
For $k = 0, 1, ...$
  Imposta $y_k^1 = x_k$.
  For $i = 1, ..., n$
    If $i \in I_c$ **then** calcola α attraverso la *Continuous search*$(\tilde{\alpha}_k^i, y_k^i, d_k^i; \alpha)$
      If $\alpha = 0$ **then** imposta $\alpha_k^i = 0$ e $\tilde{\alpha}_{k+1}^i = \theta \tilde{\alpha}_k^i$.
      else imposta $\alpha_k^i = \alpha$, $\tilde{\alpha}_{k+1}^i = \alpha$.
      Imposta $d_{k+1}^i = d_k^i$.
    else calcola α attraverso la *Local search*$(\tilde{\alpha}_k^i, y_k^i, d_k^i, \xi_k; \alpha, \tilde{z})$
      If $\alpha = 0$ e $\tilde{z} \neq y_k^i$ **then** $\alpha_k^i = 0$, $\tilde{\alpha}_{k+1}^i = \tilde{\alpha}_k^i$, imposta $y_k^{n+1} = \tilde{z}$, $d_{k+1}^i = d_k^i$ e **Exitfor**
      If $\alpha = 0$ **then** calcola α attraverso la *Local search*$(\tilde{\alpha}_k^i, y_k^i, -d_k^i, \xi_k; \alpha, \tilde{z})$
        If $\alpha = 0$ and $\tilde{z} \neq y_k^i$ **then** imposta $\alpha_k^i = 0$, $\tilde{\alpha}_{k+1}^i = \tilde{\alpha}_k^i$, $y_k^{n+1} = \tilde{z}$, $d_{k+1}^i = -d_k^i$ e **Exitfor**
        If $\alpha = 0$ **then** imposta $\alpha_k^i = 0$, $\tilde{\alpha}_{k+1}^i = max\{1, \lfloor \tilde{\alpha}_k^i / 2 \rfloor\}$ e $d_{k+1}^i = d_k^i$.
        else imposta $\alpha_k^i = \alpha$, $\tilde{\alpha}_{k+1}^i = \alpha$ e $d_{k+1}^i = -d_k^i$.
      else imposta $\alpha_k^i = \alpha$, $\tilde{\alpha}_{k+1}^i = \alpha$ e $d_{k+1}^i = d_k^i$.
    Endif
    Imposta $y_k^{i+1} = y_k^i + \alpha_k^i d_k^i$.
  Endfor
  If $(y_k^{n+1})_z = (x_k)_z$ e $\tilde{\alpha}_k^i = 1$, $i \in I_z$, **then** imposta $\xi_{k+1} = \theta \xi_k$
  else imposta $\xi_{k+1} = \xi_k$.
  Endif
  Trova $x_{k+1} \in X \cap Z$ tale che $f(x_{k+1}) \leq f(y_k^{n+1})$.
Endfor

Algorithm 11 Local search$(\tilde{\alpha}, y, d, \xi; \alpha, \tilde{z})$

Dati. $\nu > 0$
Inizializzazione. Calcola il più grande $\bar{\alpha}$ tale che $y + \bar{\alpha}d \in X \cap Z$. Imposta $\alpha = min\{\bar{\alpha}, \tilde{\alpha}\}$ e $z = y + \alpha d$.
Step 0. **If** $\alpha = 0$ o $f(z) > f(y) + \nu$ **then** imposta $\tilde{z} = y$, $\alpha = 0$ e **return**.
Step 1. **If** $\alpha > 0$ e $f(z) \leq f(y) - \xi$ **then** vai allo Step 2.
else vai allo Step 4.
Step 2. **While** ($\alpha < \tilde{\alpha}$ e $f(y + 2\alpha d) \leq f(y) - \xi$)
$\alpha \leftarrow 2\alpha$
Step 3. Imposta $\alpha \leftarrow min\{\bar{\alpha}, \alpha\}$ e $\tilde{z} = y + \alpha d$ e **return**.
Step 4. *(Grid search)* Imposta $z = y + \alpha d$.
Imposta $w^1 = z$.
For $i = 1, ..., n$
Sia $q^i = e^i$.
If $i \in I_z$ calcola α attraverso la *Discrete search*$(\tilde{\alpha}^i, w^i, q^i, \xi; \alpha)$
If $\alpha \neq 0$ e $f(w^i + \alpha q^i) \leq f(y) - \xi$ **then** imposta $\tilde{z} = w^i + \alpha q^i$, $\alpha = 0$ e **return**.
If $i \in I_c$ calcola α attraverso la *Continuous search*$(\tilde{\alpha}^i, w^i, q^i; \alpha)$
If $\alpha \neq 0$ e $f(w^i + \alpha q^i) \leq f(y) - \xi$ **then** imposta $\tilde{z} = w^i + \alpha q^i$, $\alpha = 0$ e **return**.
Imposta $w^{i+1} = w^i + \alpha q^i$.
Endfor
Imposta $\tilde{z} = y$, $\alpha = 0$ e **return**.

2.2.8 DCS

L'algoritmo DCS [1] è un algoritmo senza derivate che combina l'uso di direzioni primitive ammissibili con una ricerca di linea di tipo monotona o non-monotona per esplorare il lattice degli interi.
Nel caso in cui l'algoritmo utilizzi una ricerca di linea monotona viene indicato con M-DCS, altrimenti se la ricerca di linea usata è quella non-monotona viene indicato con NM-DCS.

Presentiamo di seguito la versione non-monotona dell'algoritmo.
Nella prima parte dell'algoritmo vengono generati punti attorno l'iterata x_k, attraverso l'uso delle direzioni presenti in $D \subseteq D(x_k)$ e una ricerca di linea non-monotona. Se con questa generazione l'algoritmo trova un nuovo punto in grado di garantire un sufficiente spostamento lungo la direzione di ricerca in uso e una riduzione del valore di riferimento f^{ref}, la generazione si ferma. In caso contrario continua fino ad esaurire completamente l'insieme D. Nel caso in cui l'algoritmo trovi un punto che garantisce una riduzione rispetto

al valore di riferimento f^{ref}, vi è un aggiornamento dei valori (iterata, valore di riferimento,...).

Algorithm 12 NM-DCS

Data. $x_0 \in X \cap \mathbb{Z}^n, D = D_0 \subset D(x_0)$ un insieme di direzioni iniziali, $\tilde{\alpha}_0^{(d)} = 1$, per ogni $d \in D_0$. $W = \{f(x_0)\}, f^{ref} = \bar{f}_0 = f(x_0), x_{min} = x_0, M \geq 1, M \in \mathbb{N}$
for $k = 0, 1, ...$ **do**
  Imposta $y = x_k$
  while $D \neq \emptyset$ e $y = x_k$ **do**
    Scegli $d \in D$ imposta $D = D \setminus \{d\}$
    Calcola α usando *Nonmonotone Search*$(\tilde{\alpha}_k^{(d)}, x_k, d, f^{ref}; \alpha)$
    if $\alpha = 0$ **then** imposta $\tilde{\alpha}_{k+1}^{(d)} = \max\{1, \lfloor \tilde{\alpha}_k^{(d)}/2 \rfloor\}$
    else
      Imposta $y = x_k + \alpha d, \tilde{\alpha}_k^{(d)} = \alpha$ e $\tilde{\alpha}_{k+1}^{(d)} = \alpha$
      if $f(y) < f(x_{min})$ **then** $x_{min} = y$
      if $|W| = M$ **then** pop(W)
      push$(f(y), W), f^{ref} = \max_{f \in W}\{f\}$
    end if
  end while
  Imposta $D_{k+1} = D_k$ e $D = D_k$
  Imposta $x_{k+1} = y$, $\bar{f}_{k+1} = f^{ref}$
end for

2.2.9 FS

L'algoritmo FS [1] è un algoritmo con stencil fisso, di tipo non-monotono/monotono. Esso si ottiene a partire dall'algoritmo DCS inibendo l'espansione del passo attraverso l'uso di una procedura di ricerca non-monotona/monotona. Anch'esso è dunque presente nelle due varianti monotona, indicato con M-FS, e non monotona, indicato con NM-FS.

Capitolo 3

Algoritmo BBOA

L'algoritmo BBOA (Black Box Optimization Algorithm) [1] ricerca soluzioni per problemi di tipo (1.2) con $C = \{x \in \mathbb{R}^n : l_i \leqslant x_i \leqslant u_i, i = 1, ..., n\}$, $l_i, u_i \in \mathbb{Z}$ e $-\infty \leqslant l_i \leqslant u_i \leqslant +\infty$ per ogni $i = 1, ..., n$. Nella versione NM-BBOA, quest'algoritmo combina l'uso di direzioni primitive ammissibili con una ricerca di linea non monotona volta ad esplorare unicamente i punti del lattice degli interi.

Lo schema dettagliato dell'algoritmo è riportato sotto. Come si può facilmente notare, in ogni iterazione ci sono tre differenti fasi.
Nella prima fase vengono generati punti attorno all'iterata x_k. Più precisamente viene presa una direzione da uno specifico insieme $D \subseteq D(x_k)$ e viene esplorata questa direzione per mezzo di una ricerca di linea non-monotona al fine di trovare un nuovo punto in grado di garantire uno spostamento sufficientemente grande lungo la direzione di ricerca e una riduzione rispetto a uno specifico valore di riferimento f^{ref} (vedi Step 12). La prima fase terminerà una volta che un tale punto viene trovato o l'insieme D diventa vuoto (ovvero tutte le direzioni di D sono state esplorate, ma non è stata ottenuta alcuna riduzione rispetto al valore di riferimento). Si noti che l'iterata x_k, le direzioni $d \in D_k$ e il passo iniziale $\tilde{\alpha}_k^{(d)}$ possono essere visti come uno skewed stencil variabile dinamicamente opportunamente modificato a ogni iterazione dell'algoritmo grazie al risultato della ricerca di linea. Nel caso in cui venga trovato un punto che garantisce una riduzione rispetto al valore di riferimento, l'algoritmo aggiorna:

- il passo iniziale collegato all'ultima direzione esplorata (ovvero allarga

lo stencil in quella direzione);

- il valore di riferimento;
- la queue necessaria per costruire il valore di riferimento nelle iterazioni successive.

Una volta che la Fase 1 è terminata, si passa alla seconda fase dove l'insieme delle direzioni di ricerca D (insieme delle direzioni di ricerca usato nella Fase 1) e D_k (insieme delle direzioni di ricerca generato finora) vengono eventualmente aggiornati. Più precisamente, se nella prima fase l'algoritmo ha successo nella ricerca di un nuovo punto, D_k resta lo stesso e D viene posto uguale a D_k. Altrimenti, l'algoritmo controlla se la *Nonmonotone Search* fallisce lungo tutte le direzioni in D_k e il passo iniziale è uguale a 1 per tutte quelle direzioni (ovvero lo skewed stencil definito dalle direzioni $d \in D_k$ e i passi $\tilde{\alpha}_k^{(d)}$ si riduce alla sua dimensione minima nella Fase 1). Se questo non è il caso, D_k resta lo stesso e D include solo le direzioni che falliscono con passo maggiore di 1. Altrimenti, l'algoritmo cerca di arricchire D_k. Se D_k è uguale all'insieme $D(x_k)$ delle direzioni primitive ammissibili in x_k, allora l'algoritmo controlla se x_k è il punto con il miglior valore di funzione obiettivo. Se è così, si arresta, altrimenti si sposta sul miglior punto trovato finora (vedi Step 19), D e D_k vengono impostati uguali a $D(x_k)$. Se l'insieme delle direzioni di ricerca generate è più piccolo di $D(x_k)$, viene arricchito ($D_{k+1} \supset D_k$) e l'insieme D include solo le nuove direzioni generate per arricchire D_k. Per finire, nella Fase 3 le iterate sono aggiornate.

Osservazione. Come per gli algoritmi DCS e FS, anche per BBOA esiste una versione che utilizza una ricerca di tipo monotono al posto di quella non-monotona. In questo caso l'algoritmo viene indicato con M-BBOA.

Algorithm 13 NonMonotone Black Box Optimization Algorithm (NM-BBOA)

Data. $x_0 \in C \cap \mathbb{Z}^n, D = D_0 \subset D(x_0)$ un insieme di direzioni iniziali, $\tilde{\alpha}_0^{(d)} = 1$, per ogni $d \in D_0$. $W = \{f(x_0)\}, f^{ref} = \bar{f}_0 = f(x_0), x_{min} = x_0, M \geq 1, M \in \mathbb{N}$
for $k = 0, 1, ...$ **do**
  Imposta $y = x_k$
  while $D \neq \emptyset$ e $y = x_k$ **do**
    Scegli $d \in D$ imposta $D = D \setminus \{d\}$
    Calcola α usando *Nonmonotone Search*$(\tilde{\alpha}_k^{(d)}, x_k, d, f^{ref}; \alpha)$
    if $\alpha = 0$ **then** imposta $\tilde{\alpha}_{k+1}^{(d)} = \max\{1, \lfloor \tilde{\alpha}_k^{(d)}/2 \rfloor\}$
    else
      Imposta $y = x_k + \alpha d, \tilde{\alpha}_k^{(d)} = \alpha$ e $\tilde{\alpha}_{k+1}^{(d)} = \alpha$
      if $f(y) < f(x_{min})$ **then** $x_{min} = y$
      if $|W| = M$ **then** pop(W)
      push$(f(y), W), f^{ref} = \max_{f \in W}\{f\}$
    end if
  end while
  if $y = x_k$ **then**
    if *Nonmonotone Search* fallisce con $\tilde{\alpha}_k^{(d)} = 1$ per ogni $d \in D_k$ **then**
      if $D_k = D(x_k)$ **then**
        if $f(x_k) = f(x_{min})$ **then** STOP
        else Imposta $y = x_{min}$ e $D = D_{k+1} = D(x_k)$
      else
        Genera $D_{k+1} \supset D_k$, imposta $\tilde{\alpha}_{k+1}^{(d)} = 1$, per ogni $d \in D_{k+1}$ e $D = D_{k+1} \setminus D_k$
      end if
    else
      Imposta $D_{k+1} = D_k$ e $D = \{d \in D_k :$ *Nonmonotone Search* fallisce con $\tilde{\alpha}_k^{(d)} > 1\}$
    end if
  else
    Imposta $D_{k+1} = D_k$ e $D = D_k$
  end if
  Imposta $x_{k+1} = y$, $\bar{f}_{k+1} = f^{ref}$
end for

Due sono le questioni che a questo punto bisogna affrontare: come avviene la ricerca di tipo non-monotona e come vengono generate le nuove direzioni di ricerca.

3.0.1 Ricerca non-monotona

La scelta di una ricerca di tipo non-monotono è molto importante in questo contesto perché migliora la robustezza e l'efficienza del metodo. Infatti se si cerca di ottenere un nuovo punto che riduca strettamente la funzione obiettivo, come accade quando usiamo una ricerca di tipo monotono, si può finire con l'avere piccoli spostamenti lungo le direzioni di ricerca o generare molte direzioni primitive per cercare di fuggire da un minimo locale.

Lo schema per la ricerca nonmonotona è riportato nell'algoritmo successivo. L'algoritmo dapprima calcola il passo iniziale per la ricerca. Questo viene scelto come il minimo tra $\tilde{\alpha}_k^{(d)}$ (che definisce la variazione dinamica dello skewed stencil) e il passo più grande $\bar{\alpha}$ che può essere preso lungo la direzione di ricerca d (vedi Inizializzazione). Se il passo iniziale è maggiore di zero e la funzione si riduce lungo la direzione di ricerca rispetto al valore di riferimento f^{ref}, la ricerca inizia espandendo il passo e continua finché viene raggiunto il massimo passo o non può più essere garantita una riduzione rispetto al valore di riferimento (vedi Step 3). Notiamo che la ricerca di linea si muove lungo la direzione garantendo l'ammissibilità (i punti scelti appartangono al lattice).

Algorithm 14 Nonmonotone Search

Input. $\tilde{\alpha}_k^{(d)}, x_k, d, f^{ref}$
Inizializzazione. Calcola il più grande $\bar{\alpha}$ tale che $x_k + \bar{\alpha}d \in X \cap \mathbb{Z}^n$. Imposta $\alpha = \min\{\bar{\alpha}, \tilde{\alpha}_k^{(d)}\}$.
Step 1. **If** $\alpha > 0$ e $f(x_k + \alpha d) < f^{ref}$ **then** Vai a Step 2
else Imposta $\alpha = 0$ e vai a Step 5
Step 2. Sia $\beta = \min\{\bar{\alpha}, 2\alpha\}$
Step 3. **If** $\alpha = \bar{\alpha}$ o $f(x_k + \beta d) \geq f^{ref}$ **then** imposta $\tilde{\alpha}_{k+1}^{(d)} = \alpha$ e vai a Step 5
Step 4. Imposta $\alpha = \beta$ e vai a Step 2
Step 5. Ritorna α

3.0.2 Generazione di nuove direzioni di ricerca

La procedura usata per la generazione di nuove direzioni di ricerca per l'algoritmo BBOA, riportata in [1], fa uso della sequenza di Halton [17] e viene riportata nell'algoritmo successivo.

Algorithm 15 Generazione di nuove direzioni di ricerca

Input. $t > 0$, $\eta > 0$, D_k
If $\eta < 50\sqrt{n}/2$
  For h = 1,...,1000
    Poni $t \leftarrow t + 1$
    Sia u_t il t-esimo vettore nella sequenza n dimensionale di Halton
    Calcola $q_t(\eta) = \lfloor \eta \frac{2u_t - e}{||2u_t - e||} \rceil \in \mathbb{Z}^n \cap [-\eta - \frac{1}{2}, \eta + \frac{1}{2}]^n$
    If $q_t(\eta)$ è un vettore primitivo e $q_t(\eta) \notin D_k$
      Poni $D_{k+1} = D_k \cup \{q_t(\eta)\}$
      return (success,t,η,D_{k+1})
    Endif
  Endfor
Endif
return (failure,t,η,D_k)

3.1 Risultati sulla convergenza

Per quanto riguarda la convergenza del metodo a un punto di minimo locale si ha il seguente risultato.

Teorema 3.1. *Siano $\{x_k\}$ e $\{\bar{f}_k\}$ sequenze di soluzioni e valori di riferimento, rispettivamente, generate da NM-BBOA. Allora, l'algoritmo non cicla e produce un punto di minimo locale.*

Dimostrazione. Prima di tutto notiamo che la sequenza $\{\bar{f}_k\}$ è limitata inferiormente, dal momento che il numero di soluzioni nella regione ammissibile è finito. Successivamente definiamo

$$K = \{k : x_{k+1} \neq x_k\} \tag{3.1}$$

e

$$H(k, \bar{k}) = \{h \in K : k < h \leq \bar{k}\}, \text{ per } \bar{k} \geq k. \tag{3.2}$$

Sia poi $\bar{k}(M)$ l'indice per cui

$$|H(k, \bar{k}(M))| = M. \tag{3.3}$$

Per ogni $k \in K$, abbiamo

$$f(x^{k+1}) < \bar{f}_k. \tag{3.4}$$

Inoltre, dalla regola di aggiornamento di f^{ref} e la definizione di $\bar{f}_k$, abbiamo che

$$\bar{f}_{k+1} \leq \bar{f}_k \tag{3.5}$$

Ricordando che $|C \cap \mathbb{Z}^n| < \infty$, possiamo definire

$$0 < \delta = \min_{x,y \in C \cap \mathbb{Z}^n} \{|f(x) - f(y)| : f(x) \neq f(y)\} \tag{3.6}$$

così da avere

$$\bar{f}_{\bar{k}(M)} < \bar{f}_k - \delta. \tag{3.7}$$

Proviamo ora che l'algoritmo NM-BBOA non cicla. Per assurdo, supponiamo che la sequenza $\{x_k\}$ sia infinita. In questo caso è facile osservare che anche l'insieme K è infinito. Dal momento che la procedura non termina, una soluzione $\bar{x}$ (che non è un minimo locale) viene generata un infinito numero di volte. Per (3.5) e (3.7), esiste un'iterazione $\tilde{k}$ tale che

$$\bar{f}_{\tilde{k}} \leq f(\tilde{x}) \tag{3.8}$$

Inoltre, poiché $\tilde{x}$ è generata un numero infinito di volte, deve esistere un'iterazione $\widehat{k} \geq \tilde{k}$ tale che

$$f(\tilde{x}) < \bar{f}_{\widehat{k}}. \tag{3.9}$$

Quindi abbiamo

$$f(\tilde{x}) < \bar{f}_{\widehat{k}} \leq \bar{f}_{\tilde{k}} \leq f(\tilde{x}) \tag{3.10}$$

che dimostra che la procedura di ricerca locale non può ciclare.
Mostriamo infine che il punto prodotto x^* è un minimo locale per il problema. Quando NM-BBOA si arresta, sia $\bar{k}$ l'ultimo indice di iterazione, così che $x^* = x_{\bar{k}}$ e $x_{\bar{k}} = x_{min}$. Inoltre, $D_{\bar{k}} = D(x^*)$ è l'insieme di tutte le direzioni ammissibili e relativamente prime in x^*. Abbiamo che

$$f(x^*) \leq \bar{f}_{\bar{k}} \tag{3.11}$$

e dalle istruzioni dell'algoritmo

$$\bar{f}_{\bar{k}} \leq f(x^* + d), \forall d \in D(x^*). \tag{3.12}$$

Quindi combinando le disuguaglianze (3.11) e (3.12), otteniamo che x^* è un minimo locale. □

3.2 Confronto BBOA e algoritmi precedenti

Dai test effettuati [1] l'algoritmo BBOA si è dimostrato un buon algoritmo per la risoluzione di problemi black box a variabili intere. Come riportato in [1], se siamo interessati a confrontare l'efficienza e la robustezza dei codici BBOA, DCS e FS, nelle loro versioni monotona e non-monotona, e dei codici NOMAD e BFO su problemi (1.2) con insieme ammissibile di tipo box, è stato dimostrato che entrambi gli algoritmi NM-BBOA e M-BBOA presentano una maggiore efficienza e robustezza rispetto a tutti gli altri metodi. Inoltre, su problemi black box con vincoli black box si sono ottenuti risultati simili. L'algoritmo BBOA è dunque risultato in generale migliore rispetto agli altri algoritmi e per questo una buona alternativa per la risoluzione di problemi del tipo (1.2).

Capitolo 4

Modifica dell'algoritmo BBOA

Presentiamo in questo capitolo una modifica dell'algoritmo NM-BBOA descritto nel *Capitolo 3*.

L'esigenza di una modifica di tale algoritmo deriva dalla volontà di trovare un metodo in grado di risolvere i problemi black box a variabili intere utilizzando un numero minore di calcoli di funzione rispetto alla versione non-monotona del BBOA.

4.1 Generazione di direzioni attraverso l'uso di coni

L'idea sviluppata in questa sezione è quella di operare una modifica nel codice di generazione delle nuove direzioni di ricerca presentato nel *Capitolo 3*. Nell'algoritmo 15, la generazione di nuove direzioni di ricerca si basava esclusivamente sull'uso della sequenza di Halton [17]. Nel momento in cui risultava necessario generare una nuova direzione di ricerca da inserire nell'insieme delle direzioni, l'algoritmo NM-BBOA nella versione base generava punti attorno alla soluzione x_k corrente, attraverso una generazione pseudorandomica basata sulla sequenza di Halton, senza dunque tenere conto delle informazioni che l'algoritmo poteva precedentemente aver raccolto. La generazione si arrestava una volta che veniva trovata una direzione primitiva che non fosse già presente nell'insieme delle direzioni utilizzato fino a quel momento. Tale direzione veniva inserita nell'insieme delle direzioni e la ricerca di una solu-

zione per il problema ripartiva.
Nella modifica proposta le nuove direzioni non vengono più generate randomicamente attorno al punto x_k, ma sono ricercate all'interno di un cono opportunamente costruito, il quale va a delimitare un insieme di punti promettente sul lattice. Nello specifico, a partire dalla soluzione corrente x_k viene considerata l'ultima direzione in cui la ricerca non-monotona *Nonmonotone Search* ha avuto successo, attorno a tale direzione viene costruito un cono, con vertice nel punto x_k, e all'interno di questo cono saranno ricercate le nuove direzioni. La scelta di prendere questa direzione come direzione attorno a cui generare il cono è data dal fatto che, essendosi la ricerca spostata in quella direzione, è più probabile che nella zona "indicata" da quella direzione si trovi la soluzione del problema.

4.2 Descrizione dell'algoritmo

Riportiamo di seguito la descrizione dettagliata dell'algoritmo proposto per la generazione di nuove direzioni e il suo pseudocodice.

La versione alternativa di generazione di nuove direzioni per l'algoritmo NM-BBOA qui proposta prende in input:

- x_k, soluzione corrente;
- f_k, valore della funzione obiettivo calcolata sulla soluzione corrente;
- d_k, l'ultima direzione lungo cui la ricerca non-monotona *Nonmonotone Search* ha avuto successo;
- t, un contatore che tiene traccia del numero di volte che la direzione d_k è già stata utilizzata per la generazione di un cono;
- D_k, l'insieme delle direzioni di ricerca finora usato.

Nello Step 2 l'algoritmo controlla se la direzione d_k coincide con la direzione utilizzata nella precedente ricerca di direzioni. Se così è, il contatore t registra un aumento di 1 nel suo valore. Altrimenti t viene impostato a 0, specificando quindi che la direzione è cambiata e la ricerca verrà effettuata in una zona diversa dalla ricerca precedente.
Una volta aggiornato tale contatore, l'algoritmo passa alla generazione del

Algorithm 16 Generazione di nuove direzioni di ricerca versione modificata

1: **Input.** x_k, f_k, d_k direzione su cui si vuole generare il cono, t numero di volte che tale direzione è già stata usata, D_k insieme delle direzioni finora usato
2: **If** d_k coincide con la direzione usata per la precedente generazione del cono
3: Aggiorna $t \leftarrow t + 1$
4: **Else** $t \leftarrow 0$
5: **Endif**
6: **If** f_k ha subito un aumento, rispetto ai valori precedentemente assunti
7: Imposta l'ampiezza del cono a $100 + 5n°$.
8: **If** $100 + 5n > 120$
9: Imposta l'ampiezza del cono a 120°.
10: **Endif**
11: **Else** Imposta l'ampiezza del cono a $15n°$
12: **If** $15n > 90$.
13: Imposta l'ampiezza del cono a 90°.
14: **Endif**
15: **Endif**
16: Calcola la probabilità $p = Prob(n, t)$ da dare al cono.
17: Genera con probabilità p una nuova direzione q dentro al cono o con probabilità $1 - p$ una nuova direzione q fuori dal cono.
18: **If** q è un vettore primitivo e $q \notin D_k$
19: Aggiungi q a D_k.
20: return(success,D_k,t,d_k)
21: **Endif**
22: return(failure,D_k,t,d_k)

cono attorno alla direzione d_k. Una questione importante a questo punto risulta essere la scelta dell'ampiezza del cono. In generale l'ampiezza del cono viene posta a $15n°$, con n dimensione del problema (Step 11). Laddove n risulti maggiore di 6, e dunque l'ampiezza del cono risulti maggiore di 90°, l'ampiezza viene comunque fissata a 90° (Step 13). Nel caso in cui sia stato registrato un aumento del valore della funzione obiettivo, dovuto all'azione della ricerca non-monotona *Nonmonotone Search*, l'ampiezza del cono assume valori differenti: tale ampiezza viene posta a $100 + 5n°$ (Step 7), ma mai maggiore di 120° (Step 9). L'aumento di valore della funzione obiettivo avviene nel momento in cui l'algoritmo cerca di fuggire da possibili minimi locali. Permettere all'algoritmo di utilizzare coni con ampiezza maggiore in queste situazioni, consente all'algoritmo di fuggire più velocemente da tali punti.
Una volta generato il cono, l'algoritmo calcola la probabilità p da dare a tale regione (Step 16). Tale valore indica la probabilità che la nuova direzione che

si vuole trovare appartenga alla regione descritta dal cono. La probabilità assegnata di default è $p = \frac{3}{4}$ per la regione descritta dal cono e $1 - p = \frac{1}{4}$ per la regione complementare. Tale probabilità non resta però invariata, ma viene resa variabile attraverso la funzione $Prob(n, t)$. Nel caso in cui t arrivi ad assumere una valore molto alto, indice del fatto che la direzione d_k e di conseguenza il cono generato continuano ad essere gli stessi da diverso tempo, la probabilità assegnata al cono inizia a diminuire lentamente a favore della probabilità della regione complementare. Questo perché se per molto tempo si sono ricercate direzioni all'interno del cono senza successo, si è indotti a pensare che o stiamo cercando nella zona sbagliata o siamo su un probabile punto di minimo globale. In entrambi i casi risulta conveniente aumentare la ricerca di nuove direzioni nella regione complementare al cono. La probabilità p assegnata al cono non sarà comunque mai minore di $\frac{2}{5}$. Di seguito presentiamo l'algoritmo per $Prob(n, t)$.

Algorithm 17 $Prob(n, t)$

Dati. $n > 0$, $t \geq 0$
a. Calcola $p = \frac{3}{4} exp(-\frac{t}{100n})$
b. If $p \leq \frac{2}{5}$
$p = \frac{2}{5}$
Endif
c. return(p)

Dopo aver impostato il valore di ampiezza del cono e la probabilità assegnata a ogni regione, ha inizio la ricerca di una nuova direzione (Step 17). Tale ricerca avviene attraverso la generazione di punti pseudorandomici nello spazio. In questo caso sono state implementate due generazioni, entrambe basate sulla sequenza di Halton [17], sulla base di quella presentata nell'algoritmo 15. Una sequenza di punti è stata utilizzata per generare nuove direzioni all'interno della regione descritta dal cono, l'altra sequenza è stata usata per generare nuove direzioni all'interno della regione complementare. Nel momento in cui una probabile nuova direzione viene trovata, l'algoritmo verifica che siano soddisfatte due condizioni (Step 18):

- la nuova direzione non deve essere già presente nell'insieme delle direzioni D_k che finora si è usato;

- la nuova direzione deve essere una direzione primitiva.

Se tali condizioni sono soddisfatte, la nuova direzione viene aggiunta all'insieme D_k (Step 19) e l'algoritmo ritorna a cercare una soluzione utilizzando l'insieme di direzioni aggiornate. Se tali condizioni non sono soddisfatte, l'algoritmo ritorna un insuccesso.

Capitolo 5

Risultati

In questo capitolo riportiamo i risultati ottenuti dal confronto dell'algoritmo NM-BBOA nella sua versione base (descritta nel *Capitolo 3*) e modificata (descritta nel *Capitolo 4*).

5.1 Performance e Data profiles

Presentiamo dapprima gli strumenti utilizzati per valutare le prestazioni degli algoritmi, ovvero i performance e data profiles per metodi senza derivate proposti da Moré e Wild [9]. I performance profiles, introdotti per la prima volta da Dolan e Moré [10], si sono rivelati importanti per l'analisi delle prestazioni di solutori per problemi di ottimizzazione. Dolan e Moré definirono un *benchmark* (*prova di prestazione*) attraverso un insieme P di problemi benchmark, un insieme S di solutori e un test di convergenza T. Una volta che questi componenti sono definiti, i performance profiles possono essere usati per comparare le performance dei solutori.
Il test di convergenza non deve dipendere da valutazioni del gradiente. Quello proposto in [9] è il seguente:

$$f(x_0) - f(x) \geq (1 - \tau)(f(x_0) - f_L), \tag{5.1}$$

dove f_L è il più piccolo valore di f ottenuto da qualsiasi solutore entro un dato valore μ_f di valutazioni di funzione, $\tau > 0$ è il parametro di precisione che solitamente è posto a 10^{-k}, con $k \in \{1, 3, 5, 7\}$ e x_0 è il punto di partenza del problema.

5.1.1 Performance profiles

I performance profiles sono definiti attraverso una misura di performance $t_{p,s} > 0$ ottenuta per ogni problema $p \in P$ e solutore $s \in S$. Nel caso dei problemi black box $t_{p,s}$ è definito come il numero di valutazioni di funzione necessari al solutore s per soddisfare la convergenza data dal test (5.1) sul problema p. Di conseguenza, valori elevati di $t_{p,s}$ indicano una cattiva performance. Poniamo inoltre $t_{p,s} = +\infty$ quando il test di convergenza fallisce entro il numero di valutazioni di funzione fissato.
Fissato un problema $p \in P$, si definisce il *performance ratio* $r_{p,s}$ di un solutore s come:

$$r_{p,s} = \frac{t_{p,s}}{min\{t_{p,s} : s \in S\}} \tag{5.2}$$

ovvero il rapporto tra il numero di valutazioni di funzione impiegato dal solutore s e il miglior numero di valutazioni di funzione impiegato da qualunque solutore per soddisfare (5.1). Pertanto, $r_{p,s} = 1$ se e solo se s è uno dei migliori solutori per il problema p e $r_{p,s} = +\infty$ se e solo se s ha fallito la convergenza sul problema p.
Il *performance profile* di un solutore $s \in S$ è definito come la frazione di problemi in cui il performance ratio è al più α, ovvero:

$$\rho_s(\alpha) = \frac{1}{|P|} size\{p \in P : r_{p,s} \leq \alpha\} \tag{5.3}$$

dove $|P|$ indica la cardinalità di P. $\rho_s(\alpha)$ rappresenta dunque la distribuzione di probabilità per $r_{p,s}$. I performance profiles cercano di catturare quanto un certo solutore $s \in S$ sia performante su un insieme di problemi P, rispetto agli altri solutori. Si noti che $\rho_s(1)$ indica la frazione di problemi per i quali il solutore $s \in S$ ha performance migliori e, per α sufficientemente grande, $\rho_s(\alpha)$ indica la frazione di problemi risolti dal solutore $s \in S$. In generale, $\rho_s(\alpha)$ è la frazione di problemi con un performance ratio $r_{p,s}$ limitato da α, per questo motivo solutori con un alto valore per $\rho_s(\alpha)$ sono preferibili.

5.1.2 Data profiles

I performance profiles forniscono una visione accurata delle relative performance dei solutori entro un fissato numero di valutazioni di funzione μ_f. Ciononostante, i performance profiles non forniscono sufficiente informazione

per problemi di ottimizzazione dispendiosi. Se si hanno problemi di ottimizzazione dispendiosi si è spesso interessati alla performance dei solutori come funzione del numero di valutazioni di funzione. In altre parole si è interessati alla *percentuale di problemi che possono essere risolti (data una tolleranza τ) con κ valutazioni di funzione* [9].
Questa informazione può essere fornita ponendo:

$$d_s(\alpha) = \frac{1}{|P|} size\{p \in P : t_{p,s} \leq \alpha\} \tag{5.4}$$

che indica la percentuale di problemi che possono essere risolti con α valutazioni di funzione. Questa definizione di d_s risulta però indipendente dal numero di variabili presenti nel problema $p \in P$, e di conseguenza è non realistica. Nella nostra esperienza, infatti, il numero di valutazioni di funzione necessarie per soddisfare un dato test di convergenza tende a crescere con l'aumentare del numero di variabili. I *data profile* sono allora definiti sostituendo il numero di valutazioni di funzione con il numero di simplex gradients, ovvero:

$$d_s(\alpha) = \frac{1}{|P|} size\{p \in P : \frac{t_{p,s}}{n_p + 1} \leq \alpha\}. \tag{5.5}$$

dove n_p è il numero di variabili del problema $p \in P$. I data profiles indicano la percentuale problemi che il solutore s riesce a risolvere con α simplex gradients o meno e sono utili se si vuole uno strumento che possa aiutare a scegliere qual è il solutore che riesce a raggiungere una certa riduzione in valori di funzione.

5.2 Risultati per problemi con vincoli box

Un *problema con vincoli box (interi)* è un problema del tipo (1.2) con $C = \{x \in \mathbb{R}^n : l_i \leqslant x_i \leqslant u_i, i = 1, ..., n\}$, $l_i, u_i \in \mathbb{Z}$ e $-\infty \leqslant l_i \leqslant u_i \leqslant +\infty$ per ogni $i = 1, ..., n$, ovvero:

$$\begin{aligned} min\ & f(x) \\ l_i \leqslant\ & x_i \leqslant u_i \\ i =\ & 1, ..., n \\ x \in\ & \mathbb{Z}^n \end{aligned} \tag{5.6}$$

Per confrontare le performances dell'algoritmo NM-BBOA, nella sua versione base e modificata, su problemi con vincoli box, usiamo 61 problemi non vincolati e nonsmooth, 48 dei quali derivanti da [4]. Più precisamente usiamo i problemi minmax non vincolati, presenti nella sezione 2 [4], e i problemi nonsmooth non vincolati, presenti nella sezione 3 [4]. Poiché tali problemi sono in realtà non vincolati, vengono aggiunti dei vincoli box sulle variabili

$$l_i = (\tilde{x}_0)_i - 10 \leq \tilde{x}_i \leq (\tilde{x}_0)_i + 10 = u_i,\ i = 1, ..., n \tag{5.7}$$

dove $\tilde{x}_0$ è il punto di partenza fornito al problema. In seguito, dato il problema box a variabili continue

$$\begin{aligned} min\ & \tilde{f}(\tilde{x}) \\ l_i \leqslant\ & \tilde{x}_i \leqslant u_i \\ i = 1, ..., n & \\ \tilde{x} \in\ & \mathbb{R}^n \end{aligned} \tag{5.8}$$

consideriamo il problema discreto

$$\begin{aligned} min\ & f(x) \\ 0 \leqslant\ & x_i \leqslant 100 \\ i = 1, ..., n & \\ x \in\ & \mathbb{Z}^n \end{aligned} \tag{5.9}$$

dove $f(x) = \tilde{f}(y)$ con

$$y_i = l_i + x_i(u_i - l_i)/100,\ i = 1, ..., n \tag{5.10}$$

Per quanto riguarda il punto iniziale x_0 per il problema (5.9), poniamo

$$(x_0)_i = 50,\ i = 1, ..., n \tag{5.11}$$

e notiamo che il punto x_0 non è altro che il punto

$$(x_0)_i = \lfloor 100((\tilde{x}_0)_i - l)/(u - l) \rceil,\ i = 1, ..., n \tag{5.12}$$

dove $\lfloor \cdot \rceil$ denota l'operatore che restituisce l'intero più vicino.

Riportiamo di seguito i risultati in termini di data e performance profiles per valori di τ in $\{10^{-1}, 10^{-3}, 10^{-5}, 10^{-7}\}$. Il numero massimo di valutazioni di funzione è stato impostato a 5000.

5.2.1 Risultati ottenuti sui 61 problemi con vincoli box

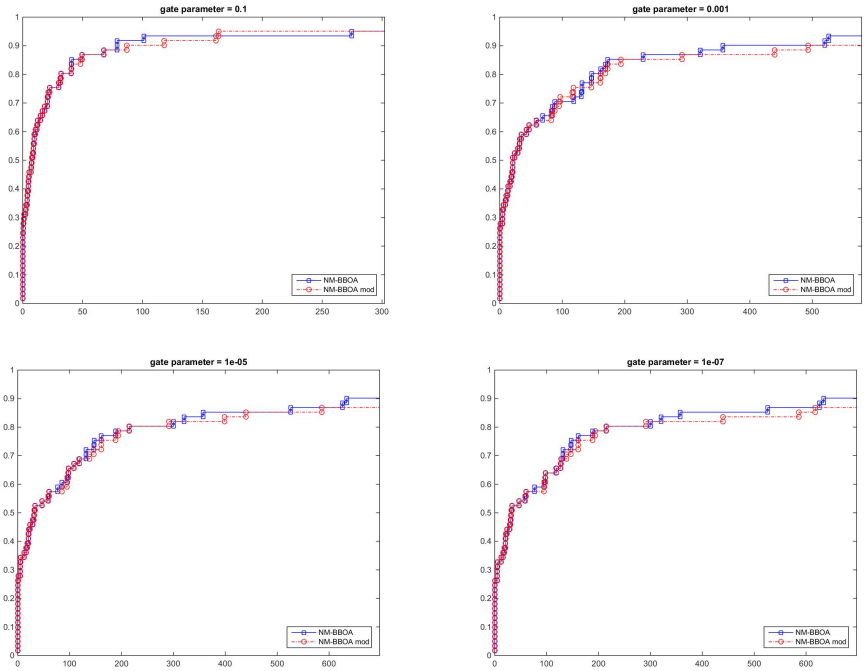

Figura 5.1: Data profiles dei 61 problemi con vincoli box

L'efficienza e la robustezza dell'algoritmo NM-BBOA e dell'algoritmo NM-BBOA modificato vengono presentate attraverso i performance e data profiles qui riportati.
Per i 61 problemi con vincoli di tipo box, i grafici mostrano che le due versioni dell'NM-BBOA sono abbastanza equivalenti. La versione NM-BBOA con la modifica proposta nel *Capitolo 4* non sembra aver portato a sostanziali miglioramenti nei risultati. Utilizzare l'uno o l'altro algoritmo è abbastanza indifferente.

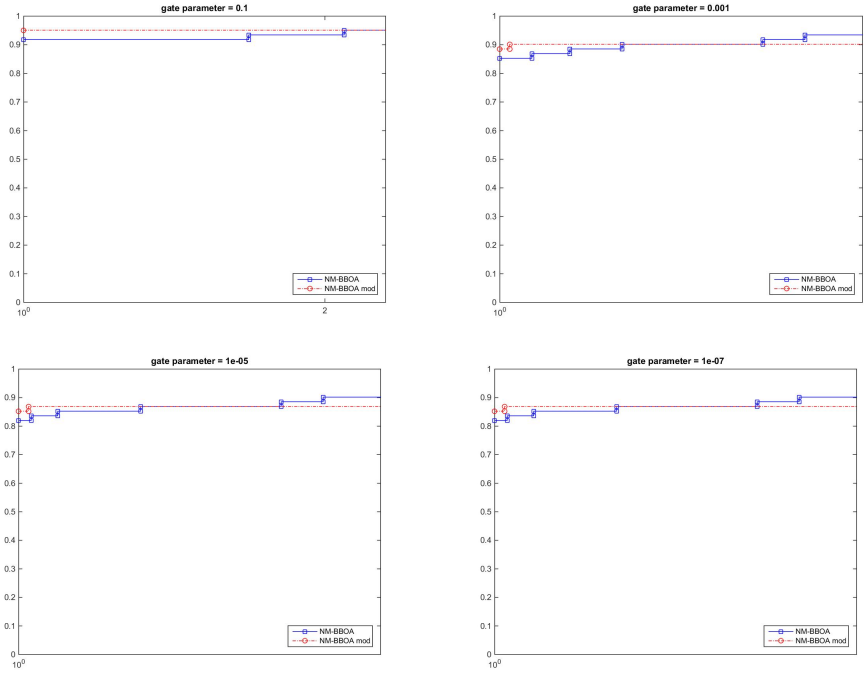

Figura 5.2: Performance profiles dei 61 problemi con vincoli box

5.2.2 Risultati ottenuti sui 32 problemi con vincoli box di dimensione < 10

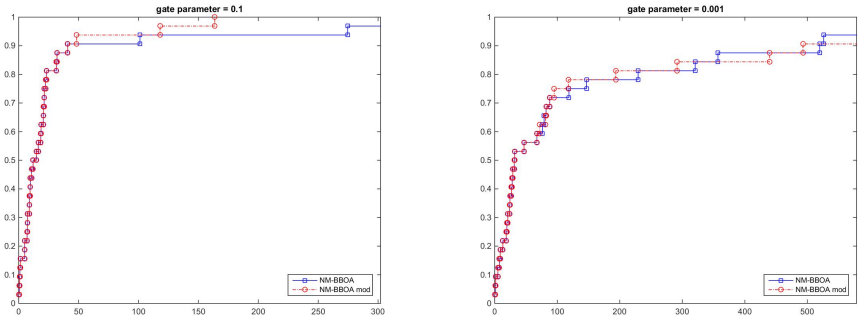

Figura 5.3: Data profiles dei 32 problemi con vincoli box di dimensione < 10

Procedendo con un'analisi ristretta ai soli problemi di dimensione < 10 si ottiene che la versione modificata dell'algoritmo NM-BBOA risulta essere migliore in termini di efficienza e robustezza rispetto alla versione base. Come riporta anche la Tabella 5.2.2, infatti, per certi problemi l'algoritmo NM-BBOA nella versione modificata riesce a trovare una soluzione migliore in termini di calcoli di funzione, rispetto all'algoritmo NM-BBOA. In alcuni casi, in particolare per i problemi di dimensione più piccola e più semplici, i due algoritmi hanno restituito lo stesso risultato con lo stesso numero di calcoli di funzione. Ci si è resi poi conto che ciò avveniva perché il problema passato all'algoritmo era così semplice da venire risolto con il solo uso dell'insieme di direzioni iniziali passate all'algoritmo. Qualsiasi altra direzione aggiunta in seguito a tale insieme non poteva dunque portare a miglioramenti in quanto l'algoritmo aveva già trovato il punto di minimo del problema. Di conseguenza la modifica apportata all'algoritmo non aveva modo di portare a miglioramenti nella ricerca di una soluzione.

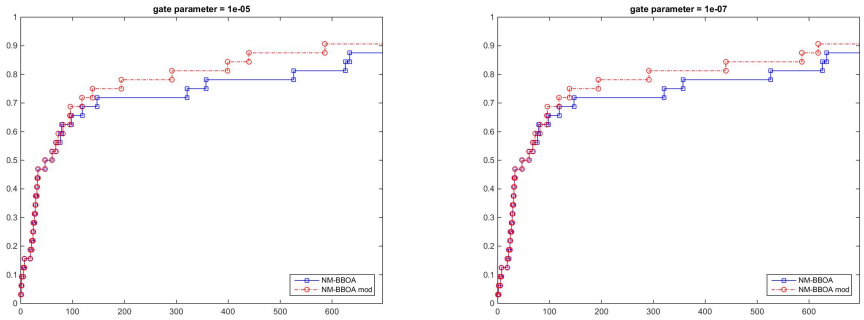

Figura 5.4: Data profiles dei 32 problemi con vincoli box di dimensione < 10

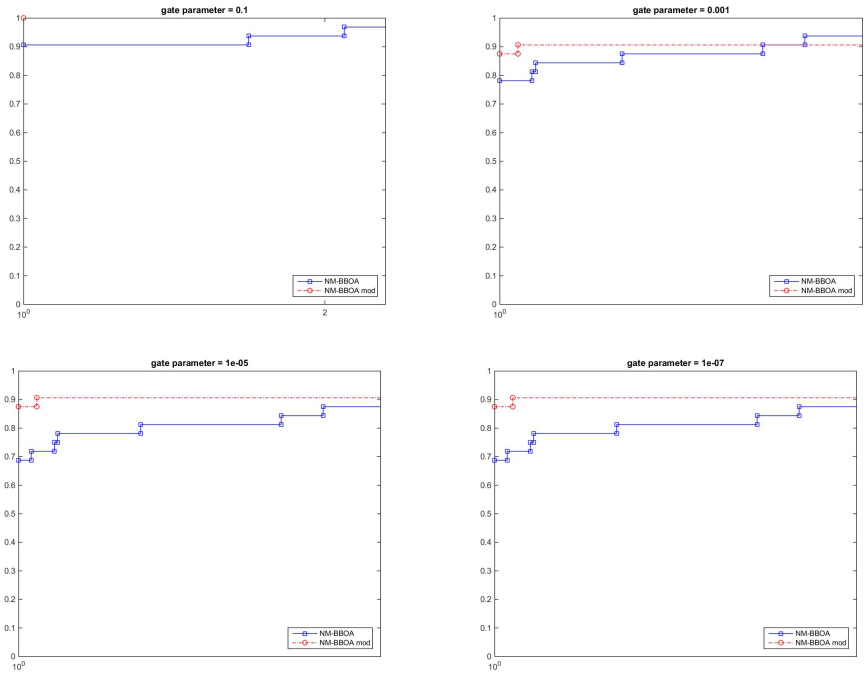

Figura 5.5: Performance profiles dei 32 problemi con vincoli box di dimensione < 10

Tabella con alcuni risultati ottenuti sui problemi con vincoli box e dimensione < 10. Nella tabella seguente vengono riportati i risultati ottenuti su alcuni problemi con vincoli box e dimensione < 10. Indichiamo con:

- *NM-BBOA*, l'algoritmo NM-BBOA nella versione base descritta nel *Capitolo 3*;
- *NM-BBOA mod*, l'algoritmo NM-BBOA con la modifica descritta nel *Capitolo 4*;
- n, la dimensione del problema;
- nf, il numero di calcoli di funzione effettuati dall'algoritmo per trovare una soluzione;
- f^*, il valore assunto dalla funzione obiettivo nella soluzione trovata.

		NM-BBOA		NM-BBOA mod	
Problemi	**n**	**nf**	**f***	**nf**	**f***
spiral	2	1223	+2.69E-05	1146	+2.69E-05
oet5	4	4877	+4.00E-02	3256	+4.00E-02
kowalik-osborne	4	1846	+2.05E-02	1141	+2.05E-02
shor	5	1279	+2.33E+01	659	+2.33E+01

Tabella 5.1: Alcuni risultati ottenuti sui problemi con vincoli box e dimensione $n < 10$.

Conclusioni e Future Work

Dai risultati precedentemente presentati si può concludere che l'algoritmo NM-BBOA nella versione modificata risulta, in termini di efficienza e robustezza, molto simile all'algoritmo base. Risultati migliori rispetto all'algoritmo base si ottengono però per problemi di dimensione < 10. Ciò porta dunque a concludere che per problemi black box a variabili intere e di dimensione contenuta, la versione modificata dell'algoritmo NM-BBOA sia una valida opzione per la loro risoluzione.
In futuro potrebbe essere utile lavorare a un'ulteriore modifica dell'algoritmo per riuscire ad ottenere buoni risultati anche sui problemi black box a variabili intere di dimensione ≥ 10.

Bibliografia

[1] G. Liuzzi, S. Lucidi, F. Rinaldi, *An algorithmic framework based on primitive directions and nonmonotone line searches for black box problems with integer variables*

[2] G. Liuzzi, S. Lucidi, F. Rinaldi, *Derivative-free methods for bound constrained mixed-integer optimization*, Comput. Optim. Appl., DOI 10.1007/s10589-011-9405-3, 2011

[3] J. Larson, M. Menickelly, S. M. Wild, *Derivative-free optimization methods*,Acta Numerica, 2019

[4] V. Luksan, J.Vlcek, *Test problems for nonsmooth unconstrained and linearly constrained optimization.* Technical report VT798-00, Institute of Computer Science, Academy of Science of the Czech Republic (2000)

[5] L. Grippo, M. Sciandrone, *Metodi di ottimizzazione non vincolata*, Cap.14, Springer, 2011

[6] S. Lucidi, M. Sciandrone, *On the global convergence of derivative-free methods for unconstrained optimization*, Society for Industrial and Applied Mathematics, 2002

[7] T. G. Kolda, R. M. Lewis, V. Torczon, *Optimization by Direct Search: New Perspectives on Some Classical and Modern Methods*, Society for Industrial and Applied Mathematics, 2003

[8] J. Larson, S. Leyffer, P. Palkar, S.M.Wild, *A Method for Convex Black-Box Integer Global Optimization*, 2019

[9] J.J.Moré, S.M.Wild, *Benchmarking Derivative-Free Optimization Algorithms*, Society for Industrial and Applied Mathematics, 2009

[10] E. D. Dolan, J. J. More', *Benchmarking Optimization Software with Performance Profiles*, Math. Program., 91 (2002).

[11] Abramson, M., Audet, C., Couture, G., Jr., J.D., Digabel, S.L.. The NOMAD project. URL `https://www.gerad.ca/nomad/`

[12] S. Le Digabel, C. Tribes, V. Rochon Montplaisir, C. Audet. *NOMAD User Guide, Version 3.9.1.*

[13] M. Porcelli, P.L. Toint. BFO, a trainable derivative-free solver for mixed integer bound-constrained optimization, URL `https://sites.google.com/site/bfocode/home`

[14] M. Porcelli, P.L. Toint. *BFO, a trainable derivative-free Brute Force Optimizer for nonlinear bound-constrained optimization and equilibrium computations with continuous and discrete variables.* ACM Transactions on Mathematical Software, 2017

[15] J. Müller, *MISO: Mixed-Integer Surrogate Optimization Framework*

[16] S. Lucidi, M. Sciandrone, *A derivative-free algorithm for bound constrained optimization.* Comput. Optim. Appl. 21(2), 119–142 (2002)

[17] J. Halton, *On the efficiency of certain quasi-random sequences of points in evaluating multi-dimensional integrals*, Numerische Mathematik 2, 84–90 (1960)

[18] J. Müller, *Surrogate Model Algorithms for Computationally Expensive Black-Box Global Optimization Problems*, 2012