POLITECNICO DI TORINO

Corso di Laurea Magistrale
in Ingegneria Matematica

Tesi di Laurea Magistrale

Algoritmi di dinamiche di sciami di particelle per l'ottimizzazione del Machine Learning

Relatori
prof. Marco Scianna
prof. Luigi Preziosi
Firma dei relatori

.....

.....

Candidato
Michela Trapani

Firma del candidato

.....

Anno Accademico 2020-2021

ABSTRACT

L'ottimizzazione di sciami di particelle (PSO) è un modello particellare che riproduce dinamiche collettive di sistemi di agenti, come sciami, utilizzato in ottica di ottimizzazione.
Il PSO trae ispirazione proprio da alcuni modelli che simulano il comportamento sociale degli animali, come ad esempio gli stormi di uccelli che si muovono in sincrono per ricercare del cibo. In quest'ottica, nel PSO sono assegnate delle regole di movimento ai singoli agenti che li portano ad esplorare il dominio spaziale alla ricerca di punti critici: ad esempio, di punti che minimizzano una certa funzione obiettivo. In particolare ogni agente si muove anche per condivisione di informazioni con il resto dello sciame (o con una sua parte).
In questo lavoro di tesi, si analizza innanzitutto l'origine del PSO, ovvero si commentano alcuni modelli analoghi che lo hanno preceduto. Si passa poi a descrivere i suoi aspetti principali e alcune sue varianti; particolare attenzione è data all'interpretazione dei vari parametri presenti nel modello e alle strutture di comunicazione, che definiscono il gruppo di agenti con cui ogni individuo scambia informazioni durante la sua dinamica. Si propone infine l'implementazione di una delle versioni dell'algoritmo PSO tramite il software Matlab e la sua applicazione ad alcune funzioni test. In quest'ottica, si commenteranno ed interpreteranno i risultati derivanti da variazioni di alcuni coefficienti caratteristici del modello e della tipologia di comunicazione tra gli individui.

Indice

Elenco delle figure

Introduzione

L'algoritmo di ottimizzazione di sciami di particelle (noto con l'acronimo PSO - *particle swarm optimization*) è una tecnica di ottimizzazione stocastica basata sulla dinamica degli sciami, proposta da Eberhart e Kennedy nel 1995 [KE95]. Fa parte della famiglia degli algoritmi evolutivi e può essere utilizzato per risolvere diversi problemi che vanno dall'allenamento di reti neurali alla minimizzazione di funzioni non convesse. Successivamente, sono state proposte modifiche e miglioramenti, derivate nuove versioni, sviluppate nuove applicazioni e pubblicati studi teorici sugli effetti dei vari parametri e su aspetti specifici dell'algoritmo.
L'algoritmo PSO simula il comportamento sociale degli animali, tra cui insetti, uccelli e pesci. La versione originale fu ispirata dal comportamento sociale di stormi, di uccelli in movimento, con l'obiettivo di trovare il modello che permette a questi di volare in sincrono e cambiare improvvisamente direzione per poi raggrupparsi in una nuova formazione ottimale.
Gli sciami si conformano in modo cooperativo per trovare cibo e ogni membro nello sciame continua a cambiare il modello di ricerca secondo le esperienze di apprendimento di sé stesso e degli altri membri.
Gli individui, ovvero le particelle, vengono letteralmente fatti "volare" in uno spazio di ricerca multidimensionale $\mathbb{R}^n$, dove n è la dimensione dello spazio di ricerca. I movimenti delle particelle all'interno di questa regione sono influenzati dalla tendenza socio-psicologica che porta ogni individuo ad emulare il successo degli altri. La ricerca di una singola particella è pertanto legata a quella delle altre particelle all'interno del gruppo.
Dato quindi un problema di ottimizzazione di una funzione f di n variabili su un insieme o *regione ammissibile* X, la $f : \mathbb{R}^n \to \mathbb{R}$ prende il nome di *funzione obiettivo* mentre ciascun punto $\mathbf{x} = (x_1, x_2, ..., x_n)^T \in X$ costituisce una *soluzione ammissibile*. Il problema consiste quindi nel determinare un punto $\mathbf{x}^* \in X$, che rende minima la funzione f. Questo concetto può essere così specificato:

$$\mathbf{x}^* : f(\mathbf{x}^*) = min_{\mathbf{x} \in X} f(\mathbf{x})$$

È inoltre definita una *struttura di comunicazione*, che assegna ad ogni individuo un insieme di vicini con cui interagire. Ogni particella è in grado di valutare, tramite la funzione obiettivo, la bontà della propria posizione e di ricordare la migliore visitata. Questa informazione è condivisa tra tutti i vicini, così che ogni particella conosca anche la posizione migliore di tutti i vicini con cui interagisce. Gli individui nello sciame modificano la loro posizione in funzione della loro esperienza e di quella dei loro vicini.

Nello studiare il comportamento sociale degli animali, si è cercato di formare dei sistemi di vita artificiali, basandosi su alcuni punti che sono poi diventati principi guida [VDB+07]:

1. **Prossimità**: ogni particella interagisce con un certo numero di particelle topologicamente vicine piuttosto che con l'intero gruppo.

2. **Semplicità**: lo sciame è in grado di eseguire computazioni semplici dello spazio e del tempo.

3. **Qualità**: lo sciame è in grado di percepire il cambiamento di qualità nell'ambiente e di rispondervi.

4. **Diversa risposta**: lo sciame non deve rispondere al suo ambiente in modo assolutamente ordinato, ma deve essere preparato ad agire e rispondere con soluzioni diverse e alternative.

5. **Stabilità**: lo sciame non deve riformare i suoi modelli di comportamento ogni volta che avviene un cambiamento.

6. **Adattabilità**: tuttavia lo sciame dovrebbe essere in grado di adattare e cambiare il suo comportamento soltanto quando ne vale la pena.

La sua intuitivamente semplice rappresentazione e il suo relativamente basso numero di parametri regolabili rendono il PSO una scelta popolare per molti problemi che richiedono la ricerca di soluzioni ottime con una certa approssimazione. Tuttavia, in base al tipo di problema che può essere analizzato, il PSO presenta delle limitazioni, soprattutto per quanto riguarda funzioni con più ottimi locali, in cui le particelle "rischiano" di rimanere intrappolate. Inoltre i valori dei parametri che descrivono l'algoritmo (spesso determinati in base a numerosi test sperimentali) possono risultare efficaci per un determinato problema, ma non esserlo per altri. Tutto ciò ha portato negli anni a studi sempre più approfonditi volti alla mitigazione e alla risoluzione delle limitazioni del PSO [W+18].

La struttura di questa tesi è organizzata come segue. Nel primo capitolo si introduce l'algoritmo PSO originale, partendo da come ha avuto origine e quali sono i modelli da cui prende spunto e procedendo con la descrizione degli aspetti teorici e dei parametri principali che lo definiscono. Il secondo capitolo descrive le modifiche che sono state effettuate nel corso degli anni all'algoritmo originale: cambiamenti nati con lo scopo di migliorare alcuni aspetti poco ottimali del PSO. Nel terzo e nell'ultimo capitolo, si sceglie un modello di PSO tra quelli descritti, implementandolo e analizzando cosa succede andando a modificare alcuni aspetti dell'algoritmo, come ad esempio i parametri principali che governano il movimento delle particelle o il modo in cui esse comunicano tra di loro, applicandolo a due funzioni obiettivo e riportandone i risultati sperimentali.

1 Ottimizzazione dello sciame di particelle

Un sistema particellare è rappresentato da un'ampia collezione di agenti, ognuno rappresentato da un punto materiale. Si definisce punto materiale un corpo le cui dimensioni sono trascurabili rispetto al fenomeno preso in esame ed è solitamente caratterizzato dalle tre coordinate spaziali, dalle relative velocità e da una massa.
La dinamica con cui i punti materiali si muovono è basata sulle *equazioni fondamentali della dinamica*, in particolare sulla seconda legge di Newton:

$$m_i \mathbf{a}_i = \sum_{k=1}^{K} \mathbf{F}_{k,i} \tag{1}$$

dove $i = 1, 2, ..., N$ indica il numero di agenti presenti nel sistema, i quali hanno una certa massa m_i e una certa accelerazione $\mathbf{a}_i$; $\mathbf{F}_{k,i}$ indica la generica forza k, che agisce sul singolo agente i.
Considerando di discretizzare il tempo e quindi valutare l'equazione per t istanti discreti, la formula (1) diventa:

$$m_i \frac{\mathbf{v}_i(t + \Delta t) - \mathbf{v}_i(t)}{\Delta t} = \sum_{k=1}^{K} \mathbf{F}_{k,i} \tag{2}$$

dove $t = 1, 2, ..., T_{max}$, con T_{max} che rappresenta il numero massimo di iterazioni del processo, scelto a priori.
Considerando un passo temporale $\Delta t = 1$, è possibile definire l'*inerzia* come:

$$m_i \mathbf{a}_i(t + 1) \approx m_i[\mathbf{v}_i(t + 1) - \mathbf{v}_i(t)]$$

Inoltre per semplicità è possibile analizzare dei sistemi di particelle con massa unitaria, quindi $m_i = 1$; l'equazione (2) allora diventa:

$$\mathbf{v}_i(t + 1) - \mathbf{v}_i(t) = \mathbf{f}_i \tag{3}$$

dove le forze $\mathbf{f}_i = \sum_{k=1}^{K} \mathbf{F}_{k,i}$ sono definite in base al modello che si prende in considerazione.
Quello che è interessante in questi sistemi è studiare e capire come le particelle si muovono nel tempo all'interno di un dominio considerato; perciò l'equazione precedente può essere riscritta nella seguente forma finale:

$$\mathbf{v}_i(t + 1) = \mathbf{v}_i(t) + \mathbf{f}_i \tag{4}$$

Di seguito verrà mostrato come diverse scelte sulle forze determineranno i vari modelli a cui fa riferimento l'algoritmo di ottimizzazione di particelle.

1.1 Origine e background

Il primo modello semplice, fonte diretta dell'algoritmo PSO, è il modello *Boids* di Reynolds del 1987 [Rey87]. Il modello cerca di "replicare" il comportamento degli stormi di uccelli, specificando il movimento di ogni singolo componente, che interagisce con gli altri. Ogni individuo dello stormo è rappresentato da un punto materiale, chiamato "*boid*". Ogni boid all'interno della simulazione viene descritto tramite un triangolo (per semplicità di visualizzazione) e la dinamica dei suoi movimenti segue le classiche leggi di Newton, in cui le forze in questo caso rappresenteranno gli stimoli dei singoli agenti. Questi stimoli si basano su tre regole geometriche (Fig. 1):

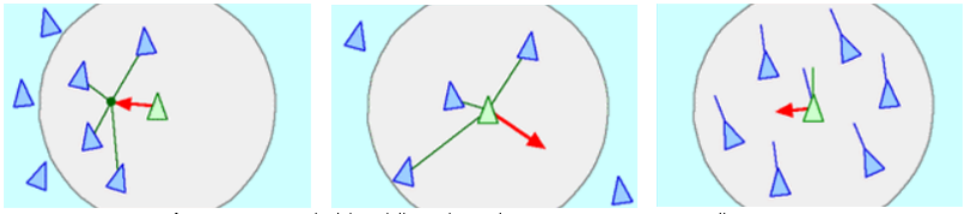


Figura 1: Le tre regole del modello di Boid, in ordine: coesione, separazione e allineamento

- **Coesione**: regola per la quale il singolo volatile tende a "sterzare" (o virare) verso il punto di massa medio del suo intorno metrico.
- **Separazione**: capacità dell'agente di evitare ostacoli nell'ambiente e/o altri agenti durante il "volo simulato".
- **Allineamento**: ogni agente cercherà di adattare la propria velocità (direzione, verso e modulo) a quella degli altri componenti dello stormo.

Indicando questi tre stimoli per un i-esimo boid, rispettivamente, $\mathbf{F}_{i,c}$, $\mathbf{F}_{i,s}$ e $\mathbf{F}_{i,a}$, la dinamica del modello può essere rappresentata tramite la formula (4), dove la componente $\mathbf{f}_i$, che si riferisce alle forze agenti sul sistema, diventa:

$$\mathbf{f}_i = \sum_{k=1}^{K} \mathbf{F}_k, i = \mathbf{F}_{i,c} + \mathbf{F}_{i,s} + \mathbf{F}_{i,a} \tag{5}$$

Inoltre il movimento, l'avvicinarsi o l'allontanarsi da compagni di volo e/o oggetti sono basati sulla distanza geometrica tra due punti. Il primo è il baricentro del volatile ed il secondo invece può essere il baricentro di un altro volatile in prossimità, piuttosto che il punto tangente dell'oggetto definibile come ostacolo. Il floccaggio richiede che il boid reagisca solo con i compagni all'interno di un certo piccolo intorno attorno a sè. Il vicinato è quindi caratterizzato da una distanza (misurata dal centro del boid) e da un angolo, misurato dalla direzione di volo del boid. I compagni di gruppo fuori da questo intorno locale vengono ignorati (Fig. 2).

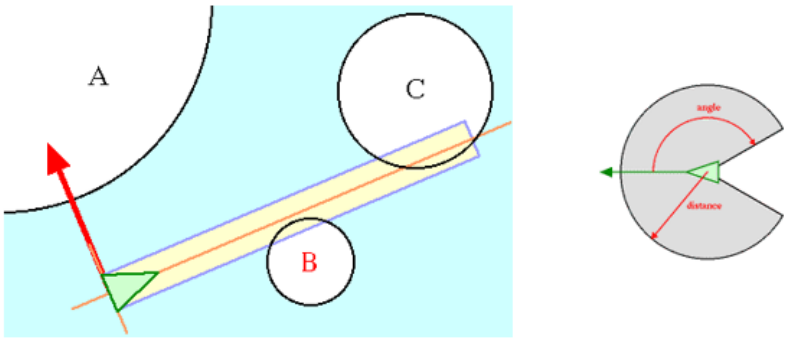


Figura 2: Distanza geometrica per un boid (a sinistra) e intorno del boid (a destra)

Tali regole definiscono un semplice comportamento individuale, considerando i soli tre stati, ma a livello globale queste semplici regole sono sufficienti a simulare il comportamento di stormi di uccelli. Una nota interessante sta nel fatto che le implementazioni base del modello non includono nessun elemento casuale nel comportamento; nonostante ciò è ugualmente possibile osservare comportamenti collettivi apparentemente caotici a partire da uno stato iniziale di posizionamento (e direzione) casuale. Infatti, come notato da Reynolds [Rey87], gli stormi non si comportano linearmente, ossia le direzioni di due stormi che si intersecano non si sommano ma avvengono dei comportamenti apparentemente caotici quando ciò accade.

Successivamente Heppner (1988), propose un modello noto come "*modello di campo di grano*", per simulare il comportamento di foraggiamento di uno stormo di uccelli.
Dal punto di vista matematico, considerando un piano bidimensionale XY, per ogni agente i, la posizione ad un istante generico t può essere definita con le coordinate $(x_i(t), y_i(t))$ con $i = 1, 2, ..., M$, dove M è il numero di agenti dello stormo.
Indicando con (x_{target}, y_{target}) la posizione del campo di grano, è possibile valutare, durante le simulazioni, la posizione attuale dell'agente i rispetto al campo di grano, come una semplice distanza euclidea:

$$d = \sqrt{\Big(x_i(t) - x_{target}(t)\Big)^2 + \Big(y_i(t) - y_{target}(t)\Big)^2} \tag{6}$$

Il movimento, nel dominio considerato, dell'agente i, per poter raggiungere il punto desiderato e quindi fare in modo che la distanza descritta sopra sia pari a zero, viene regolato tenendo conto di due fattori:

1. l'abilità di "ricordare" la miglior posizione personale raggiunta, che viene indicata con le coordinate $(p_{best}x, p_{best}y)$;

2. la posizione migliore raggiunta dall'intero stormo, rappresentata con le coordinate $(g_{best}x, g_{best}y)$.

In particolare:

- Se la componente rispetto a X della posizione dell'i-esimo uccello si trova sulla destra rispetto alla posizione del miglior valore trovato, allora la componente lungo X della velocità, v_x, viene regolata negativamente da una quantità casuale $r_1 \in [0, 1]$ ponderata con un parametro di sistema a:

$$v_x = v_x - r_1 a \quad se \;\; x_i > p_{best}x$$

  altrimenti, se si trova sulla sinistra, la componente causale viene sommata alla componente velocità:

$$v_x = v_x + r_1 a \quad se \;\; x_i < p_{best}x$$

- Allo stesso modo si ragiona per la componente lungo Y della velocità, v_y, considerando se l'uccello si trova sopra o sotto rispetto alla posizione del miglior valore trovato e indicando con $r_2 \in [0, 1]$ un numero casuale; perciò:

$$v_y = v_y - r_2 a \quad se \;\; y_i > p_{best}y$$

$$v_y = v_y + r_2 a \quad se \;\; y_i < p_{best}y$$

- In seguito viene fatto anche un controllo rispetto alla posizione migliore raggiunta dall'intero stormo, parametro memorizzato nella coppia di coordinate $(g_{best}x, g_{best}y)$. Perciò al calcolo precedente, ne segue un altro che aggiorna così le componenti della velocità, utilizzando l'analogo ragionamento:

$$v_x = v_x - r_1 b \quad se \;\; x_i > g_{best}x$$

$$v_x = v_x + r_1 b \quad se \;\; x_i < g_{best}x$$

$$v_y = v_y - r_2 b \quad se \;\; y_i > g_{best}y$$

$$v_y = v_y + r_2 b \quad se \;\; y_i < g_{best}y$$

  dove r_1 e r_2 sono sempre due numeri casuali appartenenti all'intervallo chiuso [0,1], mentre b è una costante di regolazione della velocità.

Il modello di Heppner considera quindi che ogni agente si muova sempre di un certo passo lungo ogni dimensione del dominio, a o b, moltiplicandoli per una certa quantità casuale. La direzione di questo movimento si ottiene dal calcolo del gradiente della distanza dal campo di grano, definita nella formula (6), e in particolare, dal seguente versore:

$$\hat{d} = \frac{\nabla d}{|d|} \tag{7}$$

Considerando allora la legge di Newton discretizzata (4), si può definire l'equazione differenziale come un problema del primo ordine, semplicemente impostando che lo stimolo di ogni agente sia quello di avvicinarsi al punto di dominio in cui c'è il campo di grano. In questo modello non ci sono quindi delle forze che agiscono sui singoli agenti, ma quello che si considera è solo la distanza di ognuno dal campo di grano:

$$\mathbf{f}_i = \sum_{k=1}^{K} \mathbf{F}_{k,i} = \mathbf{F}_{i,O} \tag{8}$$

dove $\mathbf{F}_{i,O}$ rappresenta la distanza dell'agente i-esimo dal punto in cui si troverà il cibo,tenendo conto della formula (7).

E' proprio a partire dalle leggi della dinamica e dalle simulazioni svolte da Reynolds prima e da Heppner successivamente, che Kennedy ed Eberhart nel 1995 idearono un algoritmo di ottimizzazione di particelle, anche noto come algoritmo PSO originale.

1.2 Algoritmo PSO originale

Nel PSO un numero di entità semplici, le particelle, sono collocate in un dominio spaziale, ognuna delle quali valuta la funzione obiettivo $f(\mathbf{x})$ nella sua posizione corrente. Ogni particella determina quindi il suo movimento attraverso lo spazio di ricerca combinando alcuni aspetti della storia delle proprie posizioni attuali e migliori con quelle degli altri membri dello sciame, con alcune perturbazioni casuali. Il movimento successivo avviene dopo che tutte le particelle si sono spostate. Alla fine lo sciame nel suo complesso è probabile che si sposti vicino ad un ottimo della funzione obiettivo.
In un sistema di coordinate dello spazio continuo, matematicamente, il PSO può essere descritto come segue.
Indicando con T_{max} il tempo massimo entro il quale le particelle dovranno spostarsi verso l'ottimo globale, si applica il concetto di "*iterazione*" $t = 0, 1, ..., T_{max}$; in questo modo ogni individuo nello sciame di particelle può essere rappresentato da tre vettori D-dimensionali, dove D è la dimensione del dominio:

1. la posizione corrente $\mathbf{x}_i(t)$=($x_{i,1}(t)$, $x_{i,2}(t)$, ..., $x_{i,D}(t)$)
2. la precedente miglior posizione $\mathbf{p}_i(t)$=($p_{i,1}(t)$, $p_{i,2}(t)$, ..., $p_{i,D}$) = arg $min_{\tau \leq t}\mathbf{x}_i(\tau)$, dove:

   $$arg\ min_{\tau \leq t}\mathbf{x}_i(\tau) = \mathbf{y}_i$$

   tale che:

   $$f(\mathbf{y}_i) = min_{\tau \leq t} f(\mathbf{x}_i(\tau))$$

3. la velocità $\mathbf{v}_i(t)$=($v_{i,1}(t)$, $v_{i,2}(t)$, ..., $v_{i,D}(t)$)

dove $i \in \mathbb{N} = \{1, 2, ..., N\}$ rappresenta ogni singolo individuo dello sciame.

Ad ogni iterazione t dell'algoritmo, la posizione corrente $\mathbf{x}_i(t)$ dell'i-esima particella viene valutata come configurazione di un problema. Se quella posizione è migliore rispetto a qualsiasi altra posizione trovata fino a quel momento dalla particella stessa, allora le coordinate sono memorizzate nel secondo vettore $\mathbf{p}_i(t)$.

Le particelle, oltre a ricordare la loro miglior posizione, hanno la capacità di comunicare con le altre particelle. La comunicazione può avvenire:

- per interazioni metriche: una particella comunica con tutte quelle che si trovano in un intorno ben definito, a prescindere da chi esse siano;
- topologicamente: le particelle comunicano tra di loro solo secondo una determinata struttura, che viene decisa a priori.

Tenuto conto di ciò, la particella, comunicando con le altre, è influenzata dalle loro esperienze e, soprattutto, dalla posizione migliore trovata tra tutte quelle delle particelle in quell'intorno, indicata con $\mathbf{P}_i^s(t)$:

$$\mathbf{P}_i^s(t) = arg\ min_{j \in S_i} f(\mathbf{x}_j(t)) \tag{9}$$

dove

$$S_i = \{j = 1, ..., M \text{ tale che } j \text{ comunica con } i\}$$

Se l'intorno S_i è topologico, questo varierà a seconda della topologia scelta; in particolare le topologie si distinguono in due classi:

1. globale, in cui ogni particella comunica con l'intero sciame;
2. locale, in cui ogni particella comunica con un suo particolare intorno.

In letteratura sono diverse le topologie rappresentate a partire da questi due concetti e verranno affrontate nel dettaglio nel successivo paragrafo.

Nel processo di ottimizzazione dello sciame di particelle, la velocità di ogni particella viene regolata iterativamente in modo che la particella stocasticamente oscilli intorno alle posizioni di $\mathbf{p}_i$ e $\mathbf{P}_i^s$ (Fig. 3).

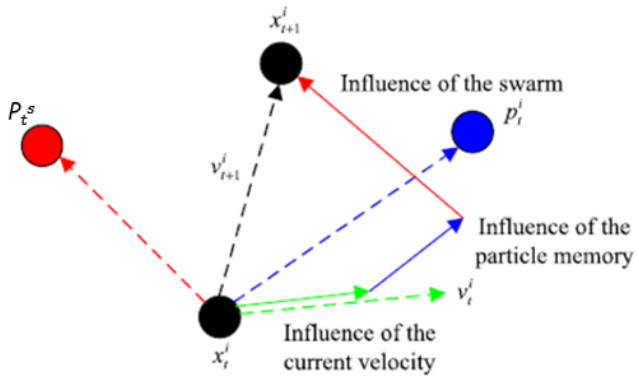


Figura 3: Schema di iterazione delle particelle

Lo spostamento che avviene per una particella i passando dalla posizione $\mathbf{x}_i(t)$ a $\mathbf{x}_i(t+1)$ può infatti essere scomposta in tre movimenti, seguendo la figura di cui sopra:

1. Il primo passo è quello di spostarsi di una certa quantità nella stessa direzione e nello stesso verso del vettore velocità $\mathbf{v}_i(t)$;

2. Successivamente la particella è influenzata dalla sua miglior posizione e quindi si sposta di una certa quantità seguendo la direzione e il verso del vettore che collega la particella stessa con la sua miglior posizione $\mathbf{p}_i$;

3. Il ragionamento è poi analogo al precedente considerando la posizione migliore trovata tra tutte le particelle dell'intorno S_i, quindi seguendo direzione e verso del vettore distanza tra la posizione della particella i e la miglior posizione $\mathbf{P}_i^s$

La somma di questi tre movimenti può essere anche riscritta nel seguente modo:

$$\mathbf{x}_i(t+1) = \mathbf{x}_i(t) + \mathbf{v}_i(t+1) \tag{10}$$

dove $\mathbf{v}_i(t+1)$ è proprio il vettore che guida il processo di ottimizzazione e riflette sia l'esperienza personale sia quella acquisita dalle interazioni sociali. Questi due fattori vengono rispettivamente chiamati componente cognitiva e componente sociale.

Tenendo conto di ciò, se si considera allora un problema di minimizzazione, invece di andare a controllare ogni volta le posizioni occupate in passato, $\mathbf{p}_i(t)$ può essere così calcolata:

$$\mathbf{p}_i(t+1) = \begin{cases} \mathbf{p}_i(t) & se\ f\left(\mathbf{x}_i(t+1)\right) \geq f\left(\mathbf{p}_i(t)\right) \\ \mathbf{x}_i(t+1) & se\ f\left(\mathbf{x}_i(t+1)\right) < f\left(\mathbf{p}_i(t)\right) \end{cases} \tag{11}$$

Per poter definire $\mathbf{v}_i(t+1)$, invece, è necessario ricorrere alle leggi della dinamica, ma prima è importante introdurre alcuni concetti, essenziali per capire la logica del PSO.
Dal punto di vista matematico, per definire il movimento delle particelle, partendo dalle leggi della dinamica, si considerano come forze i due stimoli, descritti sopra, che agiscono su ogni particella:

1. una forza $F_{i,1}$ che viene definita "*conoscenza personale*": questa nasce dal fatto che ogni particella è in grado di memorizzare la propria miglior posizione tra tutte quelle visitate;

2. una forza $F_{i,2}$, definita "*conoscenza globale*": ogni particella comunica con un certo numero di particelle secondo una struttura di comunicazione ed è quindi in grado di conoscere quali sono le posizioni migliori visitate dalle altre particelle.

Perciò, data una particella i, i due stimoli possono essere così definiti:

$$\begin{aligned} \mathbf{F}_{i,1} &= \mathbf{U}(0, \phi_1) \otimes (\mathbf{p}_i - \mathbf{x}_i) \\ \mathbf{F}_{i,2} &= \mathbf{U}(0, \phi_2) \otimes (\mathbf{P}_i^s - \mathbf{x}_i) \end{aligned} \tag{12}$$

dove $\mathbf{U}(0, \phi_i)$ rappresenta un vettore di numeri casuali distribuiti uniformemente in $[0, \phi_i]$, che viene generato casualmente ad ogni iterazione e per ogni particella; mentre il simbolo $\otimes$ rappresenta il concetto di *moltiplicazione per componente*: per definizione, dati due vettori generici $\mathbf{a}$ e $\mathbf{b}$, una moltiplicazione per componente è la seguente operazione:

$$\mathbf{a} \otimes \mathbf{b} = (a_1 b_1, ..., a_n b_n)$$

Partendo dalla legge di Newton (4), la formula di aggiornamento della velocità diventa quindi:

$$\mathbf{v}_i(t+1) = \mathbf{v}_i(t) + \mathbf{f}_i \tag{13}$$

con:

$$\mathbf{f}_i = \sum_{k=1}^{2} \mathbf{F}_{i,k} = \mathbf{F}_{i,1} + \mathbf{F}_{i,2}$$

Lo sciame di particelle è quindi più di una semplice raccolta di particelle. Una particella di per sé non ha quasi nessun potere di risolvere alcun problema; il progresso si verifica solo quando le particelle interagiscono (Fig. 4).

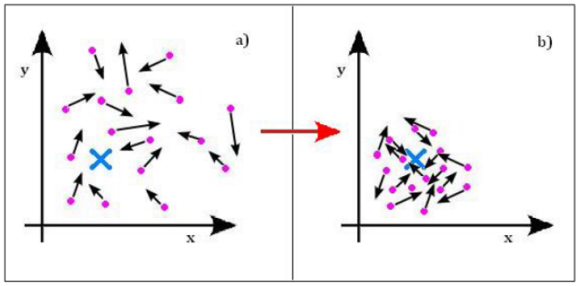


Figura 4: Esempio bidimensionale del movimento delle particelle: nella a) inizia il movimento, che via via tende a far spostare le particelle verso la posizione ottima all'interno dello spazio, come in b).

E' inoltre possibile analizzare ogni singola componente della formula della velocità (13), rappresentandola come segue:

$$v_{i,d}(t+1) = v_{i,d}(t) + c_1 r_{1,d}(t)\left(p_{i,d}(t) - x_{i,d}(t)\right) + c_2 r_{2,d}(t)\left(P^s_{i,d}(t) - x_{i,d}(t)\right) \tag{14}$$

dove:

1. $v_{i,d}(t)$ è la d-esima componente del vettore $\mathbf{v}_i$ all'iterazione t;
2. $x_{i,d}(t)$ è la componente del vettore posizione in corrispondenza della particella i nella dimensione d all'iterazione t, con $i = 1, 2, ..., N$, riferito alla i-esima particella; $d = 1, 2, ..., D$, riferito alla d-esima dimensione considerata e $t = 0, 1, 2, ..., T_{max}$;
3. c_1 e c_2 sono costanti positive, chiamate coefficienti di accelerazione, utilizzate per scalare rispettivamente il contributo della componente cognitiva e sociale; $r_{1,d}(t)$ e $r_{2,d}(t)$ sono sequenze casuali nell'intervallo [0,1), che definiscono la natura stocastica dell'algoritmo. Le componenti, cognitiva e sociale, rispettivamente indicate con ϕ_1 e ϕ_2 nella formula (12), sono così definite:

$$\phi_1 = c_1 r_1$$
$$\phi_2 = c_2 r_2$$

In particolare, il calcolo della velocità consiste nella somma di tre termini:

- La velocità precedente $v_{i,d}(t)$ serve per ricordare alla particella la direzione del suo movimento. Questo significa che la particella ha fiducia nel suo stato di movimento corrente e conduce un movimento inerziale secondo la propria velocità. Kennedy e Eberhart chiamano questa componente *inerzia*.

- La componente cognitiva $c_1 r_{1,d}(t)\,(p_{i,d}(t) - x_{i,d}(t))$ è il "ricordo" della miglior posizione raggiunta. Questo termine è dato dall'attrazione di ogni particella i verso l'ottimo trovato da ciascuna, in accordo con la predisposizione degli individui a tornare nei luoghi e nelle situazioni che più li hanno soddisfatti. Kennedy e Eberhart indicano questa componente cognitiva con il termine di "*nostalgia*".

- Infine la componente sociale, $c_2 r_{2,j}^{(\tau)}(p_{s,j}^{(\tau)} - x_{i,j}^{(\tau)})$ quantifica il successo della particella i in relazione all'intero gruppo con cui comunica. Concettualmente la componente sociale rappresenta una norma o uno standard a cui gli individui tendono ad attenersi. L'effetto di questo termine è l'attrazione delle particelle verso la miglior posizione trovata dall'intero sciame.

L'algoritmo PSO appena descritto prevede quindi in sintesi i seguenti passaggi:

1. Inizializzare la popolazione, tramite una matrice di particelle con posizione e velocità casuali nel dominio di dimensioni D e valutare la funzione obiettivo f nelle posizioni iniziali e impostare le miglior posizioni precedenti $\mathbf{p}_i$ uguali alle posizioni iniziali stesse.

2. Iniziare un ciclo, che continua fino a quando il criterio di arresto scelto non è soddisfatto:

   a. Aggiornare posizione e velocità delle particelle secondo le equazioni (10) e (13).

   b. Se $f(\mathbf{x}_i(t)) \leq f(\mathbf{p}_i(t-1))$, allora si imposta $\mathbf{p}_i(t) = \mathbf{x}_i(t)$. In questo punto il valore della funzione obiettivo alla t-esima iterazione viene valutato con il valore della funzione migliore trovata fino a quel momento. Se il valore corrente della funzione è migliore, allora la posizione della particella associata a tale valore viene settata come la nuova migliore posizione della i-esima particella.

   c. Identificare la particella nell'intorno topologico con la migliore posizione e identificandola con l'indice s, ossia $\mathbf{P}_i^S(t)$.

   d. Se si soddisfa uno dei criteri (ad esempio, la funzione obiettivo sufficientemente buona o un numero massimo di iterazioni), uscire dal ciclo continuo, altrimenti tornare al punto a.

3. Fine ciclo.

Un diagramma di flusso dell'algoritmo PSO è mostrato in Figura 5.

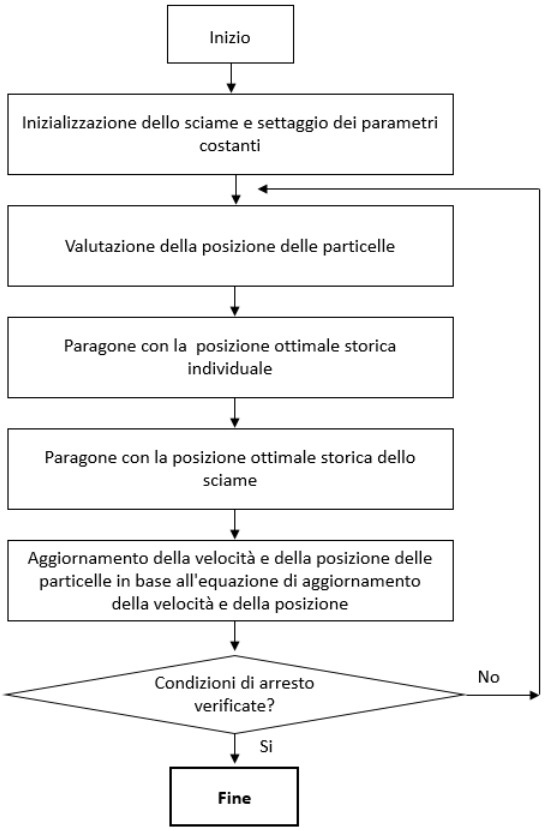


Figura 5: Flow chart per un algoritmo a sciame

1.3 Strutture di comunicazione

La chiave che guida il PSO è l'*interazione sociale*. Le particelle all'interno dello sciame comunicano le une con le altre e, sulla base della conoscenza acquisita, si muovono per seguire le loro compagne "migliori". Questo comportamento è tipico del mondo animale, dove un organismo è influenzato dai simili che lo circondano e dove gli individui meno promettenti seguono il modello di quelli con più "successo".
Come già precedentemente accennato, le topologie si dividono principalmente in due classi, che a loro volta danno vita a due versioni dell'algoritmo PSO: il *global best PSO*, in cui tutte le particelle comunicano tra loro, e il *local best PSO* in cui ogni particella comunica con un suo intorno prefissato. Quest'ultimo in particolare consiste di diversi casi in base alla topologia di comunicazione che viene scelta per l'algoritmo.
A partire da una qualsiasi particella i dello sciame, è possibile individuare un insieme S_i di particelle con cui questa può comunicare. All'interno di uno sciame, gli insiemi considerati possono essere tanti quante sono le particelle, ma se si considera l'unione di tutti questi insiemi, questa restituirà lo sciame stesso.

1.3.1 Global Best PSO

L'algoritmo PSO viene definito "*globale*" quando ogni particella comunica con tutte le altre, quindi per ogni particella i, l'insieme S_i coincide con lo sciame nel suo complesso. Dal punto di vista matematico, questo concetto si traduce nella definizione di $\mathbf{P}_i^S$, presente nella formula di aggiornamento della velocità (13), che può essere indicato con $\mathbf{P}^g$, per differenziarlo dal caso locale, ed è così calcolato:

$$\mathbf{P}^g(t) = arg\ min_{\mathbf{x}\in\{p_1,...,p_n\}} f(\mathbf{x}_i(t)) \tag{15}$$

In questo caso ogni particella, comunicando con ogni altra all'interno dello sciame, è attratta verso la miglior posizione trovata da tutte. Questa migliore posizione è il valore che la funzione obiettivo $f(\mathbf{x})$ assume in corrispondenza di $\mathbf{P}^g$ ed è memorizzato in una variabile denominata *global best*. Grazie all'interconnessione tra tutte le particelle, questo algoritmo converge più velocemente, ma questo si paga in termini di minore esplorazione.
La topologia legata a questo algoritmo viene definita "struttura sociale a stella" (anche nota come *gBest*), dove tutte le particelle sono interconnesse come in Figura (6):

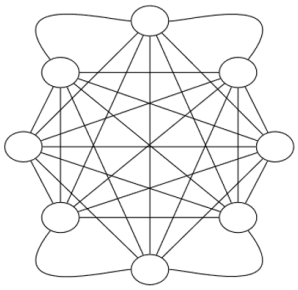


Figura 6: Topologia a stella o *gBest*

1.3.2 Local Best PSO e classificazione delle topologie

Rispetto all'algoritmo globale, in quello "*locale*" l'intorno può variare in base alla topologia che viene scelta. In questo caso le particelle non comunicano tutte tra di loro, ma scambiano informazioni solo con un certo insieme. La formula di aggiornamento della velocità (13) avrà perciò un termine $\mathbf{P}_i^s$, indicato nel caso locale con $\mathbf{P}^l$, che è così calcolato:

$$\mathbf{P}^l(t) = \arg min_{\mathbf{x}\in\aleph_i} f(\mathbf{x}) \tag{16}$$

con $\aleph_i = \{\mathbf{p}_{i-j}(t), \mathbf{p}_{i-j+1}(t), ..., \mathbf{p}_{i-1}(t), \mathbf{p}_i(t), \mathbf{p}_{i+1}(t), ..., \mathbf{p}_{i+j}(t)\}$, che raccoglie tutte le migliori posizioni trovate dalle $j \in \mathbb{N}$ particelle del sottoinsieme.
Nonostante esistano strategie basate sulla similarità spaziale, i vicini di ogni particella vengono tipicamente scelti in base ai loro indici. Le ragioni che spingono a far questo sono essenzialmente due:

1. Si riduce la complessità computazionale, poiché non è necessario calcolare la distanza euclidea tra tutte le coppie di particelle.

2. Si favorisce la diffusione dell'informazione a tutte le particelle, indipendentemente dalla loro posizione nello spazio di ricerca.

Bisogna anche notare che i sottoinsiemi di comunicazione si sovrappongono. Una particella può essere membro di diversi sottoinsiemi favorendo l'interconnessione e lo scambio d'informazione tra questi. Questo consente allo sciame di convergere verso un unico punto che corrisponde all'ottimo globale del problema.

Tenendo conto di quanto detto sopra, è possibile distinguere differenti topologie [LSO21]:

- struttura sociale ad anello (anche nota come *lBest*), dove ogni particella può comunicare con $n = 2$ suoi vicini adiacenti nella matrice della popolazione, con avvolgimento toroidale (Fig. 7(a)). E' possibile generalizzare il concetto a $n > 2$: in questo caso le particelle sono attratte e si muovono nella direzione della "migliore" del gruppo considerato (Fig. 7(b)).

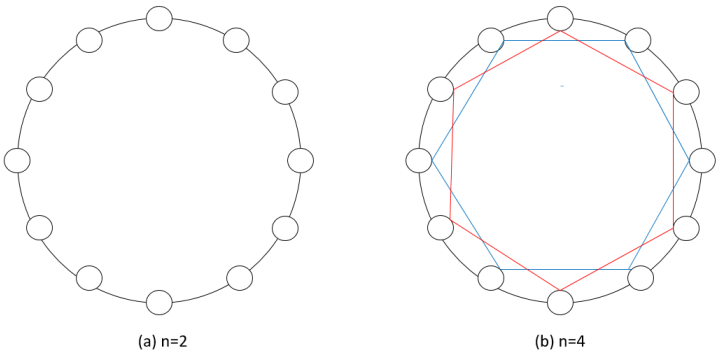


Figura 7: Esempi di topologia ad anello: a sinistra il caso in cui le particelle comunicano solo $n = 2$ vicini; a destra il caso $n = 4$ vicini.

- struttura sociale a ruota, dove gli individui in un gruppo sono isolati l'uno dall'altro. In questo caso una particella funge da punto focale e tutta l'informazione viene scambiata attraverso questa (Fig. 8(a)). La particella "focale" confronta le posizioni di tutte le particelle nel gruppo e si muove verso la migliore di queste. Se la nuova posizione ha un grado di ottimalità maggiore della precedente, il successo viene comunicato a tutti i membri del gruppo. La struttura a ruota rallenta la propagazione delle soluzioni migliori nello sciame.

- struttura sociale di Von Neumann, struttura a griglia dove ogni particella è connessa ai suoi quattro vicini: in alto, in basso, a destra e a sinistra (Fig. 8(b)).

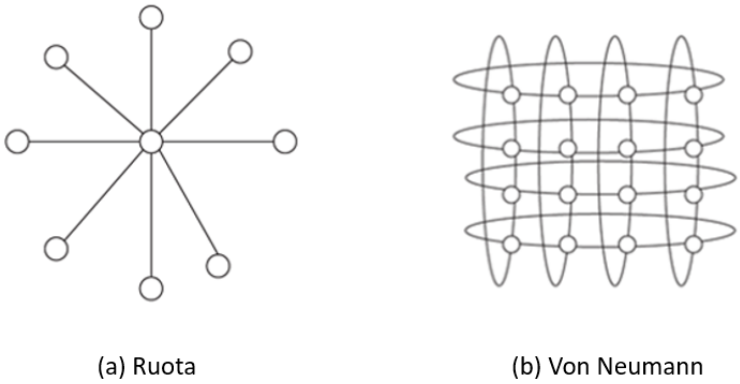


Figura 8: Strutture di comunicazione: ruota e Von Neumann

Da queste strutture è anche possibile creare delle topologie miste, come ad esempio:

1. topologia a cluster, dove ci sono n sotto-gruppi. Per ogni sotto-gruppo, si possono scegliere diverse topologie tra quelle elencate in precedenza per far comunicare le particelle tra loro e alla fine i vari gruppi si scambieranno informazioni soltanto tramite un numero limitato di particelle ciascuno.
   Ad esempio, in figura (9a), all'interno di ogni sotto-gruppo le particelle comunicano tra loro con una struttura di comunicazione *gBest*, diffondendo *globalmente* l'informazione. Le informazioni infine "viaggiano" tra i vari gruppi attraverso tre sole particelle.
   Un altro esempio è quello mostrato in figura (9b), dove una sola particella riceve informazioni da tutte quelle del proprio gruppo (attraverso una topologia a ruota) e i diversi gruppi comunicano tra loro solo tramite queste particelle "principali".

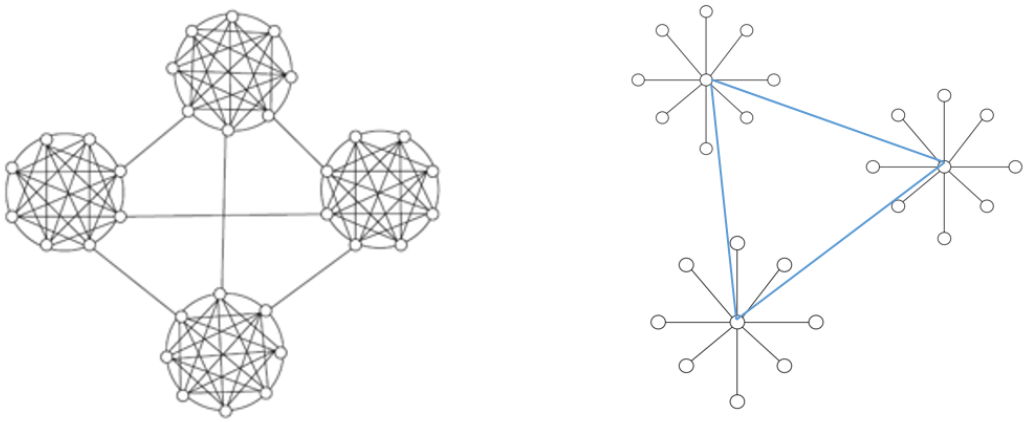


Figura 9: Diverse topologie a cluster

2. topologia multi-anello, basata sull'unione di diversi strati (o livelli) di anelli. Dati n livelli, ognuno di essi ha lo stesso numero di particelle, le quali adottano una struttura di comunicazione di Von Neumann per condividere informazioni tra i vari livelli, eccetto le particelle nel primo livello che non comunicano con il livello finale e viceversa (Fig. 10).

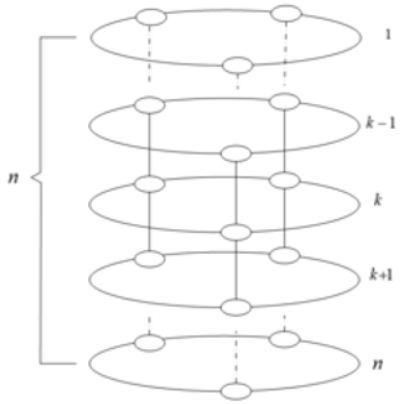


Figura 10: Esempio di topologia multi-anello

Viene poi aggiunta un' abilità di rotazione per ridurre la possibilità di cadere in un ottimo locale. Pertanto, se un livello non migliora la propria posizione migliore durante le iterazioni, verrà ruotato (Fig. 11).

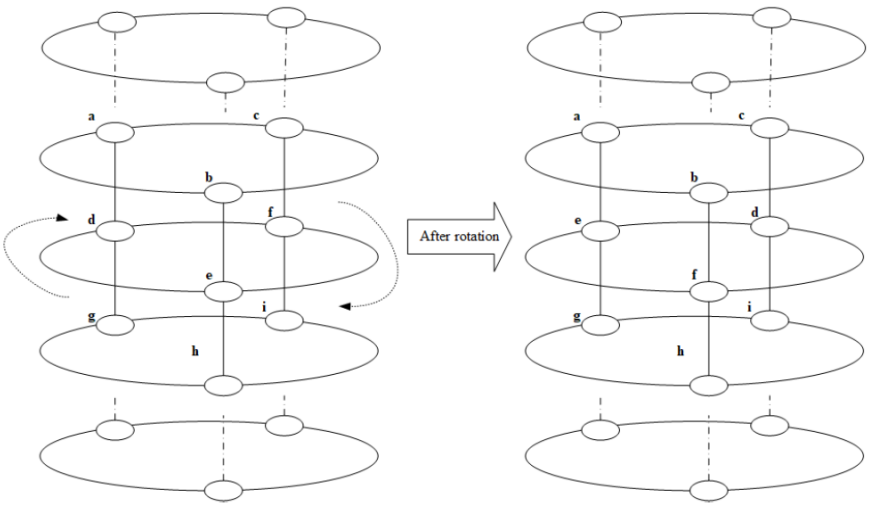


Figura 11: Esempio di rotazione di un anello

1.3.3 Topologie dinamiche

Col tempo e dopo successivi studi sono state introdotte anche delle *topologie dinamiche*, ovvero che si "adattano" durante il processo di iterazione. In particolare:

- Suganthan (1999) propose di iniziare la ricerca con una topologia *lbest* con struttura ad anello e incrementare lentamente con l'aumentare del numero di iterazioni la dimensione del vicinato fino ad avere una topologia *gbest* alla fine delle iterazioni. Il motivo di questa scelta sta nel fatto di poter sfruttare inizialmente la miglior esplorazione della topologia *lbest*, e solo successivamente sfruttare invece la veloce convergenza della topologia *gbest*.

- Peram et al. (2003) [PVM03] hanno introdotto un diverso algoritmo PSO (chiamato *FDR-PSO*), in cui ogni particella, oltre a basarsi sulla propria esperienza e sull'esperienza della particella di maggior successo, impara ed è influenzata anche dall'esperienza delle particelle vicine che hanno trovato posizioni migliori di sé stessa. L'algoritmo seleziona solo un'altra particella alla volta, rispetto all'i-esima considerata, durante l'aggiornamento di ciascuna dimensione della velocità. I criteri per scegliere la particella sono:

  1. deve essere vicino alla particella in fase di aggiornamento
  2. deve aver visitato una posizione di maggiore ottimalità

  Questo approccio si traduce in cambiamenti nelle equazioni di aggiornamento della velocità (17), sebbene le equazioni di aggiornamento di posizione rimangano invariate, ossia:

$$\begin{aligned} \mathbf{x}_i(t+1) &= \mathbf{x}_i(t) + \mathbf{v}_i(t+1) \\ \mathbf{v}_i(t+1) &= \mathbf{v}_i(t) + \mathbf{U}(0,\phi_1) \otimes (\mathbf{p}_i(t) - \mathbf{x}_i(t)) + \mathbf{U}(0,\phi_2) \otimes (\mathbf{P}_i^s(t) - \mathbf{x}_i(t)) + \\ &\qquad \mathbf{U}(0,\phi_3) \otimes (\mathbf{p}_i^n(t) - \mathbf{x}_i(t)) \end{aligned} \tag{17}$$

  dove $\mathbf{p}_i^n(t)$ viene scelto minimizzando (o massimizzando nei problemi di massimizzazione) il seguente rapporto FDR:

$$FDR = \frac{f(\bar{\mathbf{p}}) - f(\mathbf{x}_i)}{|\bar{p}_d - x_{i,d}|} \tag{18}$$

  In altre parole la d-esima dimensione della posizione della particella i è aggiornata usando una particella, chiamata *nbest*, che ha come migliore posizione $\bar{\mathbf{p}}$; dove $|...|$ denota il valore assoluto e indica la distanza tra la particella *nbest* e l'i-esima considerata nell'iterazione corrente.
  Per questo motivo l'equazione di aggiornamento della velocità è influenzata da tre differenti fattori:

  1. La precedente migliore esperienza della particella;
  2. La migliore esperienza data dall'intorno che comunica con la particella;
  3. La precedente migliore esperienza del vicino *nbest* scelto con i precedenti criteri.

  L'espressione sopra descritta è definita *Fitness-Distance-Ratio*, suggerendo il nome dell'algoritmo FDR-PSO.

- Clerc (2006) introdusse il sistema di sciami di particelle, privo di parametri, chiamato *TRIBES*, in cui i dettagli della topologia si evolvono nel tempo in risposta al feedback delle prestazioni. La popolazione è divisa in sottopopolazioni, ciascuna mantenendo il proprio ordine e la propria struttura. "Buone" tribù possono trarre vantaggio dalla rimozione dei loro membri più deboli, poiché loro possiedono già buone soluzioni al problema e quindi possono permettersi di ridurre la loro popolazione; "cattive" tribù, dall'altra parte, possono trarre vantaggio dall'aggiunta di un nuovo membro, aumentando la possibilità di miglioramento. Nuove particelle vengono generate casualmente e si rivaluta e modifica la struttura della popolazione ogni L/2 iterazioni, dove L è il numero di collegamenti nella popolazione.

- Mendes (2004) ha presentato l'algoritmo dello sciame di particelle completamente informato (*FIPS*), che utilizzava le informazioni dell'intero vicinato per guidare le particelle a trovare la soluzione migliore [Men04]. L'influenza di ciascuna particella sul suo vicino è stata ponderata dal suo valore di fitness e dalle dimensioni del vicinato. A tal proposito, è stato notato che l'effetto della topologia della popolazione era completamente differente in FIPS e nella versione originale ("miglior vicino").
  Ad esempio, mentre la topologia gbest nel PSO originale significava che ogni individuo riceveva le migliori informazioni note a qualsiasi membro della popolazione, nel FIPS significava che ogni particella riceveva informazioni riguardo alle migliori soluzioni trovate da tutti i membri della popolazione; quindi, il comportamento gbest potrebbe essere descritto come quasi casuale, poiché le particelle sono influenzate da numerosi e diversi attrattori. Mendes (2004) notò anche che la comprensione degli effetti della topologia sulle prestazioni richiede di prendere in considerazione il modo in cui la particella interagisce con il suo vicinato e quale misura di prestazione deve essere utilizzata.

2 Algoritmo proposto

Fin da subito l'algoritmo PSO originale ha dimostrato di non essere molto efficiente nei problemi di ottimizzazione. Negli anni successivi sono state infatti diverse le varianti presentate con lo scopo di miglioralo. L'obiettivo di questa tesi è quello di implementare non la versione originale bensì una delle varianti che verranno presentate di seguito. Inoltre verranno definiti i parametri che descrivono l'algoritmo, come ad esempio la dimensione dello sciame e i fattori di apprendimento, ma anche gli aspetti essenziali quali l'inizializzazione, il confinamento delle particelle e i criteri di arresto, indicando infine quelle che sono state le scelte implementative applicate all'algoritmo selezionato.
Le varianti principali sono: l'algoritmo PSO canonico (anche noto come *Inertia Weight*) e il *Constriction Factor*.

2.1 Algoritmo PSO canonico

La prima variante al modello originale è quella presentata da Shi ed Eberhart nel 1998, la quale mira a bilanciare due possibili tendenze del PSO: quella di sfruttare l'intorno di soluzioni note e quella di esplorare nuove aree del dominio. A tale scopo, nella formula di aggiornamento della velocità (13), viene aggiunto un fattore che viene chiamato *peso inerziale* w:

$$\mathbf{v}_i(t+1) - \mathbf{v}_i(t) = \mathbf{f}_i - (1-w)\mathbf{v}_i(t) \tag{19}$$

dove $\mathbf{f}_i$ è la risultate delle forze esterne che agiscono sulle particelle, come già visto nella versione originale del PSO.
Si può facilmente constatare che la costante $(1-w)$ agisce efficacemente come un coefficiente d'attrito e w può essere interpretata come la fluidità del mezzo in cui la particella si muove. Osservando inoltre l'equazione (19) nella seguente forma:

$$\mathbf{v}_i(t+1) = w\mathbf{v}_i(t) + \mathbf{f}_i \tag{20}$$

si può notare come l'idea di questo nuovo algoritmo sia quella di dare un peso bilanciato all'inerzia del movimento, moltiplicando la componente che tiene memoria della velocità precedente $\mathbf{v}_i(t)$ per una costante w, cercando di dare così più importanza all'esplorazione di nuove aree rispetto a quanto accade nella versione base.

Questo nuovo algoritmo, noto come *algoritmo PSO canonico*, ha quasi la stessa complessità della versione iniziale, ma ha migliorato notevolmente le prestazioni dello stesso.

Il valore di w influenza fortemente la convergenza dell'algoritmo:

- se $w \geq 1$ le velocità aumentano e lo sciame diverge.
- se invece $w < 1$, le particelle "decelerano" finché la loro velocità non diventa nulla.

Shi ed Eberhart, attraverso uno studio sperimentale, notarono che introducendo un peso inerziale linearmente decrescente con il tempo, scegliendo come valore iniziale $w_{start} = 0,9$ e come valore finale $w_{end} = 0,4$, le prestazioni del PSO erano notevolmente migliorate.

In particolare i risultati sperimentali hanno mostrato che il PSO converge rapidamente verso le posizioni ottimali ma rallenta la sua velocità di convergenza quando è vicino agli ottimi. Pertanto, utilizzando il peso inerziale linearmente decrescente, il PSO manca della capacità di ricerca globale alla fine dell'esecuzione anche quando in alcuni casi è richiesta la capacità di ricerca globale per saltare fuori dal minimo locale. Di conseguenza, scegliendo a priori T_{max}, numero massimo di iterazioni, è stato suggerito l'utilizzo di una strategia di adattamento per regolare il peso inerziale migliorando le prestazioni del PSO vicino all'ottimo:

$$w(t) = (w_{start} - w_{end})\frac{(T_{max} - t)}{T_{max}} + w_{end} \tag{21}$$

dove w_{start} e w_{end} sono i valori iniziali e finali del peso inerziale, t è l'iterazione corrente; $w(t) \in [0, 1]$ è il valore del peso inerziale corrispondente all'iterazione t.

Nel 2006 Van Der Bergh ha notato che più w è piccolo più le componenti di accelerazione (sociale e cognitiva) influiscono sull'aggiornamento della posizione. Il valore ottimale di w dipende allora dal problema e va scelto in funzione dei coefficienti di accelerazione c_1 e c_2 [VdBE06]. Per garantire la convergenza, propose la seguente formula:

$$w > \frac{1}{2}(c_1 + c_2) - 1 \tag{22}$$

concludendo che nel caso in cui questa condizione non sia soddisfatta lo sciame potrebbe avere un comportamento divergente o ciclico.

Un'altra opzione proposta è stata quella di farlo variare in maniera casuale; in questo caso in ogni iterazione viene scelto un fattore diverso, campionandolo da una distribuzione gaussiana:

$$w \sim N(0.72, \sigma) \tag{23}$$

dove σ è piccolo abbastanza da garantire che w non sia troppo più grande di 1.

2.2 Constriction Factor

Analizzando il comportamento di convergenza dell'algoritmo PSO originale, Clerc e Kennedy (2002) hanno introdotto una variante di PSO con il *fattore di costrizione* χ, che ha garantito la convergenza e ne ha migliorato il tasso [CK02].
L'equazione di aggiornamento della velocità può essere allora così riscritta:

$$\mathbf{v}_i(t+1) = \chi[\mathbf{v}_i(t) + \mathbf{f}_i] \tag{24}$$

Questo fattore χ, sulla base di simulazioni numeriche, viene calcolato nel seguente modo:

$$\chi = \frac{2\upsilon}{\left|2 - \phi - \sqrt{(\phi(\phi - 4))}\right|} \tag{25}$$

con $\phi = \phi_1 + \phi_2$, $\upsilon \in [0, 1]$, costante arbitraria, scelta a priori e ricordando che ϕ_1 e ϕ_2 vengono così calcolati:

$$\phi_1 = c_1 r_1$$
$$\phi_2 = c_2 r_2$$

Clerc e Kennedy, dopo vari studi, conclusero che per garantire convergenza i fattori di apprendimento c_1 e c_2 dovrebbero essere scelti in modo da far rispettare la condizione $\phi \geq 4$. In particolare mentre ϕ determina la grandezza delle forza casuali, v controlla l'ambito di ricerca globale e locale dello sciame.

Si può notare infine che un algoritmo PSO con costrizione è algebricamente equivalente ad un PSO con inerzia. Infatti le formule (19) e (24) possono essere trasformate l'una nell'altra tramite le seguenti mappature:

$$w \leftrightarrow \chi$$
$$\phi_i \leftrightarrow \chi\phi_i$$

2.3 Parametri e aspetti dell'algoritmo

Sia l'algoritmo PSO originale che le varianti presentano diversi parametri ed aspetti che non sono ancora stati discussi. Questi includono l'inizializzazione, le condizioni d'uscita e il confinamento delle particelle: tutte condizioni importanti per garantire la convergenza e il corretto funzionamento dell'algoritmo.

2.3.1 Inizializzazione

Il primo passo dell'algoritmo è la creazione dello sciame, inizializzando le particelle e i parametri di controllo. Le posizioni delle particelle sono inizializzate in modo che esse coprano uniformemente un dominio. Questo parametro è importante per l'efficienza del PSO, influenzata dalla diversità dello sciame iniziale. Sarà difficile trovare l'ottimo di un problema se questo cade in una regione non inizialmente coperta dalle particelle.

Si assuma allora che l'ottimo debba essere trovato all'interno di un dominio cubico Ω, definito da due vettori $\mathbf{x}_{min}$ e $\mathbf{x}_{max}$, che rappresentano rispettivamente il minimo e il massimo valore di ogni dimensione. Un buon modo per inizializzare la posizione delle particelle è di distribuirle in modo casuale ed uniforme in Ω.

Le velocità invece possono essere inizializzate a zero:

$$\mathbf{v}_i(0) = 0 \tag{26}$$

Sebbene sia possibile anche inizializzare le velocità a valori casuali, non è necessario e deve essere fatto con cura. Infatti, considerando le particelle nelle loro posizioni iniziali, le loro velocità sono zero: sono quindi stazionarie. Se le particelle vengono inizializzate con velocità diverse da zero, questa analogia fisica viene violata. L'inizializzazione casuale dei vettori di posizione garantisce già posizioni e direzioni di movimento casuali. Se, tuttavia, anche le velocità vengono inizializzate casualmente, tali velocità non dovrebbero essere troppo grandi, altrimenti c'è il forte rischio che le particelle lascino i confini del dominio e che diventi più difficile "fermarsi" su una soluzione ottima.

La migliore posizione trovata da ciascuna particella è invece inizializzata alla posizione iniziale:

$$\mathbf{p}_i(0) = \mathbf{x}_i(0) \tag{27}$$

2.3.2 Dimensione della popolazione

La dimensione della popolazione è spesso impostata empiricamente sulla base della difficoltà di un problema. I valori nell'intervallo 20-50 sono abbastanza comuni. La popolazione può aumentare nel caso in cui sia necessario soddisfare delle esigenze speciali.

2.3.3 Condizioni d'uscita

L'altro aspetto da considerare è la condizione di uscita, ovvero il criterio utilizzato per terminare il processo di ricerca. Per scegliere questa condizione vanno considerati due fattori:

1. La condizione di uscita non dovrebbe causare una convergenza prematura, con soluzioni sub-ottimali.

2. La condizione di uscita dovrebbe evitare valutazioni inutili della funzione obiettivo. Se il criterio di arresto richiede frequenti valutazioni della funzione obiettivo, la complessità del processo di ricerca potrebbe aumentare notevolmente.

Le condizioni di uscita più comuni sono le seguenti:

- Terminare dopo un numero massimo di passi o di valutazioni della funzione obiettivo. È chiaro che se il numero massimo di iterazioni è troppo piccolo si potrebbe uscire prima di aver trovato una buona soluzione. Questo è generalmente usato insieme a un test di convergenza per forzare l'uscita se l'algoritmo non converge entro un limite di passi. Usato da solo questo metodo è utile in quei casi in cui l'obiettivo è valutare la migliore soluzione trovata in un certo periodo di tempo.

- Considerando di conoscere a priori l'ottimo $f(\mathbf{x}^*)$ della funzione obiettivo implementata ma non la posizione $\mathbf{x}^*$ in cui questo si verifica, terminare quando l'ottimo è stato trovato. Questo criterio termina il processo di ricerca non appena viene trovata una posizione $\mathbf{x}_i$ tale che $|f(\mathbf{x}_i) - f(\mathbf{x}^*)| \leq \varepsilon$, dove ε è l'errore consentito. Il valore di soglia ε deve essere scelto con cura. Se ε è troppo grande, il processo potrebbe terminare in una soluzione sub-ottimale. Se invece ε è troppo piccolo la ricerca potrebbe non terminare mai.

- Terminare quando non ci sono miglioramenti dopo un numero fissato di passi. Ci sono vari modi per valutare un miglioramento. Per esempio, se il cambiamento medio delle posizioni è piccolo, si può assumere che l'algoritmo sia arrivato a convergenza e quindi che lo sciame abbia raggiunto un equilibrio. Oppure se la media delle velocità è prossima allo zero, verranno fatti solo piccoli passi, e quindi la ricerca può considerarsi terminata. E' perciò necessario impostare: 1) il numero di iterazioni entro cui valutare se ci sono miglioramenti, 2) una soglia che definisce quando non ci sono progressi da un passo ad un altro.

- Terminare quando il raggio dello sciame normalizzato è vicino allo zero. In questo caso è necessario calcolare il raggio dello sciame normalizzato così come segue:

$$R_{norm} = \frac{R_{max}}{d(S)}$$

dove $d(S)$ è il diametro dello sciame iniziale; mentre R_{max} rappresenta la distanza massima tra la posizione migliore trovata nell'algoritmo, indicata con $\mathbf{p}_s$, e la posizione di una particella che appartiene allo sciame.

Perciò R_{max} viene così definita:

$$R_{max} = \|\mathbf{x}_m - \mathbf{p}_s\|, \quad m = 1, ..., N$$

con

$$\|\mathbf{x}_m - \mathbf{p}_s\| \geq \|\mathbf{x}_i - \mathbf{p}_s\|, \quad \forall i = 1, ..., N$$

Da ciò si deduce che se R_{norm} è vicino allo zero, lo sciame ha poco potenziale di miglioramento. L'algoritmo terminerà nel momento in cui $R_{norm} < \delta$. Se δ è troppo grande, il processo di ricerca potrebbe interrompersi prima che venga trovata una buona soluzione; al contrario se il valore è troppo piccolo, la ricerca potrebbe richiedere un numero eccessivo di iterazioni affinchè le particelle formino uno sciame compatto, strettamente centrato attorno all'ottimo.

- Terminare quando la pendenza della funzione obiettivo è approssimativamente zero. In questo caso si considera il seguente rapporto:

$$f'(t) = \frac{f(\mathbf{P}_i^s(t)) - f(\mathbf{P}_i^s(t-1))}{f(\mathbf{P}_i^s(t))}$$

Se $f'(t) < \gamma$ per un un numero di iterazioni consecutive, si assume che lo sciame sia convergente. Questo criterio è superiore rispetto agli altri poichè determina effettivamente se lo sciame sta ancora facendo progressi utilizzando le informazioni sul dominio. Tale approccio ha tuttavia il problema che la ricerca terminerà anche nel caso in cui le particelle siano bloccate in un minimo locale, indipendentemente dal fatto che le altre particelle possano ancora essere impegnate ad esplorare altre parti del dominio. Il criterio infatti richiede l'utilizzo in contemporanea di un altro criterio di cui sopra, in modo da verificare se le particelle siano convergenti nello stesso punto prima di terminare il processo di ricerca.

2.3.4 Confinamento

Ad ogni iterazione, ogni agente i può ritrovarsi in linea di principio fuori dal dominio fisico per effetto della velocità. Per ovviare a questa criticità Robinson e Rahmat-Samii (2004) [RRS04] hanno proposto tre diverse tecniche di controllo (Figura 12).

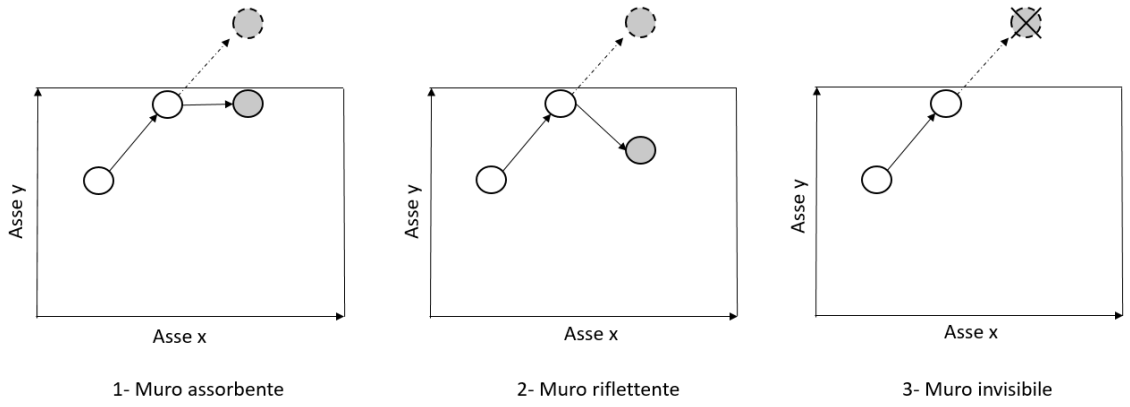


Figura 12: Tipologie di condizioni al contorno

1. *muro assorbente*: una volta che una particella colpisce il bordo del dominio Ω considerato, tramite questa condizione la velocità in quella dimensione viene azzerata e la particella è ricollocata sul bordo del dominio stesso.
   Perciò se si considera una particella i, la cui componente d-esima esce fuori dal dominio considerato, ovvero:

$$\begin{aligned} &\text{se } x_{i,d}(t+1) > x_{max} \text{ allora: } \quad x_{i,d}(t+1) = x_{max} \\ &\text{se } x_{i,d}(t+1) < x_{min} \text{ allora: } \quad x_{i,d}(t+1) = x_{min} \end{aligned} \tag{28}$$

   dove si ricorda che x_{max} e x_{min} indicano gli estremi delle dimensioni del dominio Ω (§ 2.3.1).
   In questo senso le "pareti" di confine assorbono l'energia delle particelle che cercano di sfuggire al dominio.

2. *muro riflettente*: cambia la direzione della velocità della particella; cambia quindi il segno della velocità e la particella viene riflessa all'interno del dominio:

$$x_{i,d}(t+1) = x_{i,d}(t) - v_{i,d}(t+1) \tag{29}$$

   dove d è la dimensione che indica la componente della posizione che sta uscendo dal dominio.

3. *muro invisibile*: in questo caso invece la particella può spostarsi senza alcun limite imposto. Quando però il dominio delle soluzioni accettabili viene oltrepassato dalla particella, la funzione obiettivo non viene valutata in quel punto, risparmiando così a livello computazionale.
   Per quasi tutte le applicazioni ingegneristiche, la parte computazionalmente costosa dell'algoritmo è la valutazione della funzione obiettivo. La motivazione alla base di questa tecnica è quella di risparmiare tempo di calcolo valutando solo ciò che è nello spazio consentito, senza interferire con il movimento naturale dello sciame.

2.3.5 Fattori di apprendimento

I coefficienti di accelerazione c_1 e c_2, presenti nell'equazione di aggiornamento della velocità in componenti (14), quando sono moltiplicati per dei vettori casuali r_1 e r_2, rendono controllabili le influenze stocastiche sulla velocità dello sciame. In poche parole, c_1 e c_2 rappresentano i pesi dei termini di accelerazione stocastica, che spingono ogni particella verso l'*attrattore cognitivo* $\mathbf{p}_i(t)$ o verso l'*attrattore sociale* $\mathbf{P}_i^s(t)$.
Impostare $c_2 = 0$, vuol dire che le singole particelle si baseranno esclusivamente sulla propria conoscenza, mentre impostare $c_1 = 0$ significare che le particelle si baseranno solo sulla conoscenza della particella migliore nell'intorno di comunicazione.
In generale i valori di questi due fattori sono mantenuti costanti durante l'esecuzione dell'algoritmo e uguali tra loro, in modo da garantire lo stesso peso sia per la ricerca sociale che per quella cognitiva.
Anche in questo caso diverse sono state le ricerche e le sperimentazioni per cercare dei valori ottimali dei fattori di apprendimento. In particolare Kennedy (1997) ha studiato due casi estremi: un modello con il solo termine sociale e uno con il solo termine cognitivo, dimostrando che entrambi i termini sono fondamentali per il successo della ricerca dello sciame.

2.4 Scelta implementativa del modello

Il modello scelto per l'implementazione numerica è la versione canonica dell'algoritmo PSO (nota anche come *Inerzia Weight*) di Shi ed Eberhart, descritto in questo capitolo nel paragrafo §2.1.
Come si è già visto, il movimento delle particelle in questo modello è controllato da tre fattori:

- la componente inerziale, descritta dal parametro w;
- la conoscenza propria di ogni particella;
- la conoscenza del quartiere topologico.

Tutti gli aspetti e le condizioni descritte in precedenza devono essere definite affinché l'algoritmo possa essere implementato; le caratteristiche fondamentali, che sono state scelte per questo lavoro di tesi, sono le seguenti:

- **Inizializzazione.** Si è scelto di impostare un dominio Ω che abbia come estremi limite i seguenti valori:

  $$x_{min} = -100$$
  $$x_{max} = 100$$

  In particolare, nel successivo capitolo, saranno analizzati due differenti domini Ω, uno uni-dimensionale e uno bi-dimensionale, così da poter confrontare i diversi comportamenti dell'algoritmo PSO canonico.
  Seguendo la teoria relativa al PSO, le posizioni iniziali delle singole particelle sono state impostate a valori casuali, così da spaziare lungo tutto il dominio, mentre le velocità sono state inizializzate a valori nulli. La posizione migliore di ogni particella i-esima, inoltre, è stata impostata uguale alla posizione iniziale.

- **Dimensione della popolazione.** Il numero di individui all'interno di una popolazione influenza la diversità dello sciame. Maggiore è il numero di particelle, maggiore è la copertura del dominio ad ogni iterazione. La dimensione della popolazione tuttavia influisce anche sulla complessità dell'algoritmo dal momento che ad ogni passo temporale ogni agente deve valutare la funzione obiettivo. Per raggiungere un buon compromesso tra copertura dello spazio e velocità di calcolo si è scelto di lavorare con 50 particelle.

- **Numero massimo di iterazioni.** Anche il numero di iterazioni per raggiungere una buona soluzione dipende dal problema. Un numero insufficiente di iterazioni potrebbe interrompere prematuramente la ricerca, viceversa un numero troppo elevato ha come conseguenza una complessità computazionale aggiuntiva non necessaria. In questo caso si è scelto di impostare il numero massimo di iterazioni $T_{max} = 500$.

- **Condizioni d'uscita.** Un algoritmo può continuare ad eseguire calcoli computazionali fino al numero massimo di iterazioni o terminare prima, nel caso in cui sia già stato trovato l'ottimo globale. Quest'ultima condizione è possibile solo nel caso in cui il valore minimo sia noto, altrimenti è necessario utilizzare differenti condizioni per terminare l'algoritmo. In questo lavoro di tesi, di tutte le funzioni test utilizzate si conosce il minimo globale

$f(\mathbf{x}^*)$ e quindi la condizione d'uscita che viene applicata è proprio quella di terminare l'algoritmo nel momento in cui si verifica la condizione:

$$|f(\mathbf{x}_i) - f(\mathbf{x}^*)| \leq \varepsilon$$

dove $\varepsilon = 1 \cdot 10^{-10}$ è il valore di soglia scelto.

Nell'eventualità che non si verifichi questa condizione, l'algoritmo continuerà a lavorare finchè non si raggiungerà il numero massimo di iterazioni.

- **Confinamento.** Se il dominio Ω è chiuso, le particelle non possono uscirne, quindi è necessario dare delle condizioni al bordo del dominio. In questo caso la tecnica scelta è quella del *muro assorbente*, in cui se la particella, in una certa dimensione, supera uno degli estremi limite, allora la posizione è reimpostata sul bordo stesso del dominio, azzerando la componente della velocità lungo quella dimensione.

Gli ultimi parametri che devono ancora essere definiti sono: il peso inerziale e i fattori di apprendimento. Come già accennato, la scelta di questi valori è correlata e, in particolare, sono proprio questi che governano l'algoritmo e ottimizzano il movimento delle particelle. Per fare in modo che l'algoritmo sia perfettamente funzionante e per trovare dei valori ottimali di questi coefficienti, sono state fatte diverse prove, che verranno descritte nel successivo capitolo.

3 Modifiche all'algoritmo proposto

Per l'implementazione dell'algoritmo è stato utilizzato il software *Matlab*, ambiente per il calcolo numerico e l'analisi statistica sviluppato da *MathWorks*.
L'algoritmo scelto, come già accennato nel capitolo precedente, è il modello canonico dell'algoritmo PSO, in cui si ricorda che le equazioni che descrivono il movimento delle particelle sono le seguenti:

$$\begin{cases} \mathbf{x}_i(t+1) = \mathbf{x}_i(t) + \mathbf{v}_i(t+1) \\ \mathbf{v}_i(t+1) = w\mathbf{v}_i(t) + \mathbf{f}_i \end{cases} \tag{30}$$

dove il vettore $\mathbf{f}_i = \sum_{k=1}^{2} \mathbf{F}_{i,k} = \mathbf{F}_{i,1} + \mathbf{F}_{i,2}$ è la risultate delle forze esterne che agiscono sulle particelle, ovvero conoscenza personale $\mathbf{F}_{i,1}$ e globale $\mathbf{F}_{i,2}$.

Il dominio Ω, in cui le particelle sono inizializzate e all'interno del quale devono rimanere durante l'esecuzione dell'algoritmo, è stato scelto inizialmente uni-dimensionale e successivamente si estenderanno le simulazioni al caso bi-dimensionale.

Si ricordano inoltre le condizioni e i parametri che rimarranno tali per tutte le simulazioni presentate da qui in avanti, riassunte nella tabella in Figura 13:

Limite superiore per ogni dimensione del dominio	x_{max} =100
Limite inferiore per ogni dimensione del dominio	x_{min} =-100
Posizione iniziale di ogni particella $\mathbf{x}_i(0)$	casuale
Velocità iniziale di ogni particella $\mathbf{v}_i(0)$	$\mathbf{v}_i(0) = 0$
Posizione migliore di ogni particella $\mathbf{p}_i$ all'istante iniziale	$\mathbf{p}_i(0)=\mathbf{x}_i(0)$
Numero massimo di iterazioni	t=500
Valore di soglia per la condizione d'uscita	$\varepsilon=1\cdot10^{-10}$
Condizioni al bordo	Muro assorbente

Figura 13: Tabella riassuntiva

Inoltre, è fondamentale decidere la topologia attraverso la quale le particelle comunicheranno tra loro: in questa parte prima parte del capitolo si è scelto di implementare una topologia *gBest*, in cui tutte le informazioni sono condivise dall'intero sciame. Successivamente, si cambierà struttura di comunicazione, confrontando i risultati ottenuti.

3.1 Prima funzione obiettivo

Per poter analizzare e approfondire l'algoritmo PSO a livello computazionale, è necessario decidere quale tipo di problema affrontare: in questo lavoro di tesi si è scelto di applicare un problema di minimizzazione e quindi lo scopo dello sciame sarà quello di riuscire a muoversi e, tramite la comunicazione tra gli agenti e la propria esperienza, arrivare al minimo di una funzione obiettivo $f : \mathbb{R}^n \to \mathbb{R}$, dove n è la dimensione del dominio.
In quest'ottica il problema di minimizzazione può essere definito nel seguente modo:

$$\text{Trovare } \mathbf{x}^* \in \Omega : f(\mathbf{x}^*) = min_{\mathbf{x}\in\Omega} f(\mathbf{x}) \tag{31}$$

Esistono molte funzioni obiettivo in letteratura, progettate per testare le prestazioni degli algoritmi di ottimizzazione: queste vengono divise in funzioni unimodali (con un solo minimo/massimo) e funzioni multimodali.

La prima funzione test che viene scelta è una funzione del tipo:

$$f(\mathbf{x}) = \sum_{j=1}^{N} (x_j)^2 \tag{32}$$

$$x_j \in [-100, 100], \qquad f(\mathbf{x}^*) = 0$$

dove $j = 1, 2, ..., N$ indica la dimensione del dominio Ω scelto.

In questo capitolo, le modifiche applicate all'algoritmo e i risultati sperimentali che verranno presentati faranno tutti riferimento a questa funzione obiettivo. I risultati legati ad un'altra funzione di test saranno presentati nel capitolo successivo.

3.2 Algoritmo PSO canonico nel caso uni-dimensionale

La prima simulazione che viene effettuata riguarda l'algoritmo di Shi ed Eberhart (30) in un dominio Ω uni-dimensionale, considerando la funzione obiettivo (32) con $N = 1$, che diventa quindi:

$$f(x) = x^2 \tag{33}$$

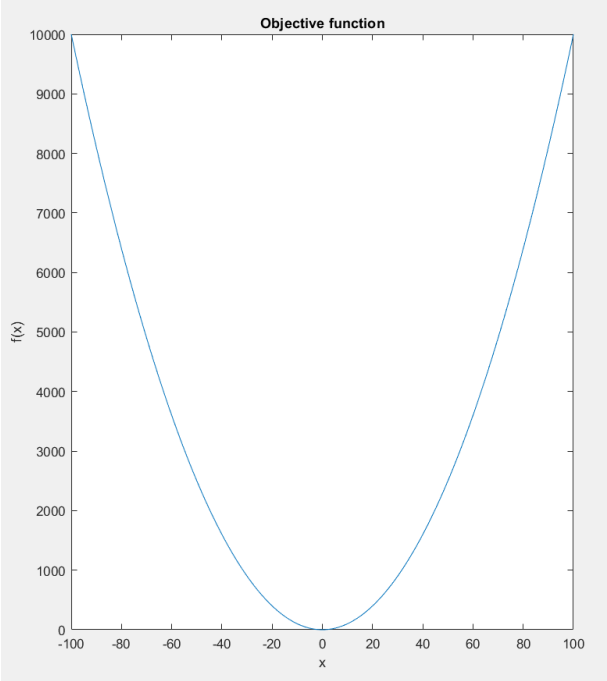


Figura 14: Grafico della funzione obiettivo con N=1

A partire dai parametri e dagli aspetti riassunti nel paragrafo precedente nella tabella in figura (13), devono essere scelti peso inerziale e fattori di apprendimento.

E' importante sottolineare che i valori dei parametri w, c_1, c_2 sono stati scelti dopo aver effettuato diverse simulazioni preliminari sull'algoritmo. Il movimento delle particelle è, come si è già visto, influenzato da questi tre parametri e per ogni problema è necessario approfondire la scelta dei loro valori: ci sono infatti algoritmi per cui alcuni valori sono ottimi, ma altri in cui gli stessi valori "peggiorano" l'algoritmo.

Sono molti gli articoli in letteratura in cui i valori dei coefficienti vengono scelti così:

$$\begin{cases} w = 1 \\ c_1 = 2 \\ c_2 = 2 \end{cases} \tag{34}$$

Andando a rappresentare, iterazione per iterazione, le posizioni delle singole particelle, si è notato un movimento poco ordinato delle particelle prima di arrivare a convergenza (Figura 15):

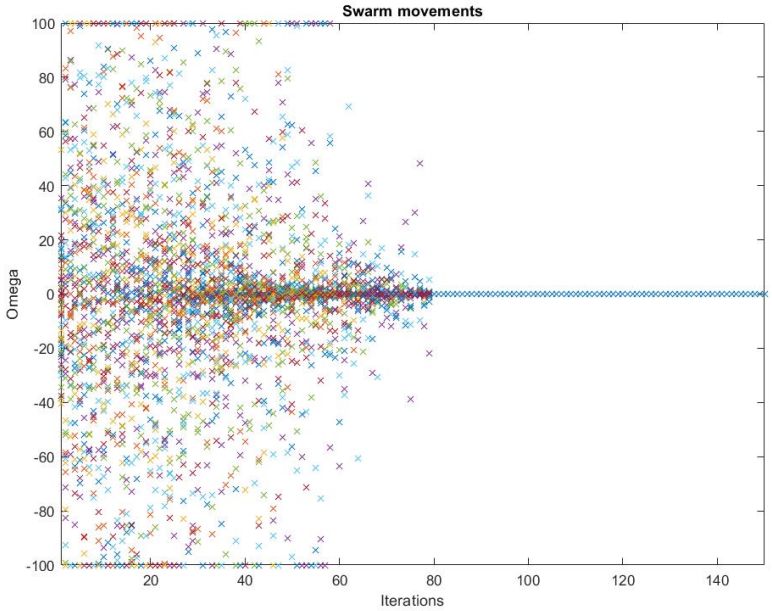


Figura 15: Posizioni delle singole particelle, iterazione per iterazione, considerando i parametri della letteratura

In quest'ottica sono state considerate diverse combinazioni di valori dei parametri w, c_1 e c_2, mantenendo il vincolo $c_1 = c_2$: nella tabella in Figura 16 vengono riportati i valori scelti per i coefficienti, calcolando il tempo medio e le iterazioni necessarie all'algoritmo per arrivare a convergenza, eseguendo 10 simulazioni per caso.

Simulazioni	Tempo medio(s)	Iterazioni totali medie
w=0.1 ; c_1=c_2=1.5	0.03	10
w=0.2 ; c_1=c_2=1.5	0.01	13
w=0.3 ; c_1=c_2=1.5	0.02	18
w=0.4 ; c_1=c_2=1.5	0.02	22
w=0.5 ; c_1=c_2=1.5	0.02	25
w=0.6 ; c_1=c_2=1.5	0.02	31
w=0.7 ; c_1=c_2=1.5	0.03	34
w=0.8 ; c_1=c_2=1.5	0.04	47
w=0.9 ; c_1=c_2=1.5	0.05	58

Figura 16: Valori medi per le simulazioni effettuate nel caso uni-dimensionale

Una volta effettuate le simulazioni, è possibile considerare un esempio di terna dei parametri, per confrontarlo con il precedente risultato. Si è scelto di impostare i parametri nel seguente modo:

$$\begin{cases} w = 0.3 \\ c_1 = 1.5 \\ c_2 = 1.5 \end{cases} \tag{35}$$

Questa scelta è arbitraria: in particolare, per valori di $w \leq 0.5$, il comportamento è pressochè lo stesso, cambia solo il numero medio di iterazioni necessario per la convergenza.

A questo punto è possibile rappresentate le posizioni, iterazione per iterazione, delle singole particelle (Fig. (17)):

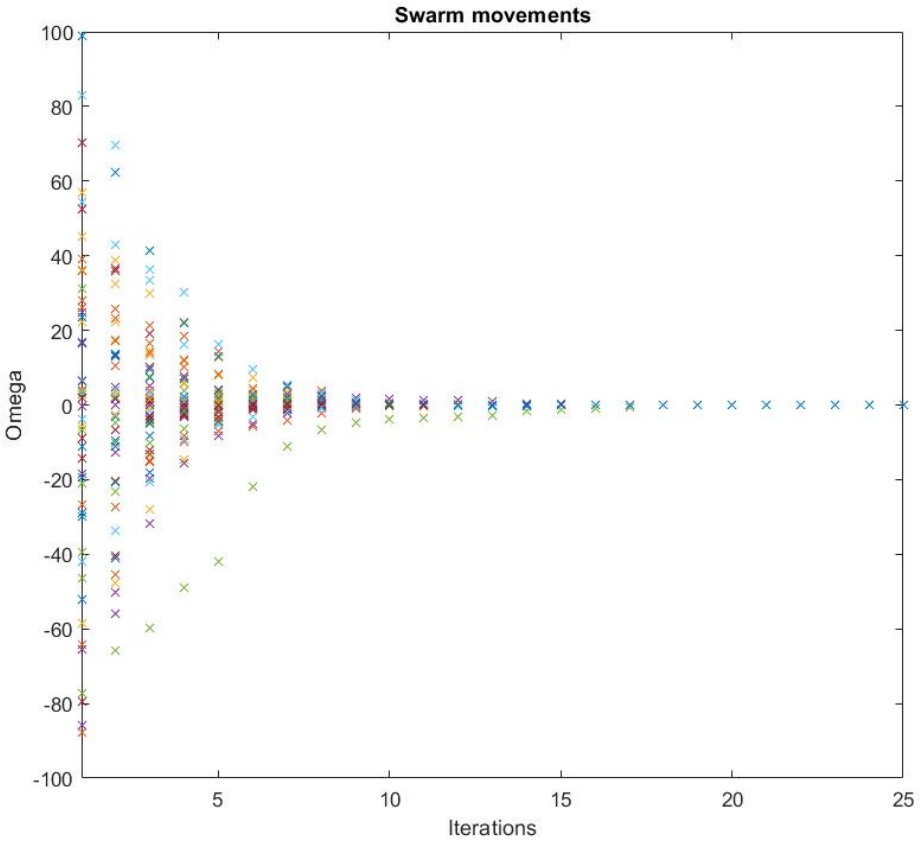


Figura 17: Posizioni delle singole particelle, iterazione per iterazione, nel caso uni-dimensionale

E' chiaro il comportamento che assumono le particelle rispetto al caso precedente: mano a mano che le iterazioni aumentano, lo sciame diventa sempre più compatto e l'algoritmo arriva subito a convergenza (come si può notare dalla figura, le particelle sono già tutte posizionate sul minimo dopo circa 20 iterazioni).

3.3 Algoritmo PSO canonico nel caso bi-dimensionale

Il secondo caso che viene analizzato è quello di considerare l'algoritmo di Shi ed Eberhart (30) in un dominio Ω bi-dimensionale.
La funzione obiettivo (32) diventa quindi:

$$f(\mathbf{x}) = \sum_{j=1}^{2} (x_j)^2 = x_1^2 + x_2^2 \tag{36}$$

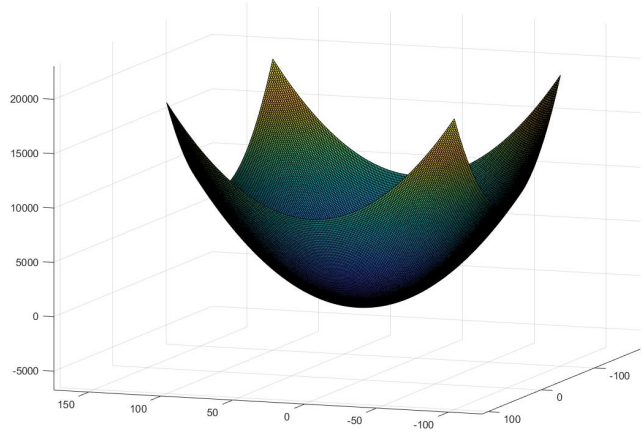


Figura 18: Grafico della funzione obiettivo con N=2

Come già accennato in precedenza, anche in questo caso vengono fatte diverse simulazioni, analizzandone i risultati sia dal punto di vista della dinamiche delle particelle che dal punto di vista di ricerca del minimo, per impostare i valori dei coefficienti w, c_1 e c_2 ottimizzando l'algoritmo.
Inoltre, durante le simulazioni, viene considerato un parametro $w_{damp} = 0.99$, che ha il compito di smorzare, iterazione dopo iterazione, il peso inerziale w:

$$w(t+1) = w_{damp} \cdot w(t) \tag{37}$$

3.3.1 Scelta dei parametri

Come prima scelta si analizza l'algoritmo a partire dai valori noti in letteratura, considerando la presenza dello smorzamento w_{damp}:

$$\begin{cases} w = 1 \\ c_1 = 2 \\ c_2 = 2 \\ w_{damp} = 0.99 \end{cases} \tag{38}$$

Tenuto conto di ciò si eseguono allora diverse simulazioni con i valori definiti in (13).

A partire da posizioni casuali iniziali delle particelle (Fig. 19), queste iniziano a muoversi nel dominio Ω: di seguito verranno mostrate alcune istantanee del processo, seguendo una specifica particella per sottolinearne il movimento (Fig. 20).

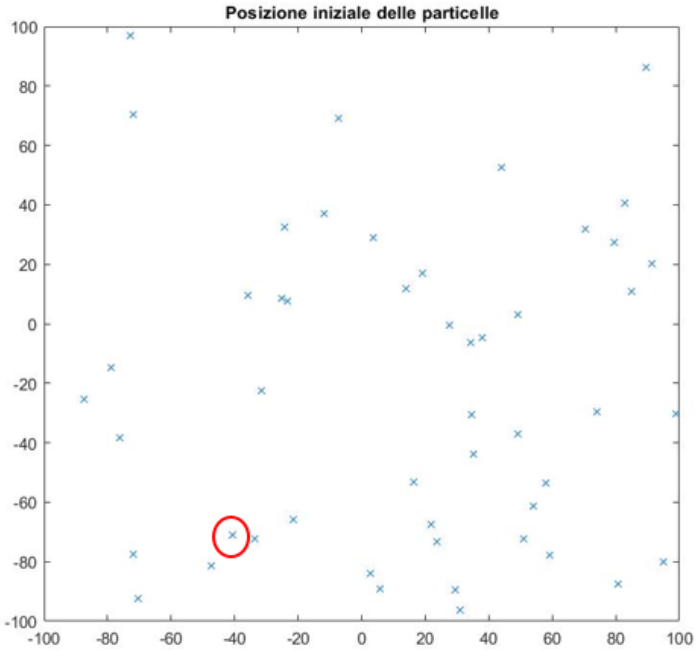
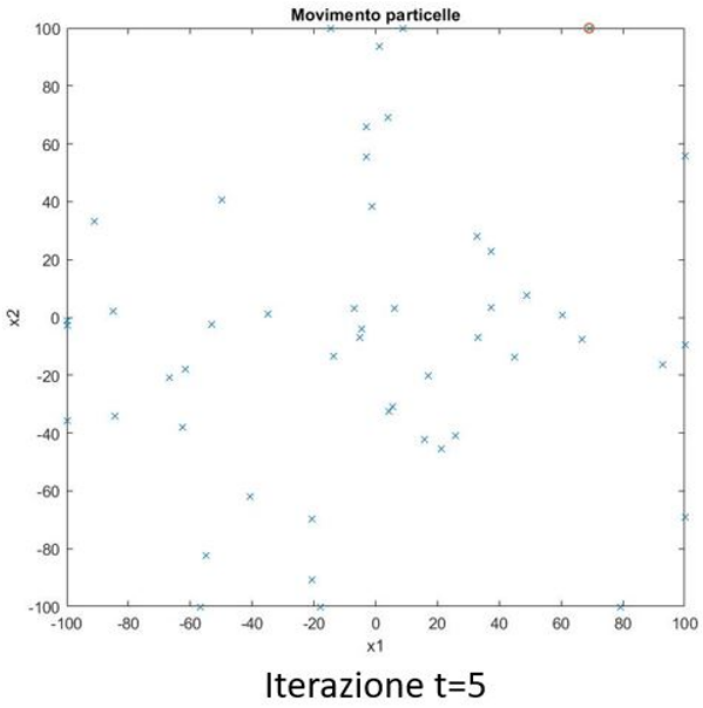
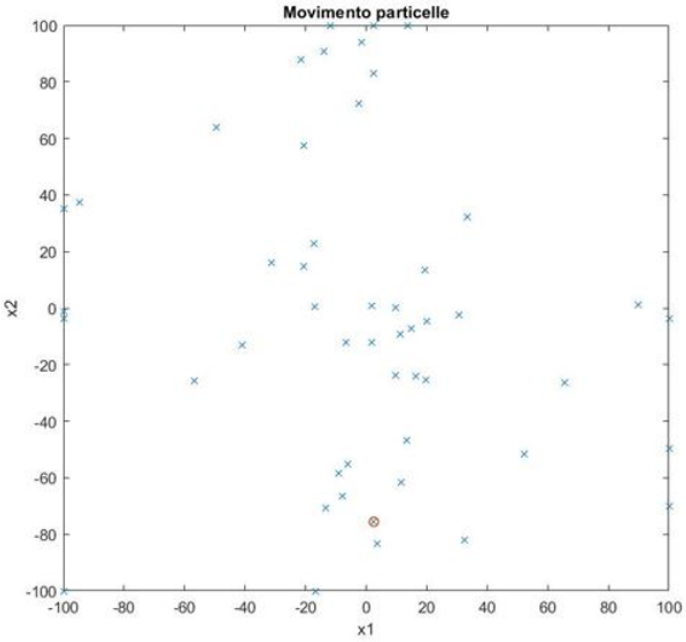
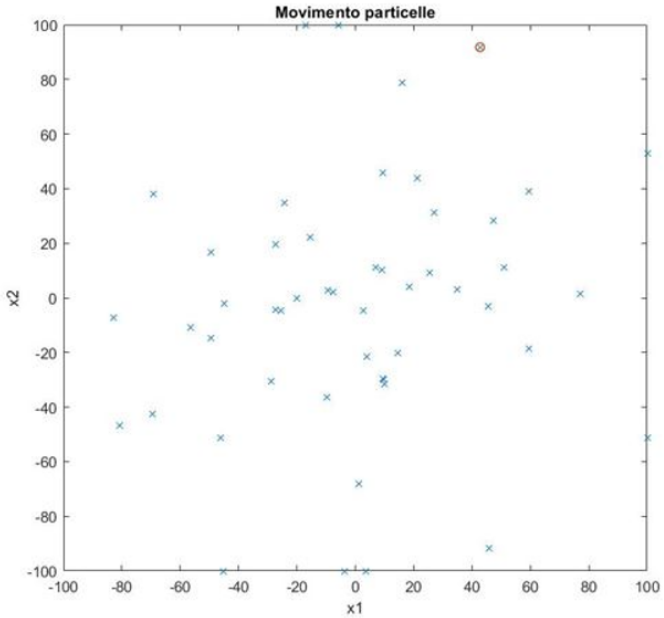


Figura 19: Posizione iniziale casuale delle particelle

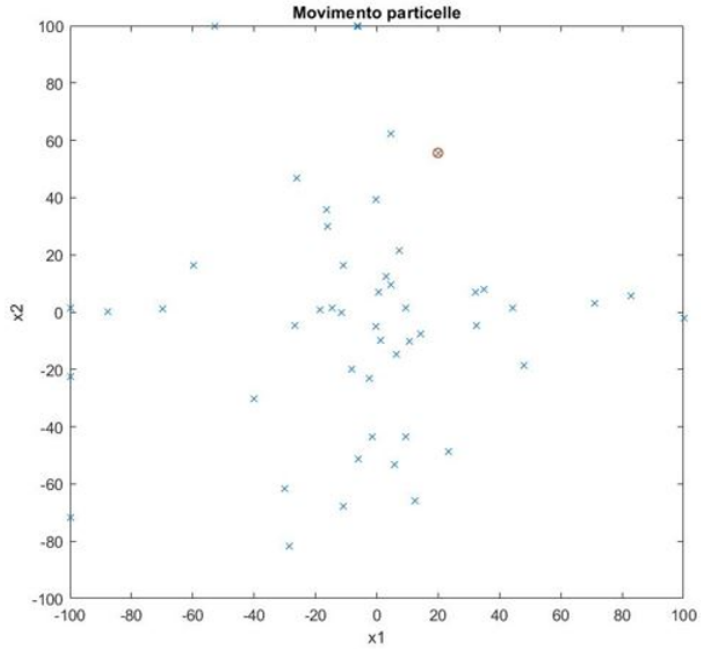




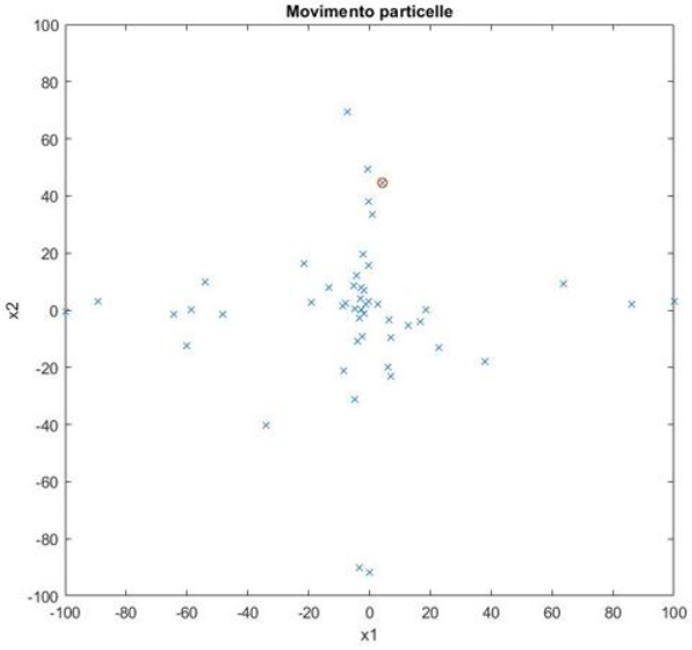
Iterazione t=10



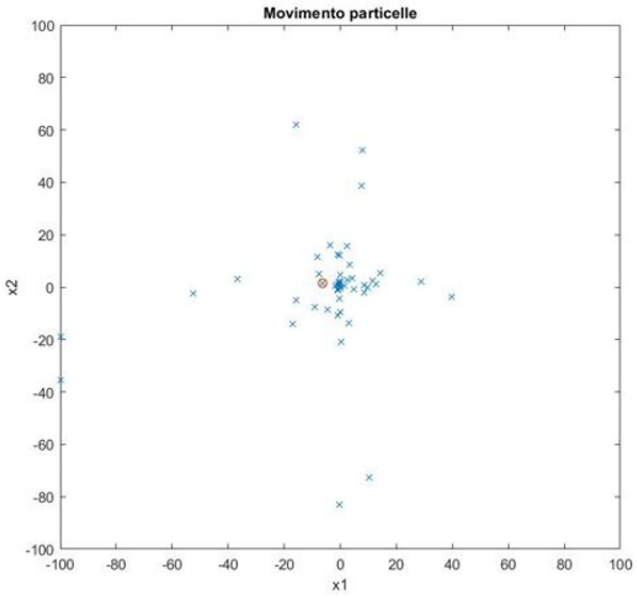
Iterazione t=20



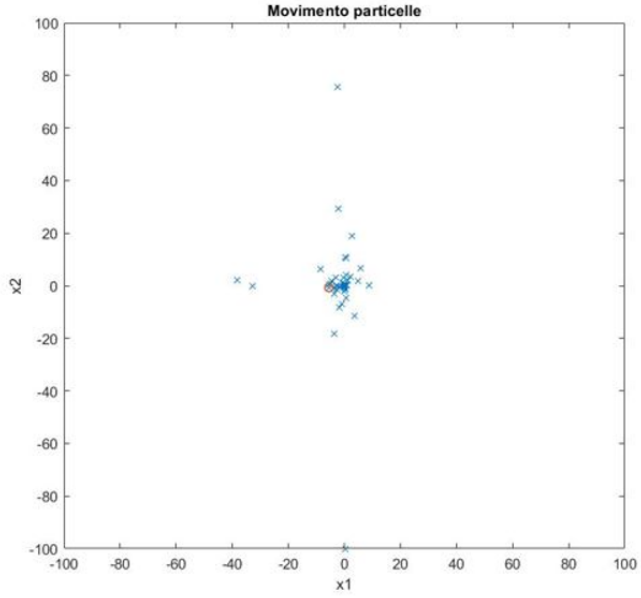
Iterazione t=30



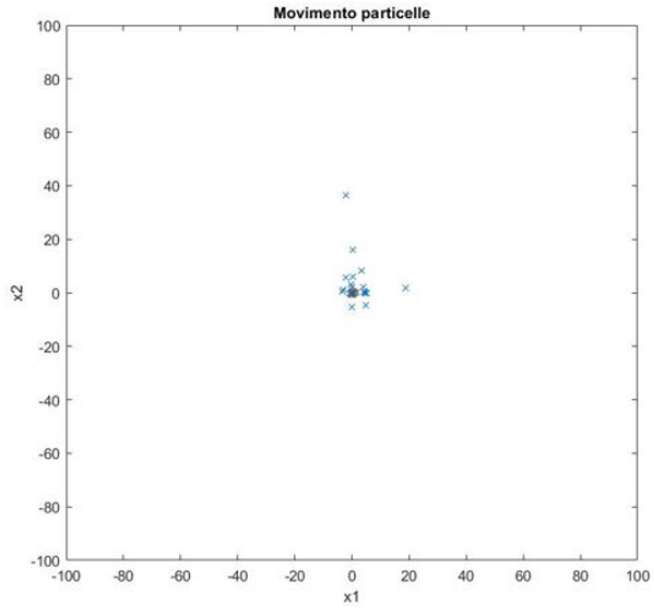
Iterazione t=40



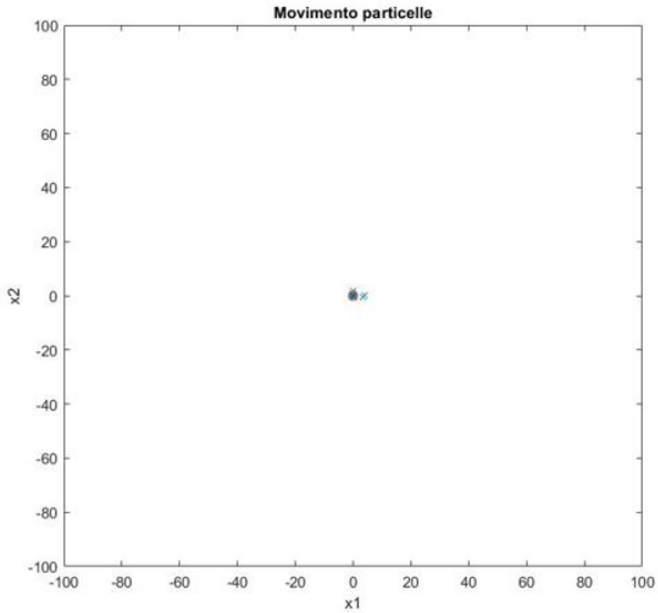
Iterazione t=50



Iterazione t=65



Iterazione t=75



Iterazione t=90

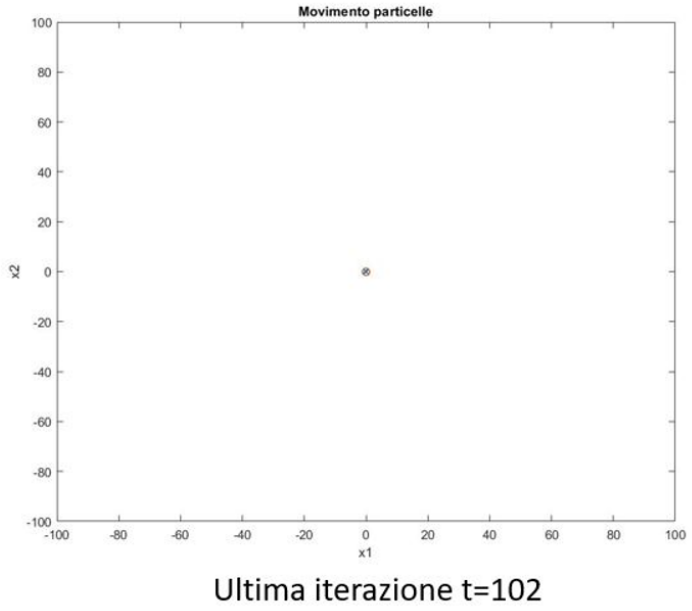


Figura 20: Vari istanti della simulazione, evidenziando il movimento di una singola particella

Si nota che le particelle, nei primi istanti di tempo, si muovono con dei "*passi*" molto grandi all'interno del dominio, arrivando anche sul bordo (allontanandosi quindi da quello che è il punto di minimo). Superato un certo numero di iterazioni, questi passi si riducono sempre di più, fino a che poi l'algoritmo si ferma perchè l'intero sciame ha raggiunto il punto desiderato. In particolare, l'algoritmo è stato eseguito per 10 volte, individuando che il numero medio di iterazioni entro cui lo sciame arriva all'ottimo è di circa 100 iterazioni, in un tempo medio di 158,3 secondi.

Andando a diminuire le componenti del vettore forza $\mathbf{f}_i$ presente nella formula di aggiornamento della velocità, governato dai coefficienti di accelerazione c_1 e c_2, e quindi impostando tali coefficienti pari a 0.5, si nota che il movimento è più lento e le particelle sembrano avvicinarsi verso l'ottimo per poi tornare indietro: questo movimento si attenua con l'aumentare del numero di iterazioni.

Una spiegazione al comportamento dello sciame, descritto sopra, può essere ricercata nel valore impostato per il peso inerziale: le particelle, nel tempo, si muovono con un effetto, che le avvicina e le allontana, sempre minore, situazione che può essere correlata al fatto che la componente inerziale è più grande delle altre componenti della dinamica e la presenza del fattore w_{damp} "smorza" il valore di w durante le simulazioni. Questo si traduce in una "riduzione" della componente inerziale presente nel passo delle particelle, che le aiuta ad arrivare più facilmente al minimo.
In quest'ottica si è allora indagato sulla variazione del valore di w, simulando diverse volte l'algoritmo con diversi valori del peso inerziale, ed eventualmente dei coefficienti di apprendimento.

- **Simulazioni con diversi valori di w:**

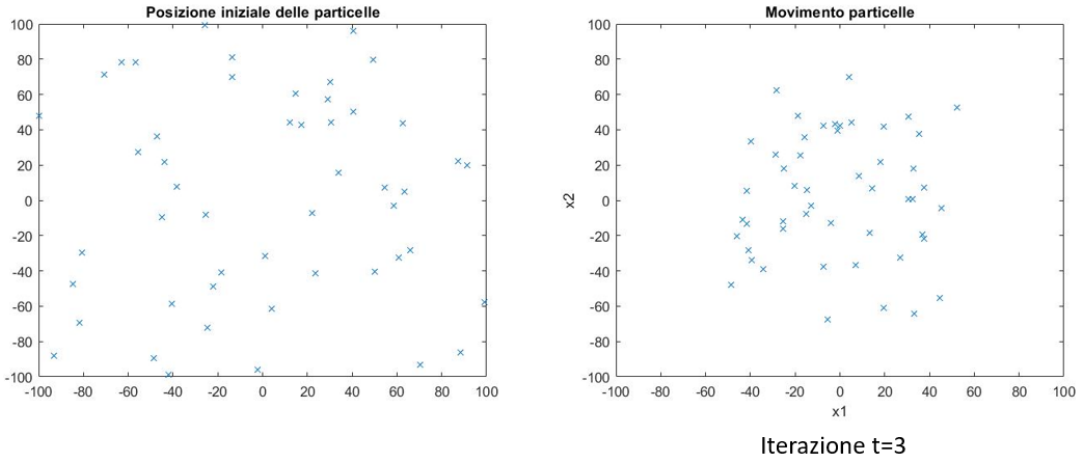
Si considera come primo caso quello di prendere $w = 0$, lasciando sempre invariati gli altri coefficienti. Appare da subito evidente un comportamento "ordinato" delle particelle, che, a partire da posizioni casuali, si avvicinano sempre di più all'ottimo, senza mai allontanarsi.
Essendo $w = 0$, gli unici fattori che influenzano il vettore velocità $\mathbf{v}_i(t+1)$ sono la componente personale e quella collettiva dell'intero sciame, governati da c_1 e c_2. Si sono allora presi in considerazione due casi: uno in cui i coefficienti di accelerazione sono grandi e uno in cui sono piccoli, per confrontarne i risultati.
Nella figura (21) vengono riassunti i risultati, effettuando 10 simulazioni per caso.

CASO: w=0, c_1=c_2=0.5		**CASO:** w=0, c_1=c_2=1.5	
Tempo (s)	**Iterazioni totali**	**Tempo (s)**	**Iterazioni totali**
115.45	57	29.01	12
90.92	51	23.94	10
100.09	45	24.64	10
103.45	46	28.73	12
107.92	45	25.78	11
91.2	47	23.39	10
93.24	41	21.56	13
75.05	42	17.85	11
115.35	44	18.84	12

Figura 21: Tabella riassuntiva dei due casi presi in esame

E' possibile notare come l'algoritmo sia più veloce ad arrivare al minimo impostando dei valori dei coefficienti di accelerazione più grandi piuttosto che dei valori piccoli. Di seguito alcune istantanee del processo, considerando:

$$\begin{cases} w = 0 \\ c_1 = 0.5 \\ c_2 = 0.5 \end{cases}$$



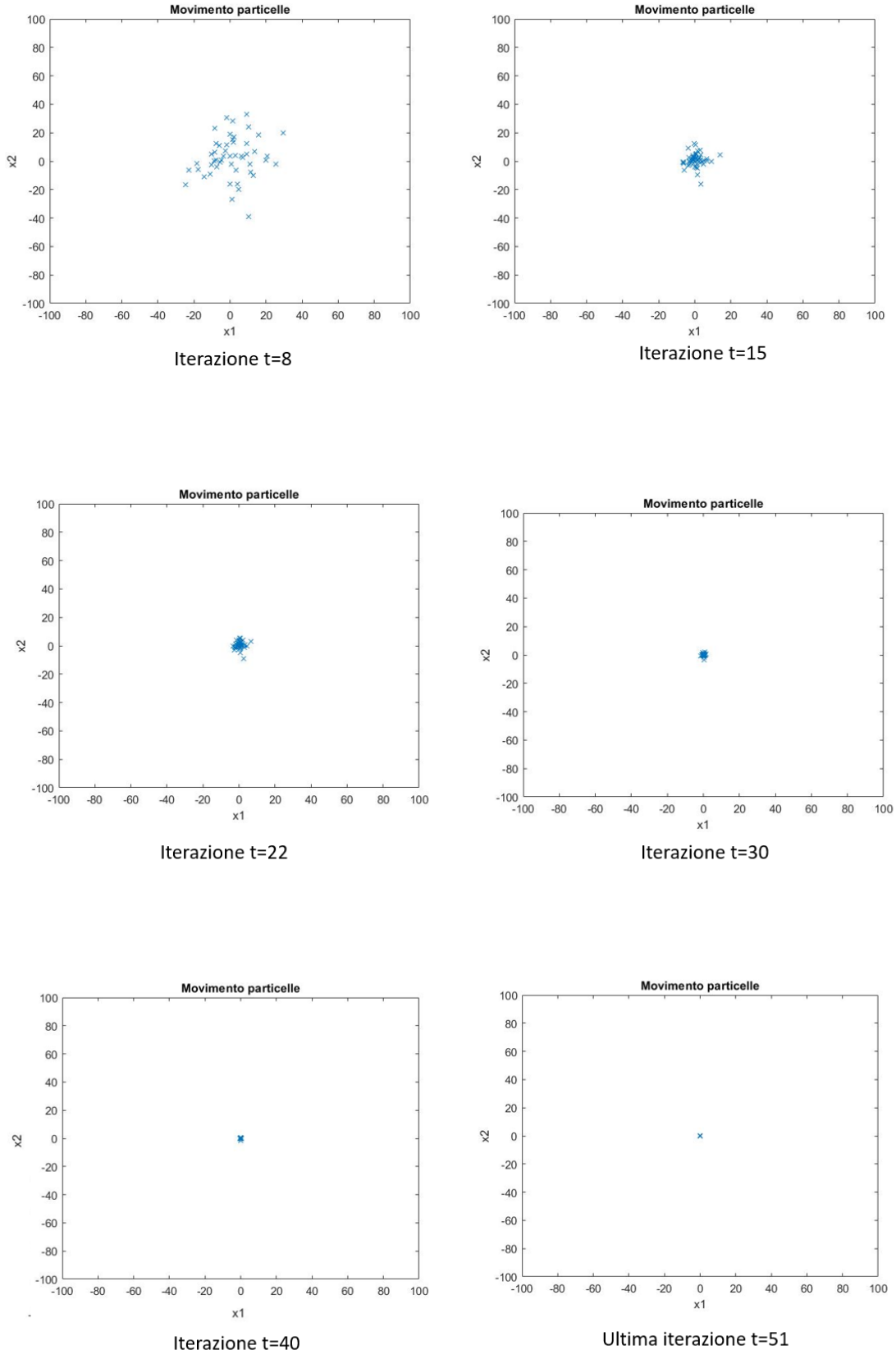


Figura 22: Vari istanti della simulazione

Impostando dunque il peso inerziale a zero, le particelle non si avvicinano/allontano dal minimo (Fig. 22), a conferma dell'ipotesi che il movimento delle particelle viene inficiato dal valore del peso inerziale, ed in particolare sembra essere legato ad un suo valore troppo grande. Mentre si può confermare che i coefficienti di accelerazione governano il modulo delle componenti delle velocità delle singole particelle.

In quest'ottica, fissando $w_{damp} = 0,99$, vengono scelti diversi valori di coefficienti di apprendimento, c_1 e c_2, e di w, analizzando se l'algoritmo arriva a convergenza e, se arriva, quanto tempo impiega.
Per ogni terna di valori vengono effettuate 10 simulazioni in presenza di smorzamento e 10 in assenza di smorzamento, ottenendo i seguenti risultati medi (Fig. 23 e 24):

Simulazioni	Tempo medio(s)	Iterazioni totali medie
	Con Smorzamento	
w=0.1 ; c_1=c_2=0.5	50.5	41
w=0.2 ; c_1=c_2=0.5	40.38	32
w=0.3 ; c_1=c_2=0.5	50.7	24
w=0.4 ; c_1=c_2=0.5	54.2	27
w=0.5 ; c_1=c_2=0.5	59.5	31
w=0.6 ; c_1=c_2=0.5	70.56	35
w=0.7 ; c_1=c_2=0.5	89.4	43
w=0.8 ; c_1=c_2=0.5	112.8	51
w=0.9 ; c_1=c_2=0.5	121.4	60
w=0.1 ; c_1=c_2=1.5	18.6	15
w=0.2 ; c_1=c_2=1.5	27	18
w=0.3 ; c_1=c_2=1.5	30.5	24
w=0.4 ; c_1=c_2=1.5	35.4	31
w=0.5 ; c_1=c_2=1.5	56.5	45
w=0.6 ; c_1=c_2=1.5	61.75	45
w=0.7 ; c_1=c_2=1.5	70.36	53
w=0.8 ; c_1=c_2=1.5	81.6	61
w=0.9 ; c_1=c_2=1.5	99.04	71

Figura 23: Tabella riassuntiva di tempi e iterazioni medie per ogni caso considerato, in presenza di smorzamento

Simulazioni	Tempo medio(s)	Iterazioni totali medie
Senza Smorzamento		
w=0.1 ; c_1=c_2=0.5	47.96	38
w=0.2 ; c_1=c_2=0.5	32	26
w=0.3 ; c_1=c_2=0.5	26.4	24
w=0.4 ; c_1=c_2=0.5	32.71	29
w=0.5 ; c_1=c_2=0.5	47.18	36
w=0.6 ; c_1=c_2=0.5	94.3	46
w=0.7 ; c_1=c_2=0.5	151.7	63
w=0.8 ; c_1=c_2=0.5	194.89	99
w=0.9 ; c_1=c_2=0.5	358.9	231
w=0.1 ; c_1=c_2=1.5	26.2	16
w=0.2 ; c_1=c_2=1.5	31.7	21
w=0.3 ; c_1=c_2=1.5	40.4	27
w=0.4 ; c_1=c_2=1.5	51.4	35
w=0.5 ; c_1=c_2=1.5	61.3	43
w=0.6 ; c_1=c_2=1.5	86.1	61
w=0.7 ; c_1=c_2=1.5	123.8	88
w=0.8 ; c_1=c_2=1.5	239.2	173
w=0.9 ; c_1=c_2=1.5	L'algoritmo non arriva a convergenza entro le 500 iterazioni	

Figura 24: Tabella riassuntiva di tempi e iterazioni medie per ogni caso considerato, in assenza di smorzamento

Durante le simulazioni si è notato che:

- Sia in presenza di smorzamento che in assenza, nel caso in cui $c_1 = c_2 = 0,5$, per valori di w piccoli ($w \leq 0.5$), il movimento delle particelle risulta analogo al caso di peso inerziale nullo e le particelle si spostano verso il minimo in maniera ordinata e graduale; lo stesso comportamento si verifica per valori di $c_1 = c_2 = 1,5$. La differenza è solo nel tempo medio entro cui l'algoritmo arriva a convergenza: per valori piccoli dei coefficienti di apprendimento il tempo è maggiore rispetto a coefficienti più grandi.

- Nel caso in cui i valori di w sono più grandi ($w \geq 0.6$), lo smorzamento è necessario all'algoritmo per arrivare più velocemente e facilmente a convergenza. Si può infatti notare nella tabella di cui sopra che i tempi medi aumentano in assenza di smorzamento e, in particolare, nel caso $w = 0,9$ l'algoritmo non riesce ad arrivare a convergenza entro le iterazioni fissate: le particelle risultano infatti sparse nel dominio e si avvicinano/allontanano dal minimo, senza fermarsi lì.

In quest'ottica, considerando tutte le simulazioni effettuate, è chiaro che l'inerzia incide sul movimento delle particelle e che risulta essere più efficace, in questo specifico algoritmo e con le condizioni definite in precedenza, un peso inerziale compreso tra $0.1 \leq w \leq 0.5$.

3.4 Cambiamenti nella struttura di comunicazione

Le simulazioni effettuate nei precedenti paragrafi, come già accennato, fanno tutte riferimento ad una struttura di comunicazione globale tra le particelle.
In questa parte del capitolo sarà analizzato il comportamento dell'algoritmo quando la comunicazione tra le particelle viene modificata; in particolare, viene considerata una struttura sociale ad anello, dove ogni particella può comunicare solo con $n = 2$ suoi vicini adiacenti nella matrice della popolazione, così come in Figura (25):

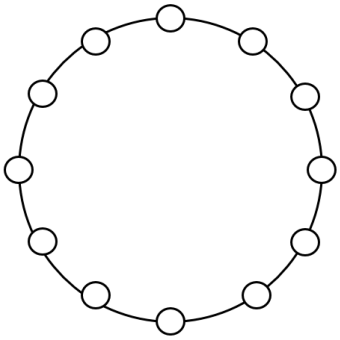


Figura 25: Rappresentazione della struttura sociale ad anello nel caso n=2

3.4.1 Algoritmo PSO canonico nel caso uni-dimensionale con topologia lBest

A partire delle formule che descrivono l'algoritmo PSO canonico (30), con le condizioni e i parametri impostati in Figura (13) e considerando la funzione obiettivo in (33), vengono considerati diversi valori per i coefficienti di apprendimento e per il peso inerziale, eseguendo 10 simulazioni per caso (Fig. 26).

Simulazioni	Tempo medio(s)	Iterazioni totali medie
w=0.1 ; c_1=c_2=1.5	0.85	58
w=0.2 ; c_1=c_2=1.5	0.35	23
w=0.3 ; c_1=c_2=1.5	0.37	23
w=0.4 ; c_1=c_2=1.5	0.41	26
w=0.5 ; c_1=c_2=1.5	0.51	32
w=0.6 ; c_1=c_2=1.5	0.54	37
w=0.7 ; c_1=c_2=1.5	0.65	43
w=0.8 ; c_1=c_2=1.5	0.78	53
w=0.9 ; c_1=c_2=1.5	1.07	63

Figura 26: Valori medi per le simulazioni effettuate nel caso uni-dimensionale con topologia ad anello

In quest'ottica è possibile notare che, rispetto al caso in cui viene considerata una comunicazione globale, il tempo medio (e di conseguenza anche le iterazioni medie) è aumentato: la comunicazione tra le particelle è più lenta e quindi, per arrivare a convergenza, l'algoritmo ha bisogno di più iterazioni.

3.4.2 Algoritmo PSO canonico nel caso bi-dimensionale con topologia lBest

In modo analogo al procedimento seguito per il caso *gBest*, vengono effettuate le simulazioni considerando diversi valori delle coppie (w, $c_1 = c_2$), impostando una topologia ad anello.

Durante le simulazioni, a differenza dei precedenti casi, l'algoritmo non sempre è riuscito ad arrivare a convergenza (si ricorda che il valore di soglia è impostato a 10^{-10}); questo è un risultato che non sorprende: come è noto dalla letteratura, le strutture di comunicazione "*locali*", oltre ad essere più lente rispetto ad una topologia *globale*, hanno anche bisogno di più tempo per arrivare a convergenza (Fig. 27 e 28).

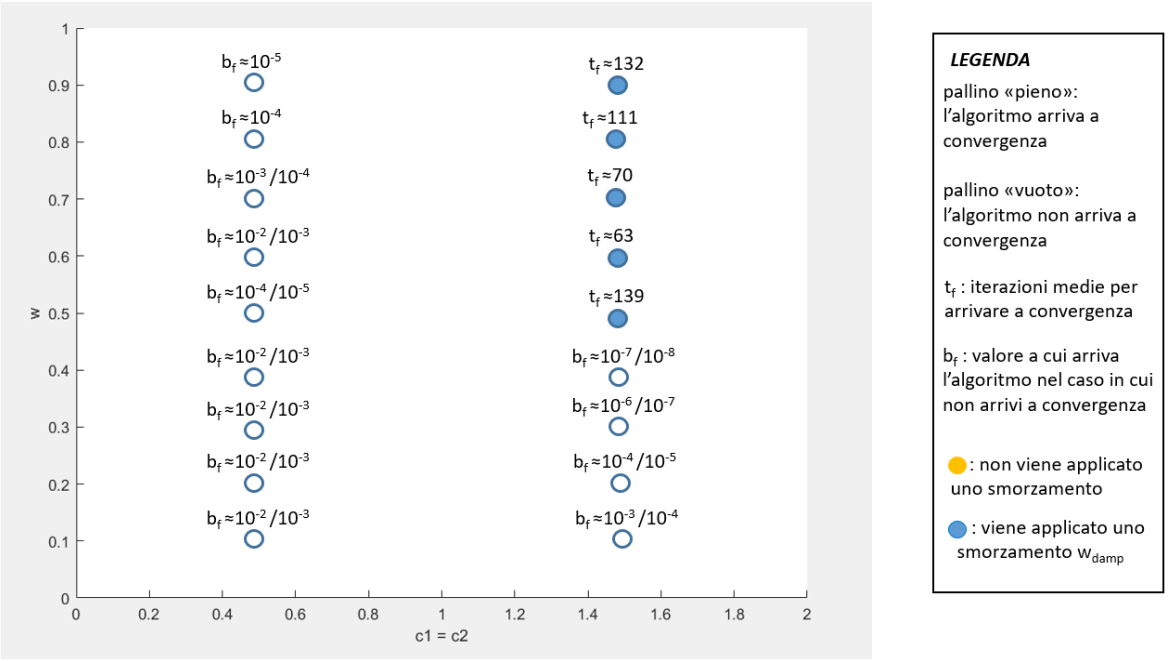


Figura 27: Valori medi per le simulazioni effettuate con topologia ad anello in presenza di smorzamento

In particolare, in presenza di smorzamento, per valori bassi di $c_1 = c_2$, l'algoritmo non arriva mai a convergenza: si nota però che i valori a fine simulazione, b_f nella tabella in Figura (27), sono sempre più piccoli mano a mano che w aumenta. Le particelle rimangono intrappolate e oscillano nell'intorno del minimo senza raggiungerlo e fermarsi lì. La situazione è analoga per il caso di $c_1 = c_2$ più grandi; in questa ipotesi l'algoritmo però converge per valori di w grandi. Una spiegazione può essere ricercata nel fatto che per valori troppo piccoli delle componenti dinamiche, le particelle hanno bisogno di più tempo per muoversi, essendo i passi effettuati molto piccoli e la comunicazione locale.

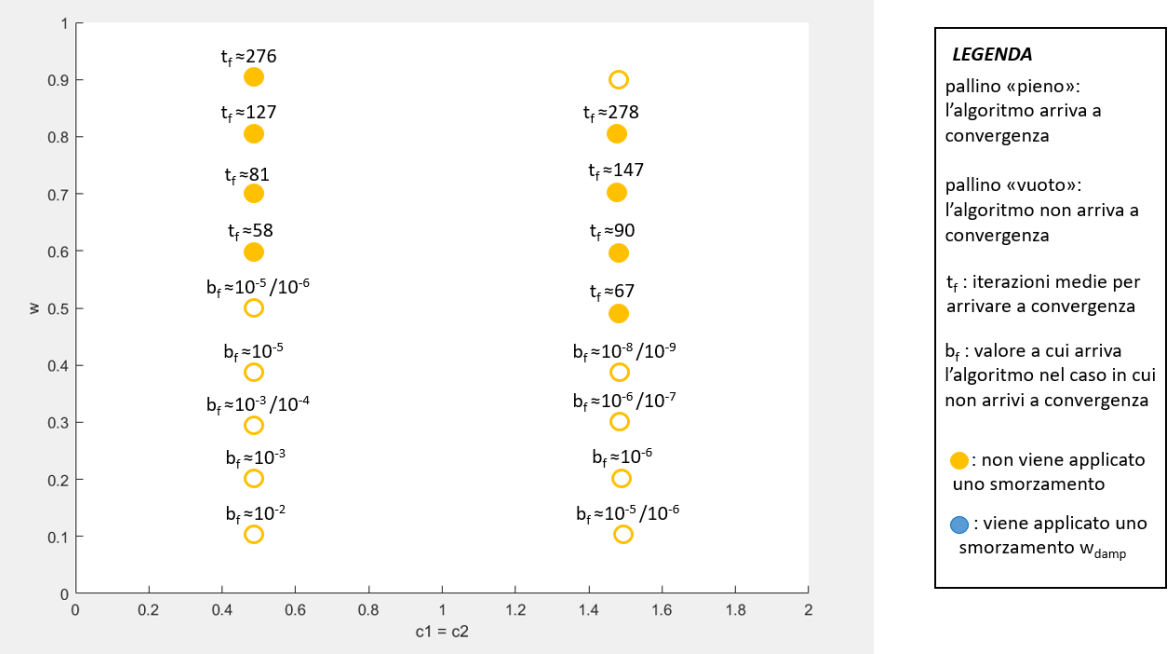


Figura 28: Valori medi per le simulazioni effettuate con topologia ad anello in assenza di smorzamento

Analogamente alle simulazioni precedenti, anche in assenza di smorzamento, per valori piccoli di w, l'algoritmo non converge. Per valori di w grandi invece la situazione è differente: sia per coefficienti di apprendimento piccoli che grandi, l'algoritmo converge.
Questo a conferma di quando detto sopra, essendo le componenti dinamiche piccole, i passi sono piccoli e le particelle hanno bisogno di più iterazioni per poter arrivare al minimo.
Un caso particolare è quello in cui $w = 0.9$, in assenza di smorzamento, nella tabella in Figura (28): le particelle infatti si muovono all'interno del dominio senza riuscire a raggrupparsi in alcun punto, ma continuando a muoversi con velocità elevate.
Una spiegazione può essere ricercata sempre nella dinamica del movimento: le particelle si muovono con un passo troppo grande, che non viene mai smorzato, e quindi arrivare al minimo diventa difficile.

In quest'ottica, come già si era notato nel caso di comunicazione globale, l'inerzia incide sul movimento delle particelle e, in particolare, sul loro "*passo*". In questa specifica condizione, l'algoritmo ha bisogno di valori di $w \geq 0.5$ per arrivare a convergenza entro le 500 iterazioni prefissate, e risulta evidente l'importanza di inserire anche un fattore di smorzamento.

4 Seconda funzione obiettivo

Procedendo in modo analogo a quanto fatto nel precedente capitolo, a partire dall'algoritmo canonico di Shi ed Eberhart (30), si considera la seconda funzione obiettivo:

$$f(\mathbf{x}) = \sum_{j=1}^{N} \Big[0.1(x_j)^2 + 10\Big(1 - cos(2x_j)\Big)\Big] \tag{39}$$

$$x_j \in [-100, 100], \qquad f(\mathbf{x}^*) = 0$$

dove $j = 1, 2, ..., N$ indica la dimensione del dominio Ω scelto.

In tal senso, verranno effettuate diverse simulazioni, analizzando cosa succede variando peso inerziale e coefficienti di apprendimento, e modificando la struttura di comunicazione tra le particelle.

4.1 Dominio uni-dimensionale con topologia *gBest*

Considerando un dominio Ω uni-dimensionale, quindi con $N = 1$, la funzione obiettivo diventa:

$$f(x) = 0.1(x)^2 + 10\Big(1 - cos(2x)\Big) \tag{40}$$

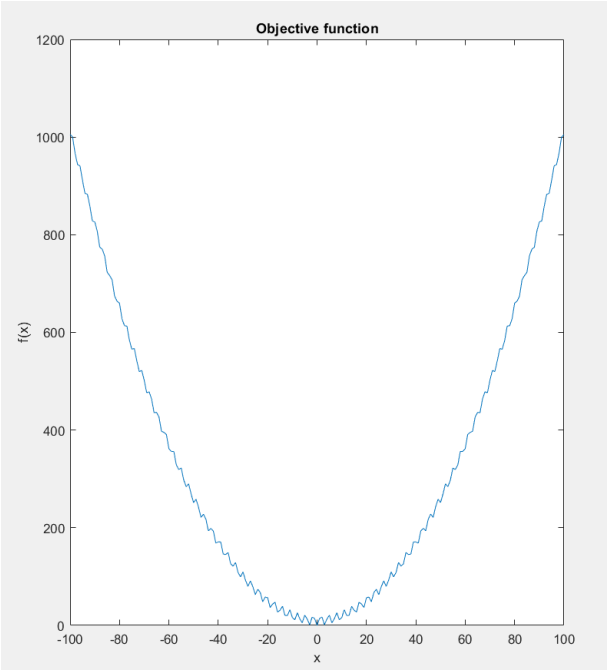


Figura 29: Grafico della funzione obiettivo con N=1

E' possibile notare meglio l'andamento della funzione in un suo dettaglio, considerando un dominio più ristretto, rappresentato in Figura (30).

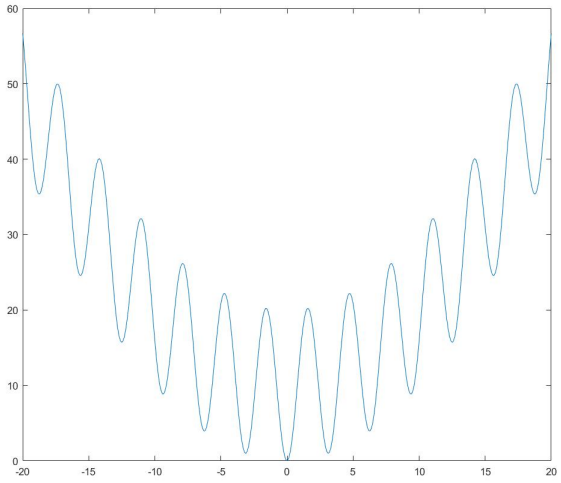


Figura 30: Dettaglio della funzione obiettivo nel caso uni-dimensionale

A partire dai parametri e dagli aspetti riassunti nella tabella in figura (13), nel capitolo 3, vengono effettuate 10 simulazioni per ogni terna di parametri (w, $c_1 = c_2$), applicando una topologia *gBest*.
Per tutte le simulazioni, l'algoritmo arriva a convergenza e nella tabella in Figura (31), vengono riportati i risultati medi.

Simulazioni	Tempo medio(s)	Iterazioni totali medie
w=0.1 ; c_1=c_2=1.5	0.01	13
w=0.2 ; c_1=c_2=1.5	0.02	16
w=0.3 ; c_1=c_2=1.5	0.02	20
w=0.4 ; c_1=c_2=1.5	0.03	24
w=0.5 ; c_1=c_2=1.5	0.03	30
w=0.6 ; c_1=c_2=1.5	0.04	36
w=0.7 ; c_1=c_2=1.5	0.05	44
w=0.8 ; c_1=c_2=1.5	0.06	51
w=0.9 ; c_1=c_2=1.5	0.07	57

Figura 31: Valori medi per le simulazioni effettuate nel caso uni-dimensionale, con topologia *gBest*

Si può già notare che, rispetto ai dati ottenuti applicando la prima funzione obiettivo, riassunti nella tabella in Figura (16), in questo caso i tempi medi sono aumentati leggermente; questo risultato non sorprende dato che la funzione obiettivo, essendo una funzione sinusoidale, presenta diversi minimi locali e quindi lo sciame ha bisogno di più iterazioni per poter arrivare al minimo globale.

In generale però il comportamento dello sciame non cambia rispetto a quanto ottenuto con la prima funzione obiettivo: mano a mano che i valori di w diminuiscono, l'algoritmo arriva prima a convergenza.

4.2 Dominio uni-dimensionale con topologia *lBest*

Mantenendo invariate tutte le condizioni precedentemente descritte, viene studiato l'algoritmo canonico, applicando la funzione sinusoidale (40) e considerando una struttura sociale ad anello (Fig. 25).
Anche in questo caso, per tutte le simulazioni, l'algoritmo arriva a convergenza e nella tabella in Figura (32), vengono riportati i risultati medi.

Simulazioni	Tempo medio(s)	Iterazioni totali medie
w=0.1 ; c_1=c_2=1.5	0.3	24
w=0.2 ; c_1=c_2=1.5	0.4	32
w=0.3 ; c_1=c_2=1.5	0.6	38
w=0.4 ; c_1=c_2=1.5	0.6	41
w=0.5 ; c_1=c_2=1.5	0.6	45
w=0.6 ; c_1=c_2=1.5	0.7	51
w=0.7 ; c_1=c_2=1.5	0.9	63
w=0.8 ; c_1=c_2=1.5	1.02	74
w=0.9 ; c_1=c_2=1.5	1.10	76

Figura 32: Valori medi per le simulazioni effettuate nel caso uni-dimensionale, con topologia *lBest*

I risultati ottenuti continuano ad essere coerenti con quanto analizzato con la prima funzione obiettivo e con quanto noto dalla letteratura. Una topologia ad anello rallenta la comunicazione tra le particelle, quindi l'algoritmo ha bisogno di più tempo, rispetto al caso di comunicazione globale, per arrivare a convergenza.
In ogni caso, all'aumentare di w, aumenta il tempo medio necessario per la convergenza.

4.3 Dominio bi-dimensionale con topologia *gBest*

Considerando un dominio Ω b-dimensionale, quindi con $N = 2$, la funzione obiettivo (39) diventa:

$$f(\mathbf{x}) = \sum_{j=1}^{2} \Big[0.1 \cdot (x_j)^2 + 10\Big(1 - cos(2x_j)\Big)\Big] \tag{41}$$

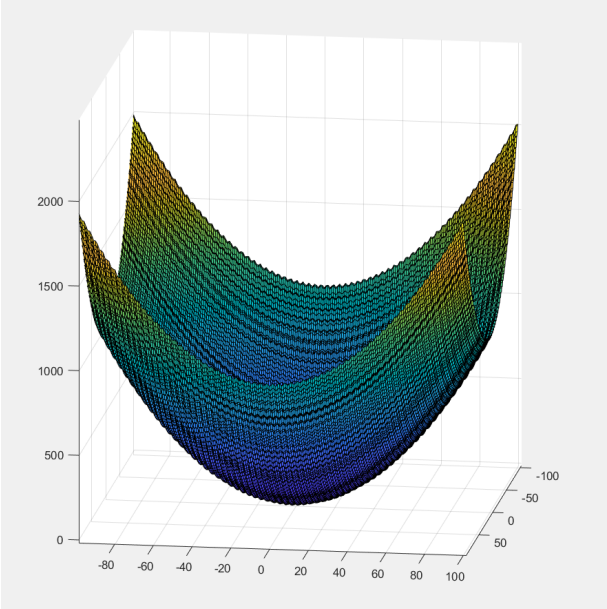


Figura 33: Grafico della funzione obiettivo con N=2

E' possibile notare meglio l'andamento della funzione in un suo dettaglio, considerando un dominio più ristretto, in Figura (34). Essendo una funzione sinusoidale, presenta diversi punti di minimo locali e un unico punto di minimo globale.

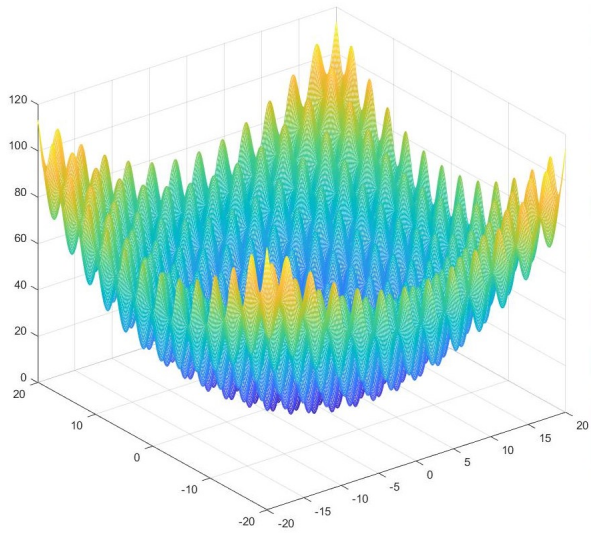


Figura 34: Dettaglio della funzione obiettivo nel caso bi-dimensionale

Si procede in modo analogo a quanto presentato precedentemente e si ottengono i seguenti risultati medi, applicando una topologia *gBest* e considerando i due rispettivi casi: presenza e assenza di smorzamento (Fig. 35 e 36).

Simulazioni	Tempo medio(s)	Iterazioni totali medie
	Con Smorzamento	
w=0.1 ; c_1=c_2=0.5	110.4	194
w=0.2 ; c_1=c_2=0.5	81.7	55
w=0.3 ; c_1=c_2=0.5	70.2	39
w=0.4 ; c_1=c_2=0.5	66.7	33
w=0.5 ; c_1=c_2=0.5	68.2	38
w=0.6 ; c_1=c_2=0.5	71.5	43
w=0.7 ; c_1=c_2=0.5	87.3	48
w=0.8 ; c_1=c_2=0.5	94.5	56
w=0.9 ; c_1=c_2=0.5	98.9	67
w=0.1 ; c_1=c_2=1.5	15.7	27
w=0.2 ; c_1=c_2=1.5	28.9	28
w=0.3 ; c_1=c_2=1.5	58.9	30
w=0.4 ; c_1=c_2=1.5	76.3	39
w=0.5 ; c_1=c_2=1.5	87.55	44
w=0.6 ; c_1=c_2=1.5	118.6	53
w=0.7 ; c_1=c_2=1.5	121.5	62
w=0.8 ; c_1=c_2=1.5	138.8	71
w=0.9 ; c_1=c_2=1.5	156.2	79

Figura 35: Tabella riassuntiva di tempi e iterazioni medie per ogni caso considerato, in presenza di smorzamento, per la funzione sinusoidale

In presenza di smorzamento (Fig. 35), è possibile notare che, in generale, i tempi e le iterazioni medie sono aumentate rispetto alla prima funzione obiettivo: le particelle hanno bisogno di più iterazioni per arrivare all'ottimo globale, data la presenza di più minimi locali. Si può inoltre osservare che:

- Per coefficienti di apprendimento piccoli ($c_1 = c_2 = 0.5$), l'algoritmo è più lento rispetto al caso di $c_1 = c_2 = 1.5$, considerando valori di $w \leq 0.4$; condizione che risulta quasi analoga al caso precedente.
- Rispetto alla prima funzione obiettivo, in questo caso, per valori di $w \geq 0.5$ e coefficienti di apprendimento grandi, l'algoritmo impiega più tempo per arrivare all'ottimo. Una spiegazione può essere ricercata nella dinamica delle particelle: essendo presente lo smorzamento, la componente inerziale diventa sempre più piccola, durante la simulazione, rispetto alle altre componenti. Il passo delle particelle rimane comunque grande, data la presenza dei coefficienti c_1 e c_2, ed essendo presenti più minimi, lo sciame ha maggiori difficoltà a raggiungere il minimo e ha bisogno di più tempo affinché tutte le particelle possano raggrupparsi in un unico punto.

Simulazioni	Tempo medio(s)	Iterazioni totali medie
Senza Smorzamento		
w=0.1 ; c_1=c_2=0.5	151.9	105
w=0.2 ; c_1=c_2=0.5	76.2	62
w=0.3 ; c_1=c_2=0.5	41.5	32
w=0.4 ; c_1=c_2=0.5	48.2	36
w=0.5 ; c_1=c_2=0.5	59.4	47
w=0.6 ; c_1=c_2=0.5	74.2	61
w=0.7 ; c_1=c_2=0.5	141.2	79
w=0.8 ; c_1=c_2=0.5	187.5	115
w=0.9 ; c_1=c_2=0.5	385.2	256
w=0.1 ; c_1=c_2=1.5	32.3	22
w=0.2 ; c_1=c_2=1.5	44.6	28
w=0.3 ; c_1=c_2=1.5	57.4	38
w=0.4 ; c_1=c_2=1.5	68.7	47
w=0.5 ; c_1=c_2=1.5	108.33	57
w=0.6 ; c_1=c_2=1.5	120.96	75
w=0.7 ; c_1=c_2=1.5	240.1	108
w=0.8 ; c_1=c_2=1.5	347.4	191
w=0.9 ; c_1=c_2=1.5	L'algoritmo non arriva a convergenza entro le 500 iterazioni	

Figura 36: Tabella riassuntiva di tempi e iterazioni medie per ogni caso considerato, in assenza di smorzamento, per la funzione sinusoidale

In assenza di smorzamento (Fig. 36), i risultati invece rimangono coerenti con le aspettative: i tempi medi aumentano rispetto al caso in cui è presente il fattore di smorzamento, ma rimangono più piccoli considerando i coefficienti di apprendimento pari a 0.5 e aumentano nel caso di valori pari a 1.5. Così come è accaduto per la prima funzione obiettivo, impostando $w = 0.9$, con $c_1 = c_2 = 1.5$, l'algoritmo non arriva a convergenza entro le iterazioni prefissate: le particelle risultano sparse nel dominio e i passi sono troppo grandi per poter permettere loro di raggrupparsi o comunque avvicinarsi.

4.4 Caso bi-dimensionale con topologia *lBest*

L'ultimo caso che deve essere analizzato è quello di comunicazione locale per il caso bi-dimensionale. Durante le simulazioni è emersa fin da subito la difficoltà dell'algoritmo di arrivare a convergenza.
Si può infatti notare in Figura (37), che, in presenza di smorzamento, l'algoritmo non arriva mai a convergenza. Il risultato non sorprende molto, dato che, come è noto, con una struttura sociale a ruota, la comunicazione è più lenta e l'algoritmo necessita di più iterazioni per arrivare a convergenza.

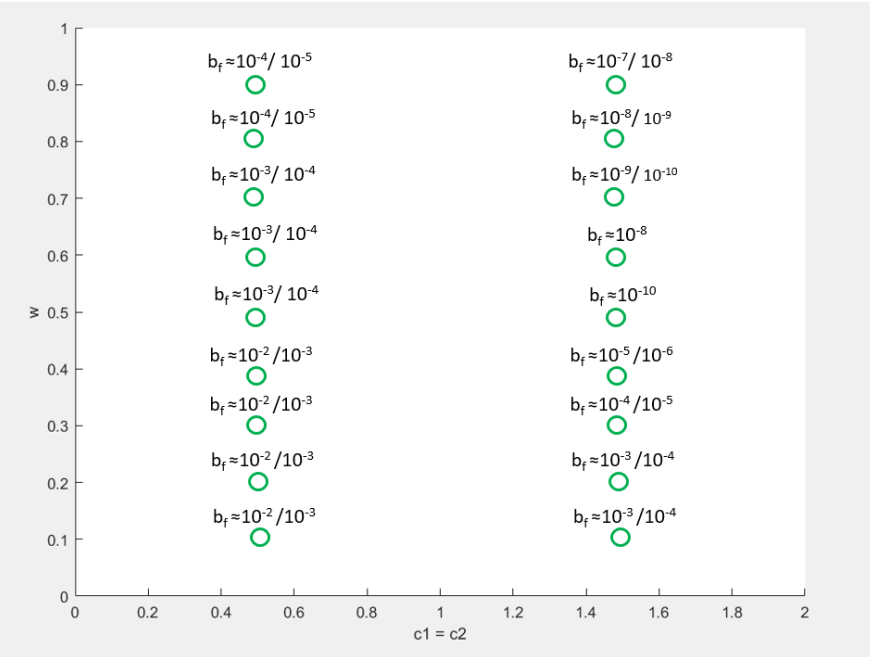


Figura 37: Valori medi per le simulazioni effettuate con topologia ad anello in presenza di smorzamento per la funzione sinusoidale

Quello che appare da subito evidente è che:

- Per valori di c_1 e c_2 piccoli, indipendentemente da w, l'algoritmo raggiunge dei valori compresi tra 10^{-3} e 10^{-5};
- Per valori maggiori di c_1 e c_2, i valori variano e in particolare il valore finale a cui arriva l'algoritmo aumenta con l'aumentare di w.

In assenza di smorzamento, la situazione non è molto diversa (Fig. 38). In questo caso però:

- Per valori dei coefficienti pari a 0.5, all'aumentare di w, aumenta il valore a cui arriva l'algoritmo al termine della simulazione, a tal punto che per $w = 0.9$ lo sciame arriva all'ottimo e converge entro le 500 iterazioni prefissate.
- Per valori dei coefficienti pari a 1.5, gli unici valori di w che permettono all'algoritmo di convergere sono i valori 0.6 e 0.7. Anche in questo caso, analogamente alla precedente funzione di test, per $w = 0.9$ le particelle si muovono all'interno del dominio senza riuscire a raggrupparsi e arrivare a valori prossimi al minimo globale.

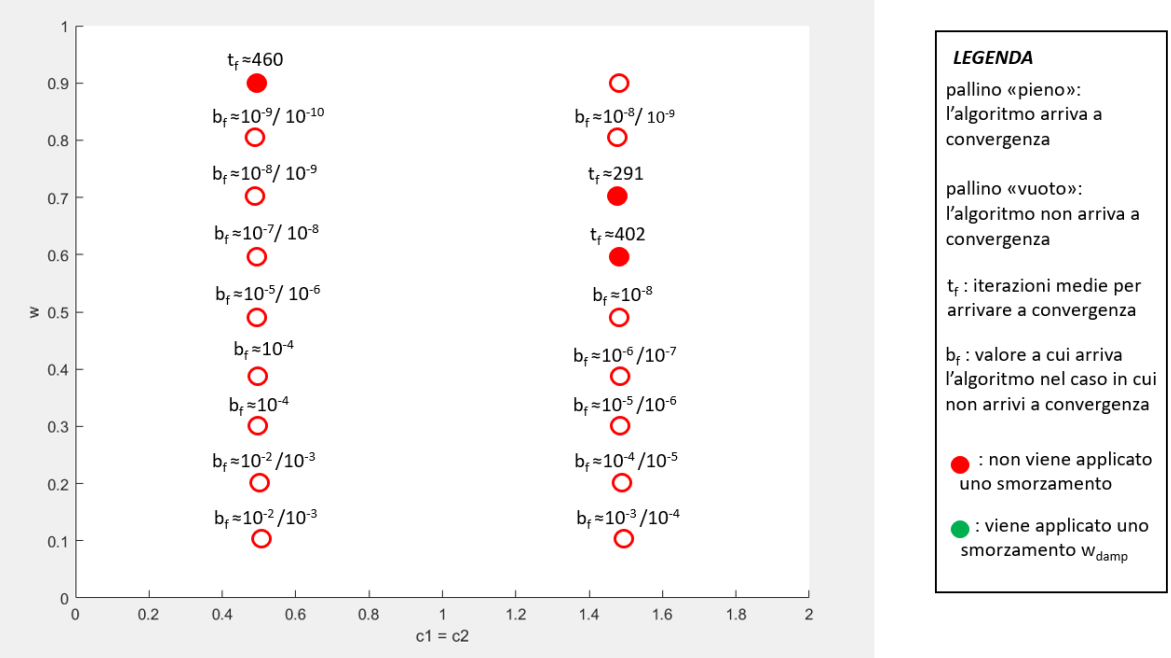


Figura 38: Valori medi per le simulazioni effettuate con topologia ad anello in assenza di smorzamento per la funzione sinusoidale

Conclusioni

Fin dalla sua introduzione nel 1995, l'algoritmo PSO ha subito numerosi miglioramenti e cambiamenti volti a garantirne la convergenza, anche con funzioni obiettivo sempre più complesse.

In questo lavoro di tesi è stato analizzato il comportamento di una versione dell'algoritmo PSO, variando i parametri principali che descrivono la dinamica dello sciame e la tipologia di comunicazione tra le particelle. Sono state effettuate diverse simulazioni per ogni caso considerato, in modo da avere una panoramica generale dei diversi comportamenti dell'algoritmo.

Dai risultati ottenuti è emerso che l'algoritmo di ottimizzazione di sciami di particelle funziona meglio se si imposta una struttura di comunicazione globale: le particelle, condividendo le informazioni con l'intero sciame, riescono ad esplorare meglio l'intero dominio.
Inoltre, una comunicazione locale, soprattutto con topologia ad anello (presa in esame in questo lavoro di tesi), rischia di far rimanere le particelle intrappolate in minimi locali e per riuscire ad arrivare all'ottimo globale sono richieste molte più iterazioni e quindi molto più tempo.
Per quanto riguarda invece i valori da impostare per i parametri che descrivono la dinamica del movimento, si è notato che non esistono valori ottimali che possano valere in generale per tutti i casi presi in esame: la scelta infatti è legata agli altri fattori che descrivono l'algoritmo e, in particolare, alla funzione obiettivo considerata.

Riferimenti bibliografici

[CK02] Maurice Clerc and James Kennedy. The particle swarm-explosion, stability, and convergence in a multidimensional complex space. *IEEE transactions on Evolutionary Computation*, 6(1):58–73, 2002.

[Eng07] Andries P Engelbrecht. *Computational intelligence: an introduction*. John Wiley & Sons, 2007.

[Hep90] Frank Heppner. A stochastic nonlinear model for coordinated bird flocks. *The ubiquity of chaos*, 1990.

[KE95] James Kennedy and Russell Eberhart. Particle swarm optimization. 4:1942–1948, 1995.

[Li09] Xiaodong Li. Niching without niching parameters: particle swarm optimization using a ring topology. *IEEE Transactions on Evolutionary Computation*, 14(1):150–169, 2009.

[LSO21] Xiao-Lin Li, Roger Serra, and Julien Olivier. An investigation of particle swarm optimization topologies in structural damage detection. *Applied Sciences*, 11(11):5144, 2021.

[LvWS15] Qingxue Liu, Barend Jacobus van Wyk, and Yanxia Sun. Small world network based dynamic topology for particle swarm optimization. In *2015 11th International Conference on Natural Computation (ICNC)*, pages 289–294. IEEE, 2015.

[Men04] Rui Mendes. Population topologies and their influence in particle swarm performance. *PhD Final Dissertation, Departamento de Informática Escola de Engenharia Universidade do Minho*, 2004.

[MZHAVD11] A Muñoz-Zavala, Arturo Hernández-Aguirre, and E Villa Diharce. A new neighborhood topology for the particle swarm optimization algorithm. *Association for Computing Machinery: New York, NY, USA*, pages 227–247, 2011.

[PVM03] Thanmaya Peram, Kalyan Veeramachaneni, and Chilukuri K Mohan. Fitness-distance-ratio based particle swarm optimization. pages 174–181, 2003.

[Rey87] Craig W Reynolds. Flocks, herds and schools: A distributed behavioral model. pages 25–34, 1987.

[RRS04] Jacob Robinson and Yahya Rahmat-Samii. Particle swarm optimization in electromagnetics. *IEEE transactions on antennas and propagation*, 52(2):397–407, 2004.

[SE98] Yuhui Shi and Russell Eberhart. A modified particle swarm optimizer. pages 69–73, 1998.

[VDB+07] Frans Van Den Bergh et al. *An analysis of particle swarm optimizers*. PhD thesis, University of Pretoria, 2007.

[VdBE06] Frans Van den Bergh and Andries Petrus Engelbrecht. A study of particle swarm optimization particle trajectories. *Information sciences*, 176(8):937–971, 2006.

[W+18] Wang et al. Particle swarm optimization algorithm: an overview. *Soft Computing*, 22(2):387–408, 2018.