Università di Pisa

Dipartimento di Ingegneria dell'Informazione

Tesi di laurea triennale in Ingegneria Informatica

Ottimizzazione stocastica multi-modale: stato dell'arte

Relatori:
Prof. Marco Cococcioni

Candidato:
Francesco De Lucchini

Anno accademico 2023/2024

Indice

1. Introduzione

La tesi si propone come uno studio dello stato dell'arte nel campo dell'ottimizzazione stocastica multi-modale, in particolare:

- **Ottimizzazione** significa che l'argomento trattato è la massimizzazione [1] o minimizzazione di funzioni reali (*funzioni obiettivo*) a variabili reali su un certo dominio di soluzioni ammissibili (*spazio di decisione*). Tale dominio è dunque un sottoinsieme di R^n, dove n è detta *dimensionalità* del problema
- **Multi-modale** significa che lo studio si focalizza su quei problemi che, per loro natura, hanno più soluzioni almeno localmente ottime, e si è interessati a trovarne il maggior numero possibile [1]
- **Stocastica**, infine, significa che lo studio riguarda quegli algoritmi euristici che, per risolvere i problemi precedentemente descritti, inseriscono aleatorietà nel processo di ottimizzazione, al fine di velocizzarlo, rinunciando alla garanzia matematica del trovare la soluzione ottima globale. Si punta infatti a trovare una soluzione globale *molto buona* ed in *tempi accettabili*

Problemi multi-modali compaiono spesso in ambito ingegneristico, un esempio è il problema del commesso viaggiatore: dato un insieme di città, e note le distanze tra ciascuna coppia di esse, si vuole trovare il tragitto più breve che un commesso viaggiatore deve percorrere per visitare tutte le città una ed una sola volta e ritornare a quella di partenza; è chiaro che ci possono essere più percorsi ottimi (o vicini all'essere ottimi), e che dunque il problema è multi-modale. Un altro esempio sono i problemi di assegnamento, che tendono ad ammettere più soluzioni che raggiungono lo stesso valore ottimo della funzione obiettivo.

Risolvere problemi di questo tipo trovando il maggior numero di soluzioni possibile è importante perché:

- Fornisce ai *decision-makers* più alternative tra le quali applicare criteri di costo secondari [2]
- Aiuta a rivelare la natura dei problemi, favorendo la loro analisi e comprensione [3]
- Aiuta a trovare rapidamente soluzioni alternative in contesti dinamici, ad esempio, in seguito ad un cambiamento del dominio delle soluzioni ammissibili

[1] Solitamente, dato che $\max(f) = -\min(-f)$, i problemi vengono formulati dai matematici come problemi di minimo, senza perdita di generalità

Il sottocampo dell'ottimizzazione dinamica multimodale non verrà tuttavia approfondito in questo lavoro di tesi. Il lettore interessato potrà trovare maggiori dettagli in [3].

2. Algoritmi stocastici

2.1. Introduzione

Come è noto, per la soluzione dei problemi di ottimizzazione NP-hard si tendono ad utilizzare degli algoritmici euristici, nell'intento di trovare una soluzione vicina all'ottimo in un tempo minore rispetto ad un algoritmo di ricerca esatta.

Con lo stesso spirito, nel campo dell'ottimizzazione stocastica nel tempo sono stati introdotte delle *metaeuristiche*: "Una **metaeuristica** può essere vista come una generica strategia di base per la risoluzione stocastica di un problema di ottimizzazione, la quale può essere adattata, con relativamente poche modifiche, ad uno specifico problema" [4].

Ad oggi, alcune delle metaeuristiche più utilizzate sono:

- **Algoritmi Genetici** (GA): Ispirati dal processo di selezione naturale, questi algoritmi implementano i meccanismi biologici di riproduzione, mutazione e selezione per evolvere una "popolazione di soluzioni"
- **Particle Swarm Optimization** (PSO): Uno "sciame di particelle" esplora lo spazio di decisione simulando il comportamento sociale degli uccelli o dei pesci, ogni particella, infatti, regola la propria posizione basandosi sia sull'esperienza propria che su quella delle vicine
- **Ant Colony Optimization** (ACO): È ispirato dal comportamento delle formiche, che, per orientarsi, depositano ferormoni sul percorso che compiono, guidando le formiche successive verso soluzioni favorevoli. Questo approccio risulta particolarmente adatto per risolvere problemi di ottimizzazione discreta su grafi

Gli algoritmi stocastici sopracitati appartengono alla categoria dei cosiddetti **algoritmi evolutivi**, in quanto evolvono nel tempo un'intera "popolazione di soluzioni", imitando processi che avvengono in natura. Come vedremo più avanti, evolvere una popolazione intera di soluzioni anziché una singola soluzione, è vantaggioso sia nell'ottimizzazione multi-obiettivo che in quella multi-modale.

Un altro aspetto interessante degli algoritmi evolutivi è che spesso bisogna fare ben poche ipotesi sulla funzione obiettivo e sullo spazio di decisione del problema, la maggior parte di questi, infatti, funziona anche in caso di funzioni obiettivo non lineari, non convesse, non derivabili e persino non continue [5].

2.2. Algoritmi genetici

Gli algoritmi genetici, i più studiati tra quelli evolutivi, sono processi iterativi che, ad ogni iterazione, producono una nuova popolazione di soluzioni ammissibili, solitamente migliore della precedente, massimizzando una **funzione di fitness**.

Tale funzione, che solitamente coincide con la funzione obiettivo stessa o con una sua trasformazione, misura la qualità di ogni individuo nella popolazione, in altre parole esprime quantitativamente la *bontà* di una soluzione per un particolare problema.

Metaforicamente, gli individui di una popolazione corrispondono al concetto biologico dei **cromosomi**, ognuno dei quali contiene un numero tipicamente fisso di **geni**, che possono assumere valori (**alleli**) diversi.

La codifica dei cromosomi, ossia il numero di geni che contengono e come questi vengono implementati in un calcolatore, dipende sempre dal particolare problema che si vuole risolvere.

In generale, una codifica, per essere considerata adatta ad un particolare problema, deve:
- Fare in modo che ogni punto dello spazio di decisione sia raggiungibile
- Coprire il meno possibile lo spazio esterno a quello di decisione
- Rispettare il principio di località [6]: piccoli cambiamenti nel **genotipo** devono portare a piccoli cambiamenti nel **fenotipo**, ad esempio, invertire un bit in un cromosoma deve spostarlo di *poco* nello spazio di decisione

Tipicamente, i geni vengono codificati in forma **binaria**, dunque i possibili alleli sono 0 ed 1, ed i cromosomi sono di conseguenza stringhe di bit.

È tuttavia possibile implementare anche codifiche ottali, esadecimali e **reali**, ad esempio, per minimizzare una funzione di tre variabili reali $f(x, y, z)$ utilizzando un algoritmo genetico, si potrebbe scegliere la seguente implementazione:
- Ogni cromosoma è un vettore di numeri reali e rappresenta un punto (x, y, z) nello spazio di decisione
- Ogni gene, ossia ogni elemento di tale vettore, rappresenta il particolare valore di x, y o z
- La funzione di fitness è la funzione f stessa

Rappresentazioni di questo tipo sono particolarmente utilizzate nella ricerca dei pesi ottimali delle reti neurali [7].

Un'ulteriore rappresentazione, utilizzata per risolvere problemi di ottimizzazione **combinatoria**, consiste nell'implementare i geni come numeri interi, ognuno dei quali rappresenta una posizione in una determinata sequenza. Ad esempio, riprendendo il problema iniziale del commesso viaggiatore, un possibile cromosoma potrebbe essere [1, 4, 2, 3], ossia l'ordine in cui visitare le varie città (4 in questo caso).

Infine, è anche possibile utilizzare rappresentazioni ad albero per modellare in un singolo cromosoma un intero algoritmo, approccio che prende il nome di **genetic programming**.

Una possibile applicazione per rappresentazioni di questo tipo è la ricerca di strategie ottime nell'ambito del trading finanziario [8]. A questo proposito, un potenziale cromosoma potrebbe essere quello illustrato in Figura 1, il quale indica: "Se il prezzo è minore della media degli ultimi 30 giorni allora compra un determinato titolo, altrimenti vendilo"

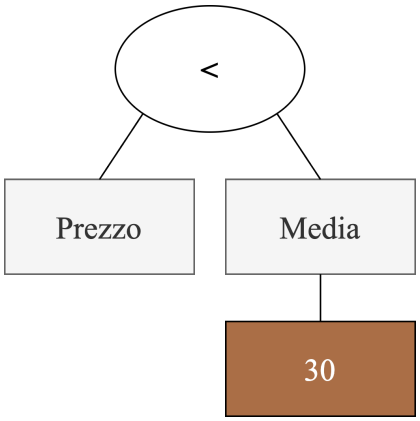


Figura 1: Possibile cromosoma per un algoritmo di *genetic programming* nell'ambito del trading finanziario

Ogni iterazione di un algoritmo genetico produce una *generazione*, ossia una popolazione di soluzioni, idealmente migliore della precedente. La prima generazione viene spesso [2] inizializzata casualmente, mentre tutte le successive vengono derivate dalle rispettive precedenti per mezzo dei seguenti passi [7]:

1. Selezione
2. Riproduzione
3. Mutazione
4. Rimpiazzamento

2.2.1. Selezione

Lo scopo della selezione è quello di permettere solamente agli individui considerati migliori di riprodursi, in modo da favorire l'aumento globale della fitness e, idealmente, preservare anche la diversità della popolazione (per esplorare il più possibile lo spazio di decisione).

Le principali tecniche di selezione, tratte da [7], [10], [11], sono:

- **Roulette wheel selection**: simula una ruota girevole dove ad ogni cromosoma viene assegnato un settore di dimensioni proporzionali alla sua fitness, più formalmente, il cromosoma x_i ha probabilità di essere estratto (anche più volte) $p_i = \frac{f(x_i)}{F}$, dove $F = \sum f(x_i)$ è la fitness globale
- **Stochastic universal sampling**: è una variante della tecnica sopracitata nella quale, metaforicamente, la ruota viene fatta girare una volta sola, gli individui selezionati per la riproduzione sono quello estratto più tutti quelli ad intervalli equidistanti da quest'ultimo, come mostrato in Figura 2
- **Stochastic remainder selection**: può essere considerata come un'ulteriore variante della *roulette wheel selection*, in particolare, sia n_i il risultato (in genere decimale) della moltiplicazione tra la probabilità p_i che il cromosoma x_i ha di essere estratto ed il numero N di individui della popolazione, allora, $\lfloor n_i \rfloor$ cromosomi di tipo x_i vengono direttamente selezionati per la riproduzione, mentre un ulteriore cromosoma x_i viene selezionato con probabilità $n_i - \lfloor n_i \rfloor$

[2] Un'alternativa all'inizializzazione casuale, computazionalmente più onerosa, che però garantisce una maggiore diversità della popolazione, è nota come **latin hypercube sampling** [9], e consiste nell'inizializzare la popolazione in modo tale che ogni possibile iperpiano parallelo ad un asse in cui si può dividere lo spazio di decisione contenga un unico individuo. In due dimensioni, questo processo può essere visto come il cercare di posizionare delle torri su una scacchiera in modo che queste non si attacchino a vicenda

- **Rank selection**: la probabilità di selezione di un particolare individuo non dipende direttamente dalla sua fitness, ma dal posizionamento che questa ha rispetto alla fitness di tutti gli individui.
- **Tournament selection**: vengono scelti casualmente k [3] individui dalla popolazione, e quello con fitness più alta tra questi viene selezionato per la riproduzione (il processo è ripetuto finché desiderato)

In generale, gli approcci di tipo proporzionale come *roulette wheel selection* e le sue varianti, tendono a convergere prematuramente ad una singola soluzione, e per questo motivo non sono particolarmente utilizzati [11]. *Rank selection* è computazionalmente più oneroso delle varianti di *roulette wheel selection* e di *tournament selection*, ma fornisce anche agli individui peggiori una possibilità realistica di essere selezionati per la riproduzione e dunque di migliorare [13]. *Tournament selection* è invece considerato un ottimo compromesso perché è semplice, veloce e facile da implementare in parallelo [11].

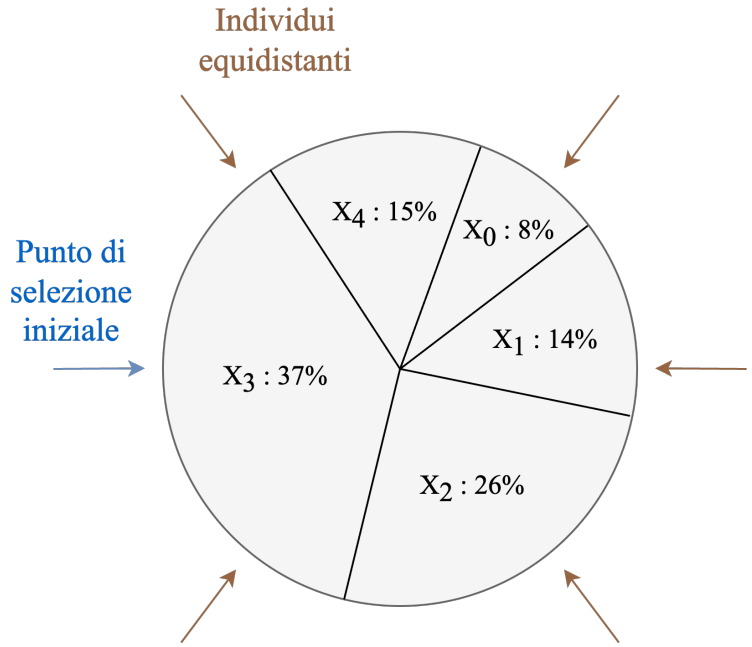


Figura 2: Illustrazione dello *stochastic universal sampling*

2.2.2. Riproduzione

Una volta completato il processo di selezione, ogni coppia di genitori scelti casualmente tra quelli selezionati precedentemente genera, con una certa probabilità, uno o più figli. Le principali modalità attraverso le quali avviene la ricombinazione genetica, tratte da [7], [14], [15], sono:

[3] Il valore più comune di k è 2 [12]

- Per cromosomi con codifica *binaria*:
  - **Single-point crossover**: sia L la lunghezza in bit dei cromosomi e k un numero casuale compreso tra 1 ed $L-1$, allora il primo figlio eredita i bit fino a k-esimo (compreso) dal primo genitore ed i restanti dal secondo, mentre il secondo figlio fa esattamente il contrario

    Ad esempio:

    $$G_1 = [010\ 1110], G_2 = [011\ 0101], L = 7, k = 3$$

    $$F_1 = [010\ 0101]$$

    $$F_2 = [011\ 1110]$$
  - **K-point crossover**: è equivalente ad eseguire k volte il single-point crossover, scegliendo ogni volta un punto di crossover (k) diverso
  - **Uniform crossover**: ogni gene viene ereditato dal primo o dal secondo genitore con una certa probabilità prefissata
- Per cromosomi con codifica *reale*:
  - **Convex crossover**: i geni i-esimi dei figli F_1, F_2 sono combinazione convessa dei geni i-esimi dei genitori G_1, G_2:

    $$F_{1_i} = kG_{1_i} + (1-k)G_{2_i}$$

    $$F_{2_i} = kG_{2_i} + (1-k)G_{1_i}$$

    Per essere una combinazione convessa, k dovrebbe essere preso nell'intervallo $[0, 1]$, tuttavia, spesso viene scelto in $[-d, d+1]$, con $d \approx 0.25$, per combattere la tendenza che hanno i geni di ridurre il proprio valore nel tempo, presente quando $d = 0$ [15]
  - **Simulated binary crossover (SBX)**: Simula il comportamento del single point crossover attraverso il seguente algoritmo:

    1. Genera un numero casuale $u \in [0, 1)$
    2. Scegli l'indice di distribuzione η_c [4] e calcola:

$$\beta = \begin{cases} 2u^{\frac{1}{\eta_c+1}} & \text{se } u \leq 0.5 \\ \left(\frac{1}{2(1-u)}\right)^{\frac{1}{\eta_c+1}} & \text{altrimenti} \end{cases}$$

3. Genera i figli come:

$$F_1 = 0.5[(1+\beta)G_1 + (1-\beta)G_2]$$

$$F_2 = 0.5[(1-\beta)G_1 + (1+\beta)G_2]$$

Ad esempio:

$$G_1 = [3.465], G_2 = [7.210], u = 0.591, \eta_c = 2$$

$$\beta = \left(\frac{1}{2(1-u)}\right)^{\frac{1}{\eta_c+1}} = \left(\frac{1}{2(1-0.591)}\right)^{\frac{1}{2+1}} \approx 1.069$$

$$F_1 = 0.5[(1+\beta)G_1 + (1-\beta)G_2]$$
$$= 0.5[(1+1.069)3.465 + (1-1.069)7.210] \approx [3.336]$$

$$F_2 = 0.5[(1-\beta)G_1 + (1+\beta)G_2]$$
$$= 0.5[(1-1.069)3.465 + (1+1.069)7.210] \approx [7.339]$$

- Per cromosomi con codifica *combinatoria*:
  - **Partially matched crossover (PMX)**:

    Per semplicità questa tecnica viene illustrata direttamente attraverso un esempio:

$$G_1 = [1, 2, 3, 4, 5, 6, 7, 8]$$

$$G_2 = [3, 7, 5, 1, 6, 8, 2, 4]$$

1. Seleziona casualmente un segmento di geni dal primo genitore e copialo nel figlio:

$$G_1 = [1, 2, 3, 4, 5, 6, 7, 8]$$

$$F = [-, -, -, 4, 5, 6, -, -]$$

3. Per ogni gene m appartenente allo stesso segmento nel secondo genitore $[..., 1, 6, 8, ...]$, sia n il gene già presente nel figlio nella medesima posizione, allora copia m nella posizione in cui si trova n nel secondo genitore (se m è già copiato vai avanti, se la posizione è già occupata allora n diventa l'elemento che occupa tale posizione):

[4] Valori alti di η_c tendono a generare figli vicini ai genitori, valori bassi di η_c il contrario, un valore comune di η_c è 2 [14]

- $m = 1, n = 4$: 8° posizione in $G_2 \to F_1 = [-, -, -, 4, 5, 6, -, 1]$
- $m = 6 \to$ già copiato
- $m = 8, n = 6$: 5° posizione in G_2, già occupata da 5, allora $n = 5$: 3° posizione in $G_2 \to F_1 = [-, -, 8, 4, 5, 6, -, 1]$

4. Copia, nella prima posizione disponibile del figlio, i geni del secondo genitore non ancora ereditati:
   - $3 \to F_1 = [3, -, 8, 4, 5, 6, -, 1]$
   - $7 \to F_1 = [3, 7, 8, 4, 5, 6, -, 1]$
   - $2 \to F_1 = [3, 7, 8, 4, 5, 6, 2, 1]$
5. Scambia i ruoli dei due genitori per ottenere un secondo figlio

2.2.3. Mutazione

La prole generata precedentemente subisce, con una certa probabilità , una mutazione, con l'obbiettivo di introdurre diversità nella popolazione e favorire l'esplorazione dello spazio di decisione.

Gli operatori di mutazione principali, tratti da [7], [16], sono:

- Per cromosomi con codifica *binaria*:
  - **Bit-string mutation**: scelto un bit casuale del cromosoma, si parla di *mutazione forte* se questo viene invertito ($0 \to 1, 1 \to 0$), e di *mutazione debole* se questo viene sostituito da uno generato casualmente (potrebbe dunque non cambiare)
- Per cromosomi con codifica *reale*:
  - **Polynomial mutation**: viene utilizzata una distribuzione di probabilità polinomiale per perturbare lievemente un cromosoma:
    1. Genera una numero casuale $u \in [0, 1)$
    2. Scegli l'indice di distribuzione η_c [5] e calcola:

$$\beta = \begin{cases} (2u)^{\frac{1}{\eta_c+1}} - 1 & \text{se } u \leq 0.5 \\ 1 - (2(1-u))^{\frac{1}{\eta_c+1}} & \text{altrimenti} \end{cases}$$

    3. Perturba il cromosoma x come:

$$x' = \begin{cases} x + \beta(x - x_L) & \text{se } u \leq 0.5 \\ x + \beta(x_U - x) & \text{altrimenti} \end{cases}$$

dove x_L ed x_U sono rispettivamente lower ed upper bound della variabile x

Ad esempio:

$$x = [4.512], x_L = 1, x_U = 6, u = 0.348, \eta_c = 20$$

$$\beta = (2u)^{\frac{1}{\eta_c+1}} - 1 = (2 \cdot 0.348)^{\frac{1}{20+1}} - 1 \approx -0.017$$

$$x' = x + \beta(x - x_L) = 4.512 - 0.016(4.512 - 1) \approx [4.456]$$

In generale, questo approccio può essere utilizzato con diverse distribuzioni di probabilità, ad esempio quella Gaussiana, di Cauchy, o di Lévy [17], [18].

- Per cromosomi con codifica *combinatoria*:
  - **Right rotate**: viene scelto un segmento casuale di geni (considerando anche la possibilità di fare *wrap-around*) e ruotato di k posizioni, con k compreso tra 0 e la dimensione del cromosoma

  Ad esempio:

$$X_1 = (1, 4, 5, 3, 6, 2), k = 1 \rightarrow X_1 = (2, 1, 5, 3, 4, 6)$$

2.2.4. Rimpiazzamento

La nuova popolazione, che in genere conterrà lo stesso numero di individui della precedente, può essere composta in diversi modi [12]:

- **Delete-all**: è la tecnica più utilizzata, rimpiazzare tutti gli individui dalla popolazione con quelli appena prodotti per riproduzione e mutazione
- **Steady-state**: come prima, ma il rimpiazzamento è limitato a k individui, scelti casualmente, scelti tra i peggiori, oppure scelti tra i quelli selezionati per la riproduzione

Si può infine notare che, con le tecniche illustrate finora, non è sicuro che gli individui migliori di una popolazione vengano selezionati per la riproduzione, tantomeno che sopravvivano invariati dopo riproduzione e mutazione. Una possibile opzione, nota come **elitism selection**, è quella di includere automaticamente i migliori k individui di ogni generazione nella successiva.

[5] Un valore adeguato per la maggior parte dei problemi è $\eta_c \in [20, 100]$ [19]

2.2.5. Condizioni di terminazione

Per finire, alcune comuni condizioni di terminazione sono:

- È stato prodotto un numero massimo di generazioni
- È stato raggiunto un numero massimo di valutazioni della funzione di fitness
- È stato raggiunto un tempo di calcolo massimo
- È stata trovata almeno una soluzione che soddisfa certe condizioni di qualità
- Il valore di fitness dell'individuo migliore non è aumentato in nessuna delle ultime k iterazioni
- Una certa percentuale di individui ha raggiunto un valore di fitness simile a quello dell'individuo migliore

2.2.6. Stato dell'arte

L'algoritmo genetico di base illustrato precedentemente è stato oggetto di numerosi studi nel corso degli anni, alcune tra le varianti ed i miglioramenti più significativi proposti sono riassunti nel seguente elenco:

- Per loro natura, gli algoritmi genetici, e più in generale quelli evolutivi, sono facilmente **parallelizzabili** su più processori, ad esempio, basti pensare all'operazione di valutazione della funzione di fitness [20]
- Spesso, al termine del rimpiazzamento, viene avviato un processo di **ricerca locale** [6], al fine di assicurarsi che ogni individuo si trovi in un ottimo locale. Gli algoritmi genetici che fanno uso di questa tecnica vengono chiamati **memetic algorithms** [21]
- Il processo di inizializzazione casuale della popolazione potrebbe non essere sempre il più adeguato, idealmente questo dovrebbe essere implementato *ad hoc* conoscendo le caratteristiche del problema da risolvere [22]. Lo stesso concetto vale per gli operatori di riproduzione: quelli standard funzionano sempre, ma, a seconda del problema, potrebbero esserci euristiche migliori [23]. In generale, dunque, è richiesto avere una **buona conoscenza delle caratteristiche problema** per poter ottenere le prestazioni migliori da un algoritmo genetico
- Spesso, in problemi complessi del mondo reale (ad esempio quelli di ottimizzazione strutturale [7]), la funzione di fitness è estremamente costosa, si può dunque trarre beneficio dal sostituirla con una sua approssimazione meno onerosa, processo noto come **evaluation relaxation** [12]
- Le probabilità di riproduzione e di mutazione possono, a seconda del problema, influenzare notevolmente l'accuratezza e la velocità di convergenza di un

algoritmo genetico. Per questo motivo sono stati studiati **algoritmi genetici adattivi**, i quali, basandosi su informazioni relative alla popolazione (ad esempio i valori di fitness), variano ad ogni generazione le probabilità di riproduzione e di mutazione, in modo da mantenere la diversità della popolazione ed aumentare la capacità di convergenza [24]

[6] Un esempio di processo di ricerca locale è il classico algoritmo di **hill climbing**: il cromosoma viene perturbato lievemente in modo casuale, al fine di trovare, in un suo intorno, un punto con funzione di fitness più alta. Sono tuttavia possibili alternative più sofisticate [25]

[7] Un celebre esempio è quello dell'*evolved antenna* [26]

3. Niching

3.1. Significato del termine

Come anticipato nell'introduzione, la difficoltà principale dei problemi multi-modali è quella di trovare il maggior numero possibile di soluzioni almeno localmente ottime. Gli algoritmi classici, data la loro natura deterministica, performano male in questo tipo di problemi, spesso infatti garantiscono la convergenza ad una singola soluzione almeno localmente ottima, e bisognerebbe dunque eseguirli più volte con condizioni iniziali differenti per anche solo sperare di ottenere soluzioni diverse.

Gli algoritmi evolutivi, invece, operando con una popolazione di soluzioni, hanno un vantaggio naturale nei problemi multi-modali, riuscendo, con qualche accorgimento, a trovare più soluzioni in una singola esecuzione [27].

Tali accorgimenti sono i cosiddetti meccanismi di **niching** [28], ossia meccanismi che permettono di trovare e mantenere diverse "zone interessanti" dello spazio di decisione, evitando di convergere in una singola soluzione.

3.2. Niching per algoritmi genetici

Il primo meccanismo di niching applicato con successo ad un algoritmo genetico fu quello basato sul concetto di *fitness sharing*, risalente al lavoro svolto da Goldberg e Richardson [29]. Successivamente, negli anni, sono state proposte numerose tecniche di niching, raccolte ed analizzate in altrettanti numerosi studi comparativi [30]–[32].

Ad oggi, le tecniche di niching più utilizzate sono:
- Clearing
- Clustering
- Crowding
- Restricted tournament selection
- Sharing
- Species-Conserving Genetic Algorithms

3.2.1. Clearing

Introdotto in [33], è un approccio ispirato al meccanismo naturale della condivisione di risorse limitate tra individui di una stessa sottopopolazione. Tuttavia, anziché essere divise equamente, nella procedura di *clearing* le risorse vengono assegnate interamente all'individuo migliore.

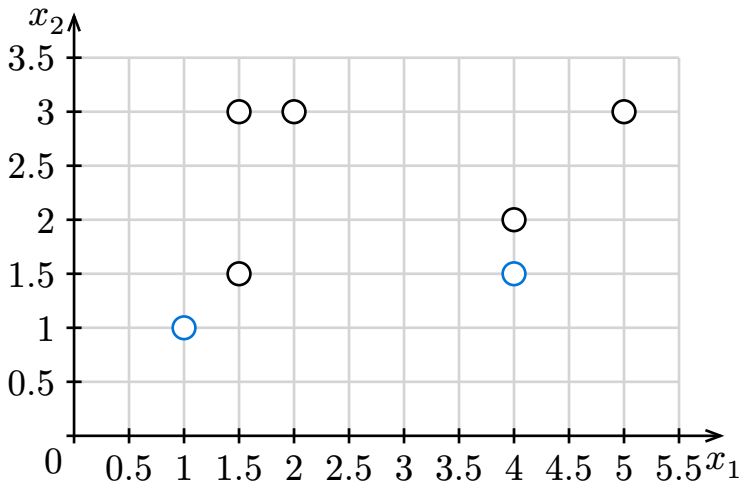
Tale procedura consiste nei seguenti passaggi:

1. Ordina la popolazione in ordine decrescente di fitness
2. Isola i migliori k individui (la loro fitness rimane invariata)
3. Azzerata la fitness a tutti gli individui restanti che distano [8] meno di σ_{clear} da (almeno) uno dei migliori k
4. Riparti dal secondo passaggio, considerando questa volta i successivi migliori k individui. Il processo termina quando sono stati considerati tutti gli individui della popolazione

Per fare un esempio semplice, nel caso di ottimizzazione di una funzione di due variabili reali, utilizzando $k = 2$ e $\sigma_{\text{clear}} = 1$, il processo di clearing ha il seguente effetto:

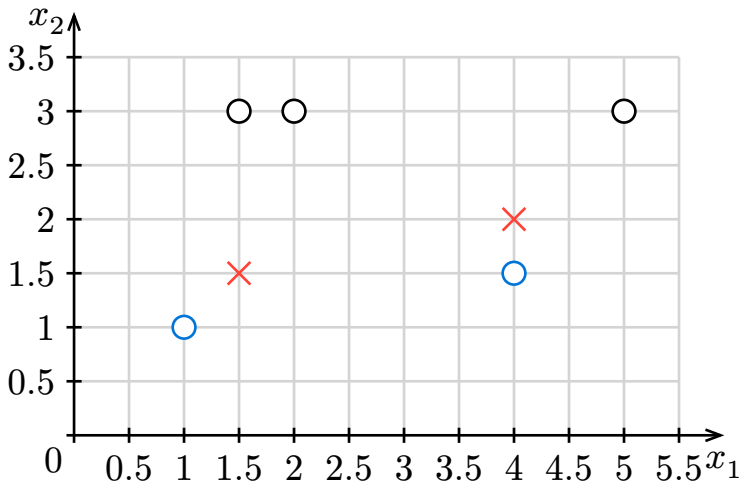
- Prima iterazione

Individuo	Fitness
$X_0 = (1.0, 1.0)$	15.6
$X_1 = (4.0, 1.5)$	12.0
$X_2 = (5.0, 3.0)$	11.3
$X_3 = (4.0, 2.0)$	9.8
$X_4 = (2.0, 3.0)$	7.9
$X_5 = (1.5, 3.0)$	6.1
$X_6 = (1.5, 1.5)$	5.2



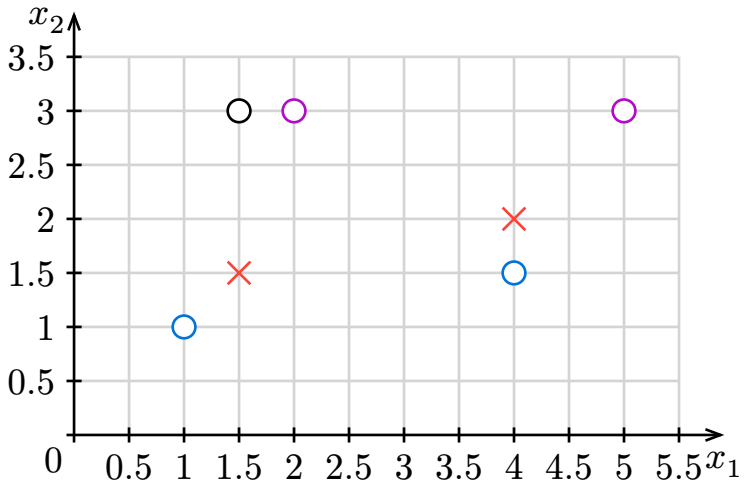
[8] Il concetto di "distanza" nello spazio di decisione dipende dalla codifica utilizzata per i cromosomi, ad esempio, nel caso reale può essere quella Euclidea, e nel caso binario può essere quella di Hamming

Individuo	Fitness
$X_0 = (1.0, 1.0)$	15.6
$X_1 = (4.0, 1.5)$	12.0
$X_2 = (5.0, 3.0)$	11.3
$X_3 = (4.0, 2.0)$	0.0
$X_4 = (2.0, 3.0)$	7.9
$X_5 = (1.5, 3.0)$	6.1
$X_6 = (1.5, 1.5)$	0.0

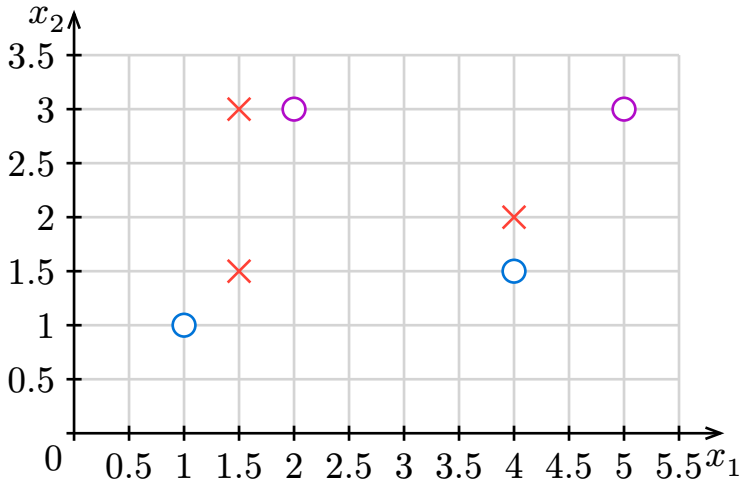


- Seconda iterazione

Individuo	Fitness
$X_0 = (1.0, 1.0)$	15.6
$X_1 = (4.0, 1.5)$	12.0
$X_2 = (5.0, 3.0)$	11.3
$X_3 = (4.0, 2.0)$	0.0
$X_4 = (2.0, 3.0)$	7.9
$X_5 = (1.5, 3.0)$	6.1
$X_6 = (1.5, 1.5)$	0.0



Individuo	Fitness
$X_0 = (1.0, 1.0)$	15.6
$X_1 = (4.0, 1.5)$	12.0
$X_2 = (5.0, 3.0)$	11.3
$X_3 = (4.0, 2.0)$	0.0
$X_4 = (2.0, 3.0)$	7.9
$X_5 = (1.5, 3.0)$	0.0
$X_6 = (1.5, 1.5)$	0.0



Il fatto che questa tecnica azzeri completamente la fitness degli individui vicini ad uno con fitness superiore, fa sì che questi, pur occupando posti nella popolazione, non contribuiscano al suo sviluppo (non verranno infatti scelti per la riproduzione).

Una possibile variante, computazionalmente più onerosa, è nota come **modified clearing** [30]:

1. Esegui la procedura di *clearing* standard
2. Per ogni individuo con fitness nulla, sia questo X, controlla se ricade entro $1.5 \cdot \sigma_{\text{clear}}$ da uno con fitness non nulla, sia quest'ultimo Y, se così è allora sposta X in modo casuale ad una distanza compresa tra $1.5 \cdot \sigma_{\text{clear}}$ e $3.0 \cdot \sigma_{\text{clear}}$ da Y, come mostrato in Figura 3, e ricalcola la sua fitness

Spostare gli individui "inutili" e rivalutarne la fitness non solo evita di sprecare posti nella popolazione, ma aiuta ad esplorare lo spazio di decisione.

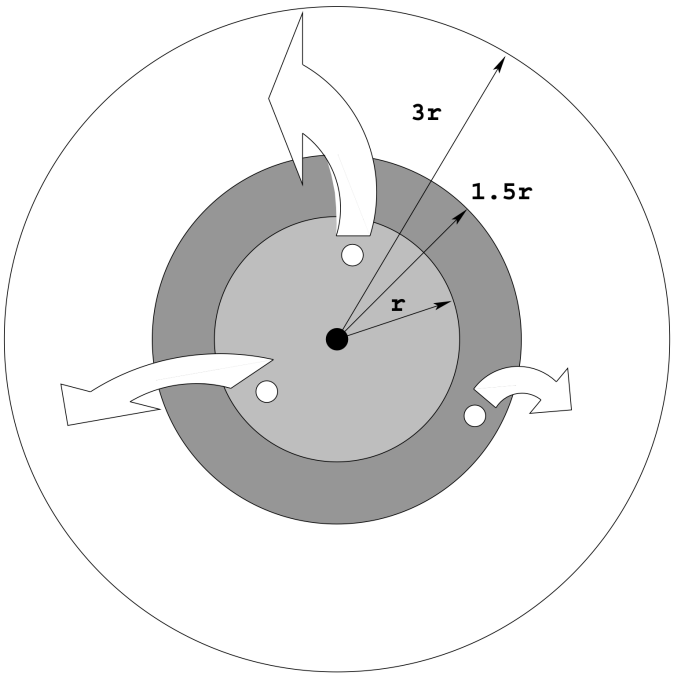


Figura 3: Illustrazione della tecnica del *modified clearing* [30]

3.2.2. Clustering

Introdotto in [34], utilizza una versione adattiva dell'algoritmo K-means, basata sui parametri $d_{\min}$ e $d_{\max}$, per dividere la popolazione in cluster (niches):

- $d_{\min}$ è la distanza minima consentita tra due centroidi di due cluster diversi
- $d_{\max}$ è la distanza massima consentita tra un centroide di un cluster ed un elemento del cluster stesso

Successivamente all'identificazione dei cluster, la fitness degli individui viene aggiornata nel seguente modo:

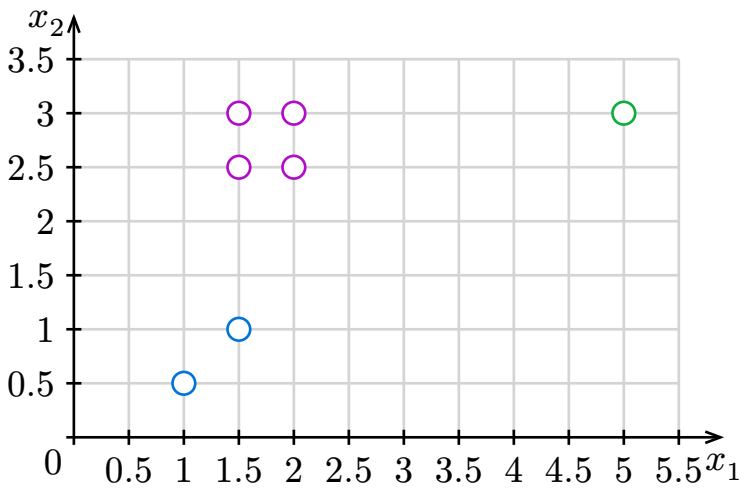
$$f(x_i)' = \frac{f(x_i)}{n_c\left(1 - \left(\frac{d(i,c)}{2d_{\max}}\right)^{\alpha}\right)}$$

Dove:

- $f(x_i)$ è la fitness dell'individuo x_i precedente all'aggiornamento
- n_c è il numero di individui appartenenti allo stesso cluster di x_i
- $d(i,c)$ è la distanza tra l'individuo x_i ed il centroide del cluster al quale appartiene
- α è una costante

Considerando un esempio simile al precedente, ponendo $\alpha = 0.5$ e $d_{\max} = 1$, possiamo osservare come questo approccio penalizzi la fitness degli individui concentrati nella stessa zona dello spazio di decisione, promuovendone dunque l'esplorazione:

	n_c	$f(x_i)$		$f'(x_i)$
X_0	2	15.6	$\approx$	13.46
X_1	4	14.5	$\gg$	6.25
X_2	1	11.3	$=$	11.3
X_3	4	10.9	$\gg$	4.70
X_4	4	7.9	$\gg$	3.41
X_5	4	6.1	$\gg$	2.63
X_6	2	5.2	$\approx$	4.49



3.2.3. Crowding

Introdotta in [35], e successivamente migliorata in [36], la tecnica del *crowding* è composta dai seguenti passaggi, da ripetere $\frac{N}{2}$ volte (dove N è il numero di individui nella popolazione):

1. Seleziona, senza rimpiazzamento, due individui casuali dalla popolazione, siano questi G_1, G_2
2. Applica l'operatore di riproduzione a G_1, G_2, generando F_1, F_2
3. Applica l'operatore di mutazione a F_1, F_2, generando F_1', F_2'
4. Se $d(G_1, F_1') + d(G_2, F_2') \leq d(G_1, F_2') + d(G_2, F_1')$ allora:
   - Se $f(F_1') \geq f(G_1)$ rimpiazza G_1 con F_1'
   - Se $f(F_2') \geq f(G_2)$ rimpiazza G_2 con F_2'

   Altrimenti:
   - Se $f(F_2') \geq f(G_1)$ rimpiazza G_1 con F_2'
   - Se $f(F_1') \geq f(G_2)$ rimpiazza G_2 con F_1'

Si può subito notare che, mentre le due tecniche precedenti agiscono sulla fitness degli individui, e possono dunque essere utilizzate con qualsiasi operatore di selezione e di rimpiazzamento, la tecnica del *crowding* agisce sull'algoritmo in sé, imponendo un particolare processo di selezione e di rimpiazzamento.

Sostanzialmente, questa tecnica assicura che gli individui figli competano direttamente con il genitore più simile, mantenendo dunque la diversità della popolazione e riducendo il rischio di convergere prematuramente ad una singola soluzione.

La tecnica sopracitata è nota come **deterministic crowding** perché gli individui con fitness alta vincono sempre (dunque in modo deterministico) contro quelli con fitness bassa, questo comportamento, tuttavia, potrebbe portare, in seguito ad una serie di confronti sfavorevoli, alla perdita di niches.

Per questo motivo, è stata introdotta in [37] una variante a tale approccio, nota come **probabilistic crowding**, nella quale il rimpiazzamento di un generico individuo G rispetto ad un secondo individuo F avviene con la seguente probabilità:

$$p(F \text{ rimpiazza } G) = \frac{f(F)}{f(F) + f(G)}$$

3.2.4. Restricted tournament selection

Introdotto in [38], questo approccio, per certi versi simile a quello del *crowding*, agisce sugli operatori di selezione e di rimpiazzamento, impedendo che individui troppo distanti competano per un posto nella popolazione.

Tale procedura è composta dai seguenti passaggi:

1. Seleziona, senza rimpiazzamento, due individui casuali dalla popolazione, siano questi G_1, G_2
2. Applica l'operatore di riproduzione a G_1, G_2, generando F_1, F_2
3. Applica l'operatore di mutazione a F_1, F_2, generando F_1', F_2'
4. Per ogni figlio F_i' generato, scegli casualmente k individui dalla popolazione e, tra questi k, seleziona quello più vicino ad F_i', sia questo F_i''
5. Gli individui accettati nella popolazione successiva sono quelli con fitness maggiore tra F_1' ed F_1'' e tra F_2' ed F_2''. In alternativa, si può anche utilizzare un approccio non deterministico, dove F_i' ed F_i'' hanno ognuno probabilità di "vincere" proporzionale alla propria fitness

3.2.5. Sharing

Inizialmente proposto in [29], è un approccio che, come il *clearing*, si ispira al meccanismo naturale della condivisione di risorse limitate tra individui di una stessa sottopopolazione.

La tecnica impone che la fitness di ogni individuo venga aggiornata ad ogni generazione attraverso la seguente equazione:

$$f(x_i)' = \frac{f(x_i)}{n_i}$$

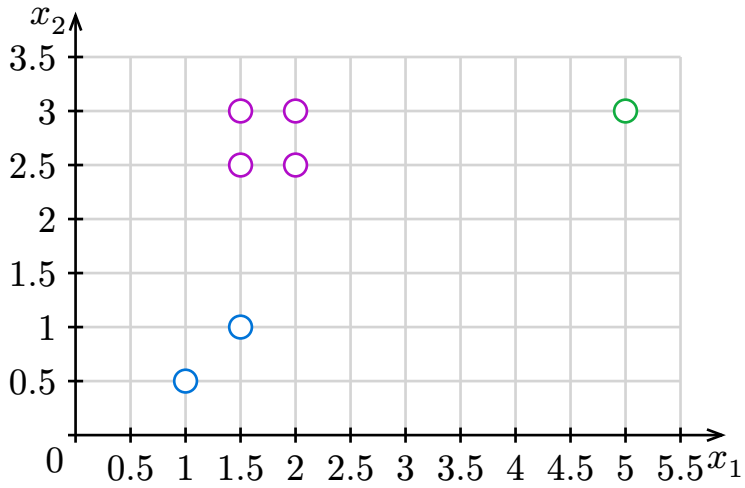
$$n_i = \sum_{j=1}^{N} \mathrm{Sf}(d(i,j))$$

$$\mathrm{Sf}(d(i,j)) = \begin{cases} 1 - \left(\frac{d(i,j)}{\sigma_{\text{share}}}\right)^{\alpha} \text{ se } d(i,j) < \sigma_{\text{share}} \\ 0 \text{ altrimenti} \end{cases}$$

Dove:
- Sf è la *sharing function*

- σ_{share} è la distanza entro la quale due individui sono considerati vicini, e dunque in condivisione
- α è una costante

Considerando l'esempio introdotto nella trattazione del *clustering*, ponendo $\alpha = 0.5$ e $\sigma_{\text{share}} = 1$, si possono trarre le stesse conclusioni, ossia che la tecnica di *sharing* penalizza la fitness degli individui concentrati nella stessa zona dello spazio di decisione, promuovendone dunque l'esplorazione:



Individuo	n_i	$f(x_i)$		$f'(x_i)$
$X_0 = (1.0, 0.5)$	1.16	15.6	$\approx$	13.45
$X_1 = (1.5, 2.5)$	1.74	14.5	$\gg$	8.33
$X_2 = (5.0, 3.0)$	1.00	11.3	$=$	11.3
$X_3 = (2.0, 2.5)$	1.74	10.9	$\gg$	6.26
$X_4 = (2.0, 3.0)$	1.74	7.9	$\gg$	4.54
$X_5 = (1.5, 3.0)$	1.74	6.1	$\gg$	3.50
$X_6 = (1.5, 1.0)$	1.16	5.2	$\approx$	4.48

3.2.6. Species-Conserving Genetic Algorithms

Introdotta in [39], questa tecnica di niching si basa sul concetto di specie, ossia un sottoinsieme della popolazione in cui la distanza tra ogni coppia di individui è minore di un certo parametro σ_s (nota che non deve essere vero il contrario, ossia che se due individui distano meno di σ_s allora devono appartenere alla stessa specie).

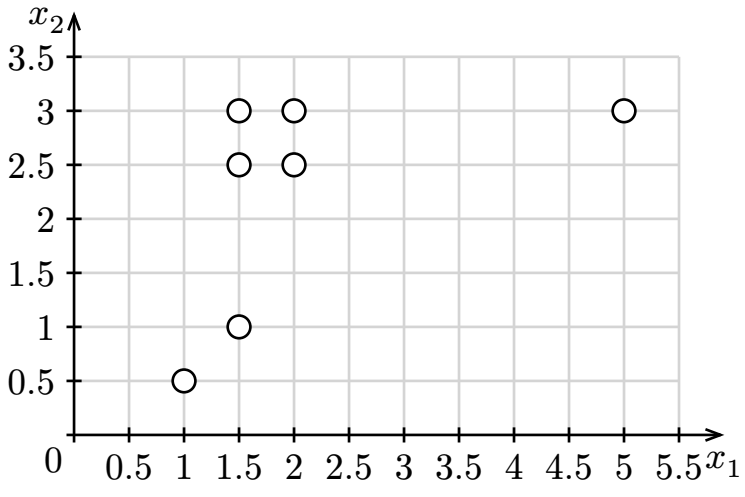
Tale tecnica punta ad identificare e a preservare il miglior individuo di ogni specie, chiamato *seed*, trasferendolo automaticamente alla generazione successiva, in particolare:

1. Ordina la popolazione in ordine decrescente di fitness e crea una lista, inizialmente vuota, di seed
2. Considera un individuo alla volta partendo dal primo, se questo dista più di $\frac{\sigma_s}{2}$ da ogni seed già esistente, allora aggiungilo alla lista dei seed
3. Una volta considerati tutti gli individui, copia i seed nella popolazione successiva. Infine, prosegui assegnando i rimanenti slot di popolazione attraverso i classici operatori di selezione, riproduzione, mutazione e rimpiazzamento

Riprendendo sempre l'esempio considerato nella trattazione del *clustering*, considerando $\sigma_s = 1.5$, si può osservare come i seed identificati dall'algoritmo siano effettivamente gli individui migliori della specie a cui appartengono:

- Situazione iniziale

Individuo	$f(x_i)$
$X_0 = (1.0, 0.5)$	15.6
$X_1 = (1.5, 2.5)$	14.5
$X_2 = (5.0, 3.0)$	11.3
$X_3 = (2.0, 2.5)$	10.9
$X_4 = (2.0, 3.0)$	7.9
$X_5 = (1.5, 3.0)$	6.1
$X_6 = (1.5, 1.0)$	5.2

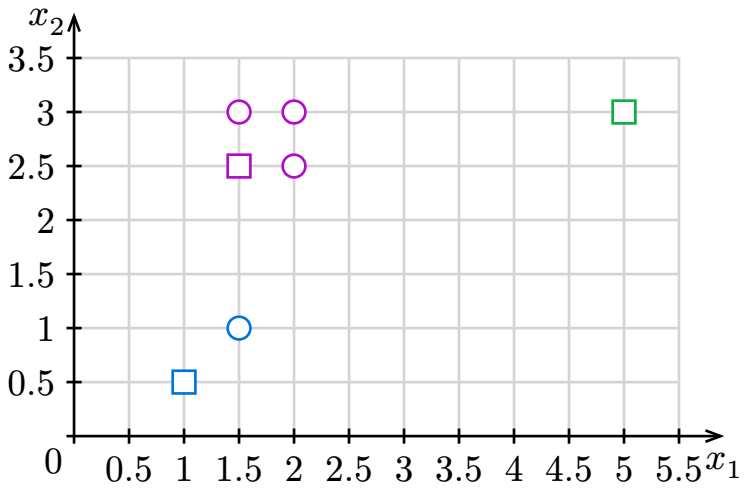


- Esecuzione dell'algoritmo
  - Considero X_0: è il primo individuo, dunque viene aggiunto automaticamente alla lista dei seed
  - Considero X_1: non c'è nessun seed che dista meno di $0.5 \cdot \sigma_s$ da X_1, dunque viene aggiunto alla lista dei seed
  - Considero X_2: non c'è nessun seed che dista meno di $0.5 \cdot \sigma_s$ da X_2, dunque viene aggiunto alla lista dei seed
  - Considero X_3: dista meno di $0.5 \cdot \sigma_s$ dal seed X_1, dunque appartiene alla sua stessa specie
  - Considero X_4: dista meno di $0.5 \cdot \sigma_s$ dal seed X_1, dunque appartiene alla sua stessa specie
  - Considero X_5: dista meno di $0.5 \cdot \sigma_s$ dal seed X_1, dunque appartiene alla sua stessa specie
  - Considero X_6: dista meno di $0.5 \cdot \sigma_s$ dal seed X_0, dunque appartiene alla sua stessa specie

Nota che non è necessario tenere traccia di quale individuo appartiene a quale specie, basta memorizzare solamente i seed.

- Situazione finale (i seed sono rappresentati con dei quadrati)

Individuo	$f(x_i)$
$X_0 = (1.0, 0.5)$	15.6
$X_1 = (1.5, 2.5)$	14.5
$X_2 = (5.0, 3.0)$	11.3
$X_3 = (2.0, 2.5)$	10.9
$X_4 = (2.0, 3.0)$	7.9
$X_5 = (1.5, 3.0)$	6.1
$X_6 = (1.5, 1.0)$	5.2



3.3. Stato dell'arte

3.3.1. Metriche di valutazione

Nel tempo sono state proposte diverse metriche di valutazione [30], [31], [40], ad oggi, le più utilizzate sono:

- **Success rate**: percentuale di esecuzioni in cui vengono trovati tutti gli ottimi
- **Tempo di computazione** o, equivalentemente, **numero medio di valutazioni della funzione di fitness** (dato che è sempre la componente computazionalmente più onerosa dell'algoritmo)
- **Success performance**: se la *success rate* è diversa da zero, allora un ulteriore metrica di valutazione può essere calcolata come il *numero medio di valutazioni della funzione di fitness* diviso la *success rate*
- **Peak Ratio**: numero medio di ottimi trovati
- **Maximum Peak Ratio**: $\mathrm{MPR} = \frac{\sum(f_i)}{\sum(F_i)}$, dove $\sum(f_i)$ è la somma della fitness degli individui considerati ottimi nella popolazione finale, mentre $\sum(F_i)$ è la somma della fitness dei veri ottimi del problema

3.3.2. Funzioni di benchmark

Le funzioni di benchmark per problemi multi-modali a singolo obbiettivo sono innumerevoli, di conseguenza, di seguito ne vengono illustrate solamente alcune (per un elenco esaustivo il lettore può fare riferimento all'appendice di [31]):

- Funzione con picchi di altezza decrescente, posti ad intervalli irregolari (Figura 4):

$$f(x) = \exp\left(-2\ln(2)\left(\frac{x-0.01}{0.8}\right)^2\right)\sin^6(5\pi(x^{0.75}-0.05))$$
$$x \in [0,1]$$

- "Six-hump camel back function" (Figura 5):

$$f(x,y) = -\frac{4}{3}x^6 + 8.4x^4 - 16x^2 - 16y^4 + 16y^2 - 4xy$$
$$x \in [-1.9, 1.9],\ \ y \in [-1.1, 1.1]$$

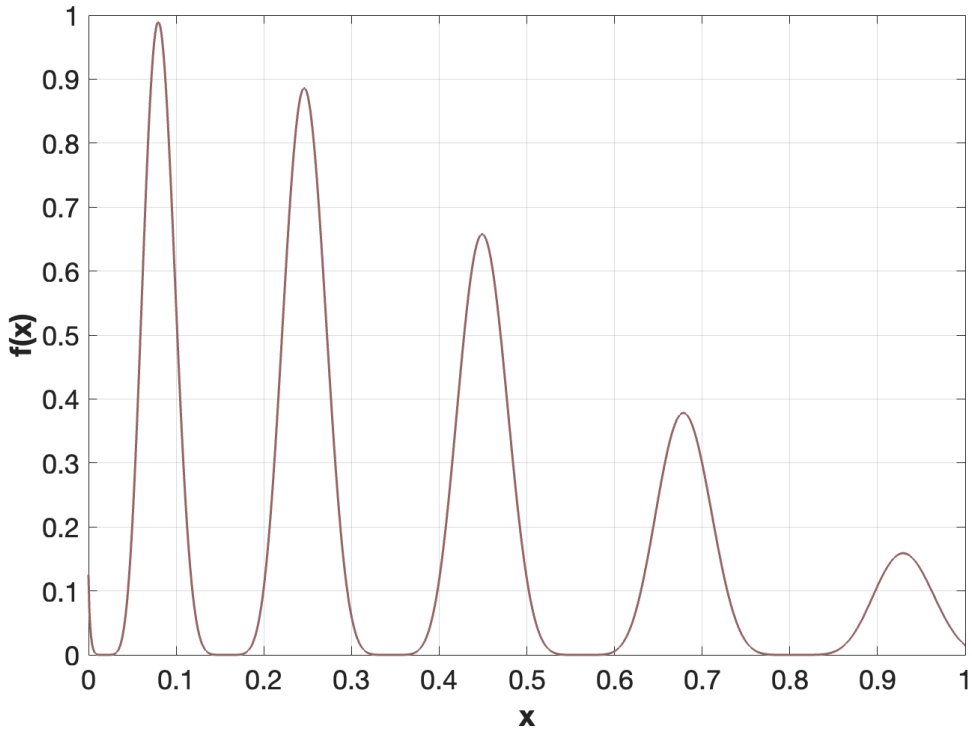


Figura 4: Funzione con picchi di altezza decrescente, posti ad intervalli irregolari

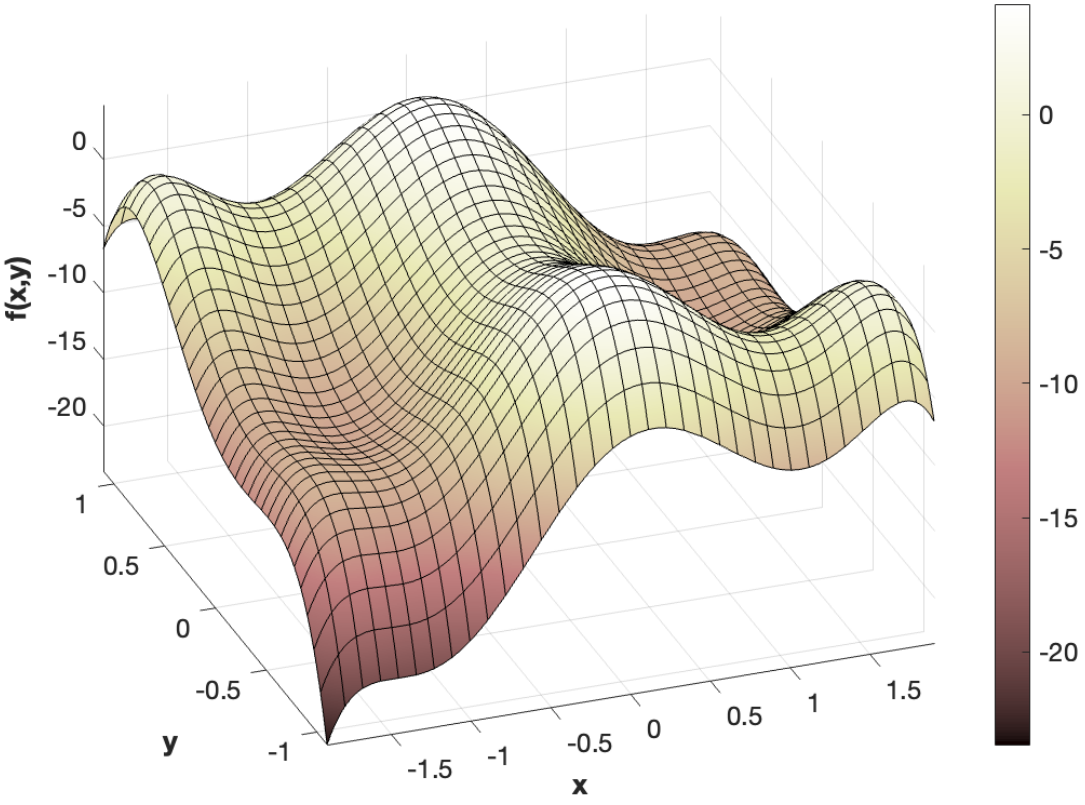


Figura 5: “Six-hump camel back function”

È anche possibile definire funzioni di benchmark per problemi di tipo binario, un esempio, tratto da [41], è il seguente, dove i cromosomi (stringhe di 30 bit) sono genericamente indicati con C:

$$f(C) = u(\text{ones}([C_{29}, C_{28}, C_{27}, C_{26}, C_{25}, C_{24}])) + ...$$
$$... + u(\text{ones}([C_5, C_4, C_3, C_2, C_1, C_0]))$$

Dove:

$$C_i = \text{Bit } i\text{-esimo della stringa } C$$

$$\text{ones}(C) = \text{Numero di bit con valore 1 in } C$$

$$u(x) = \begin{cases} 1 & \text{se } x = 0 \vee x = 6 \\ 0 & \text{se } x = 1 \vee x = 5 \\ 0.360384 & \text{se } x = 2 \vee x = 4 \\ 0.640576 & \text{se } x = 3 \end{cases}$$

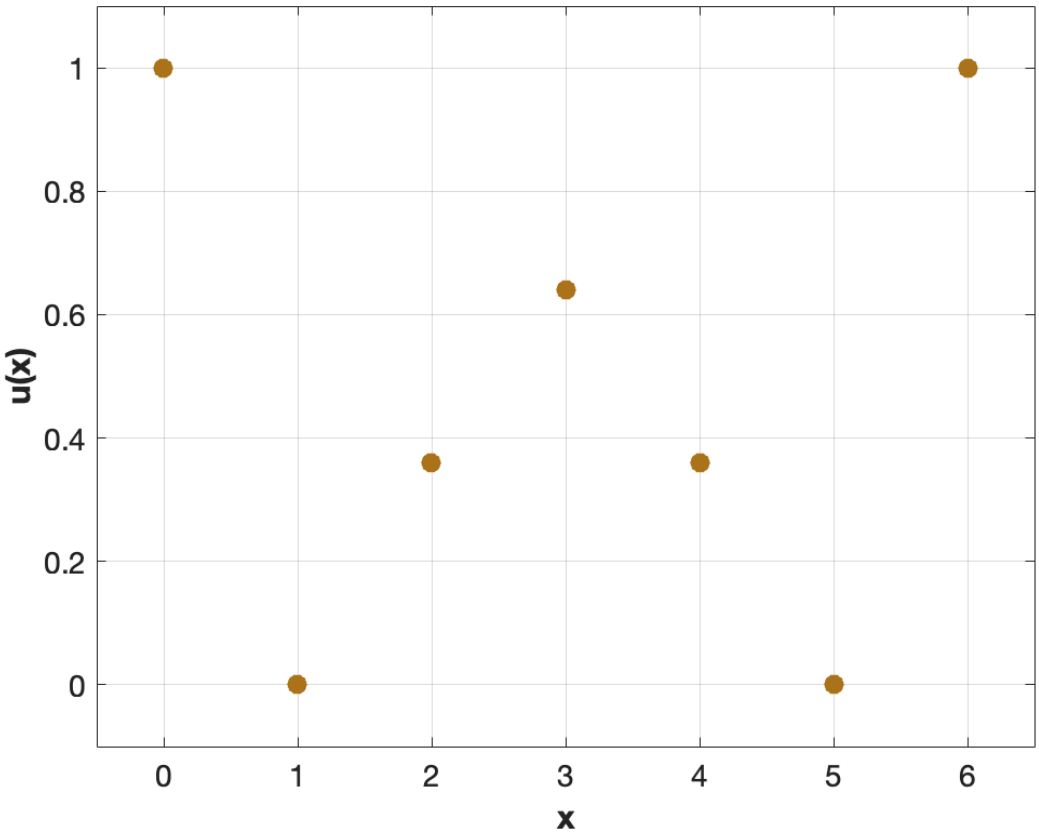


Figura 6: Funzione $u(x)$ definita sopra

3.3.3. Risultati sperimentali

In [30] è stato utilizzato un algoritmo genetico a codifica reale, con operatore di riproduzione *SBX* ed operatore di mutazione polinomiale, per studiare l'efficacia delle tecniche di niching elencate sopra; i risultati ottenuti sono i seguenti:

- Le tecniche di *Sharing* ed *SCGA* si sono rivelate consistentemente sia lente ($O(N^2)$) che imprecise, inoltre, richiedendo entrambe parametri aggiuntivi per poter essere eseguite, non sono sicuramente la prima scelta
- La procedura di *Clustering* è spesso imprecisa, e richiede anch'essa dei parametri aggiuntivi, tuttavia, finché i cluster non sono $O(N)$, dove N è il numero di individui nella popolazione, risulta computazionalmente migliore di *Sharing* ed *SCGA*
- La tecnica del *Crowding*, essendo $O(N)$, risulta essere sempre la tecnica più veloce, inoltre, come *RTS*, è facile da implementare e non richiede alcun parametro aggiuntivo. Anche la tecnica di *clearing* standard si rivela essere un buon compromesso, tuttavia, richiede il parametro σ_{clear}
- Infine, la tecnica del *modified clearing* si rivela essere con consistenza una delle più accurate, ma anche la più lenta, impiegando fino a due ordini di grandezza in più rispetto a quella del *crowding*

In generale, vale il "no free lunch theorem" [42], ossia, non esiste una singola tecnica di niching che performi sempre meglio delle altre in ogni problema. Di conseguenza, per risolvere un particolare problema multi-modale, vengono spesso provate diverse tecniche di niching, ognuna con diversi parametri.

Una possibile alternativa, proposta in [41], sono i cosiddetti algoritmi **ensemble**, che traggono vantaggio dall'utilizzare in parallelo diverse tecniche di niching, o comunque le stesse tecniche ma con parametri diversi.

Più precisamente, vengono eseguiti in parallelo più algoritmi genetici, ognuno con una propria popolazione, dunque indipendenti l'uno dall'altro. Al termine delle operazioni di selezione, riproduzione e mutazione, la prole di ogni algoritmo viene analizzata da tutti gli altri, ed accettata o rifiutata nella propria popolazione a seconda dei particolari meccanismi di rimpiazzamento.

È stato dimostrato in [41] che, al netto di un costo computazionale maggiore, questo approccio risulta migliore rispetto all'eseguire singolarmente le tecniche di niching considerate.

Infine, tutti quelli visti finora sono esempi di **parallel niching**, in quanto puntano a formare e a mantenere parallelamente diverse niches, un ulteriore approccio, noto come **sequential niching**, consiste nel sviluppare le niches singolarmente, senza necessità di doverne mantenere più di una contemporaneamente.

Un esempio di *sequential niching*, introdotto in [43], consiste nel degradare artificialmente la funzione di fitness nei punti in cui sono state precedentemente trovate delle *buone* soluzioni, in modo tale che la popolazione si sposti da quelle zone, esplorando il più possibile lo spazio di decisione.

In particolare, la funzione di fitness $f(x)$ alla generazione $t+1$ può essere espressa dalla seguente relazione di ricorrenza:

$$f(x)_{t+1} = f(x)_t \cdot D(x, \text{best}_t)$$

Dove:

- best_t è l'individuo migliore alla generazione t
- $D(x, x^*)$ è la *derating function*, un esempio può essere:

$$D(x, x^*) = \begin{cases} \left(\frac{\|x - x^*\|}{r}\right)^{\alpha} & \text{se } \|x - x^*\| < r \\ 1 & \text{altrimenti} \end{cases}$$

  - r è il raggio d'effetto
  - α determina la concavità ($\alpha > 1$) o la convessità ($\alpha < 1$) della funzione

4. Problemi multi-obiettivo

4.1. Introduzione

Fino a questo momento, sono stati considerati solamente problemi di ottimizzazione multi-modali a singolo obiettivo, tuttavia, buona parte dei problemi del mondo reale sono per loro natura multi-obiettivo [44], e spesso anche multi-modali [45] (ad esempio, considerando un investimento finanziario, si desidera non solo massimizzare il profitto, ma anche minimizzare il rischio). Si procede dunque con il definire questa tipologia di problemi e con l'analizzarne brevemente lo stato dell'arte.

4.2. Definizioni di base

Un generico problema di ottimizzazione multi-obiettivo, indicato con l'acronimo **MOP** (Multi-Objective Problem), può essere espresso come:

$$\min_x \{F(x)\} \mid F(x) = \{f_1(x), ..., f_m(x)\},\ x = (x_1, ..., x_n) \in X$$

Le funzioni obiettivo $f_1, ..., f_m$ sono spesso contrastanti e dunque impossibili da minimizzare contemporaneamente, in genere non esisterà dunque un'unica soluzione ottima, ma un insieme di soluzioni ottime che meglio bilanciano le varie funzioni obiettivo.

Più formalmente, si dice che una soluzione x_a domina, nel senso di Pareto, una soluzione x_b se e solo se:

$$\forall i = 1, ..., m, f_i(x_a) \leq f_i(x_b) \land \exists j = 1, ..., m \mid f_j(x_a) < f_j(x_b)$$

Se una soluzione non è dominata, nel senso di Pareto, da nessun'altra soluzione, ossia se, partendo da essa, nessuna funzione obiettivo può essere minimizzata ulteriormente senza incrementarne almeno un'altra, allora tale soluzione è definita **Pareto ottima**.

L'insieme di tutte le soluzioni Pareto ottime (che può anche essere infinito) è definito **insieme di Pareto**, mentre l'immagine di tale insieme è definita **fronte di Pareto**, come illustrato in Figura 7.

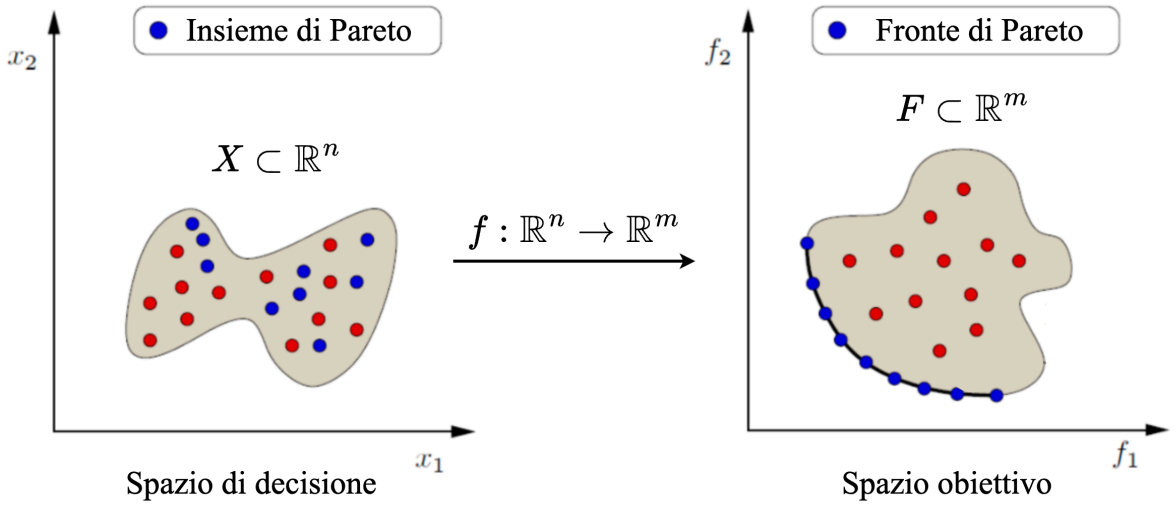


Figura 7: Illustrazione dell'insieme di Pareto e del fronte di Pareto [46]

4.3. Algoritmi genetici per problemi multi-obiettivo

L'algoritmo genetico di base illustrato nella Sezione 2.2 rimane pressoché invariato anche per problemi multi-obiettivo, l'unico cambiamento risiede nel modo in cui viene valutata ed assegnata la funzione di fitness. La popolazione viene infatti divisa in fronti: il primo fronte contiene solo soluzioni non dominate, il secondo fronte contiene soluzioni dominate solamente dal primo, e così via. Al termine dell'algoritmo il primo fronte verrà considerato come approssimazione del vero fronte di Pareto del problema.

Algoritmi evolutivi multi-obiettivo che operano come descritto sopra vengono definiti **Pareto-based**, esistono tuttavia degli approcci alternativi, uno di questi è, ad esempio, quello adottato dai cosiddetti algoritmi **decompose-based**, che dividono il problema multi-obiettivo in più sottoproblemi singolo-obiettivo da risolvere parallelamente [44].

4.4. Concetto di multi-modalità per problemi multi-obiettivo

Come accennato precedentemente, anche i problemi multi-obiettivo possono essere multi-modali, tuttavia, la loro definizione è ancora controversa [2]. Una possibile definizione, proposta in [45], è la seguente:

1. Sia x_{Po} una soluzione Pareto ottima di un MOP, se esiste una "soluzione distante" x tale che $\|F(x_{\text{Po}}) - F(x)\| \leq \varepsilon, \varepsilon \geq 0$, allora il particolare MOP è considerato multi-modale, e dunque definito **MMOP** (Multi-modal Multi-Objective Problem)
2. Due soluzioni x_a, x_b sono dette "distanti" se $\|x_a - x_b\| \geq \delta$, dove δ è un parametro libero (positivo)

In particolare:
- se $\varepsilon = 0$ significa che esistono più insiemi di Pareto globalmente ottimi che corrispondono allo stesso fronte di Pareto
- se $\varepsilon > 0$ significa che esistono insiemi di Pareto e Fronti di Pareto localmente ottimi, in questo caso si parla di **MMOPLs** (Multimodal Multi-Objective Problems with Local Pareto fronts)

4.5. Metriche di valutazione

I problemi di ottimizzazione multi-obiettivo richiedono alcune accortezze in più rispetto a quelli singolo-obiettivo, è infatti necessario bilanciare due aspetti [47]: la convergenza al fronte di Pareto e la distribuzione delle soluzioni su quest'ultimo, desiderata il più possibile uniforme [9].

Una metrica di valutazione che misura la qualità di tali aspetti è l'**Inverted Generational Distance** [50]:

$$\text{IGD}(X) = \frac{1}{|X^*|} \sum_{y \in X^*} \min_{x \in X} \{d(f(x), f(y))\}$$

Nei problemi multi-obiettivo considerati anche multi-modali, l'obbiettivo principale non è più quello di approssimare bene il fronte di Pareto, ma è quello di ottenere più soluzioni Pareto ottime possibile [51].

Di conseguenza, la metrica di valutazione preferita è la diversità delle soluzioni nello spazio di decisione (**Inverted Generational Distance in the Decision Space**) [52]:

$$\text{IGDX}(X) = \frac{1}{|X^*|} \sum_{y \in X^*} \min_{x \in X} \{d(x, y)\}$$

[9] L'algoritmo NSGA-II [48], ad esempio, utilizza la **crowding distance**, ossia la distanza media delle due soluzioni più vicine, come secondo criterio di fitness dopo la qualità della convergenza. Sono tuttavia possibili anche tecniche alternative [49]

In particolare:

- X è l'insieme di soluzioni fornito in output dal particolare algoritmo scelto
- X^* è un insieme di soluzioni Pareto ottime campionate in modo uniforme dal vero insieme di Pareto del problema scelto come benchmark

I due indicatori elencati precedentemente sono ritenuti complementari, in quanto uno guarda lo spazio di decisione e l'altro lo spazio obiettivo, e dunque vengono spesso utilizzati entrambi.

Un ulteriore indicatore, il quale riflette la similarità tra il vero insieme di Pareto del problema scelto come benchmark e l'insieme di Pareto ottenuto dal particolare algoritmo scelto, è il **Pareto Sets Proximity** (PSP) [53]:

$$\mathrm{PSP}(X) = \frac{\mathrm{CR}(X)}{\mathrm{IGDX}(X)}$$

$$\mathrm{CR}(X) = \left(\prod_{i=1}^{n} \sigma_i \right)^{\frac{1}{2n}}$$

$$\sigma_i = \begin{cases} 1 \text{ se } x_i^{*,\max} = x_i^{*,\min} \\ 0 \text{ se } x_i^{*,\max} \leq x_i^{\min} \quad \text{oppure} \quad x_i^{\max} \leq x_i^{*,\min} \\ \left(\frac{\min(x_i^{*,\max}, x_i^{\max}) - \max\left(x_i^{*,\min}, x_i^{\min}\right)}{x_i^{*,\max} - x_i^{*,\min}} \right)^2 \text{ altrimenti} \end{cases}$$

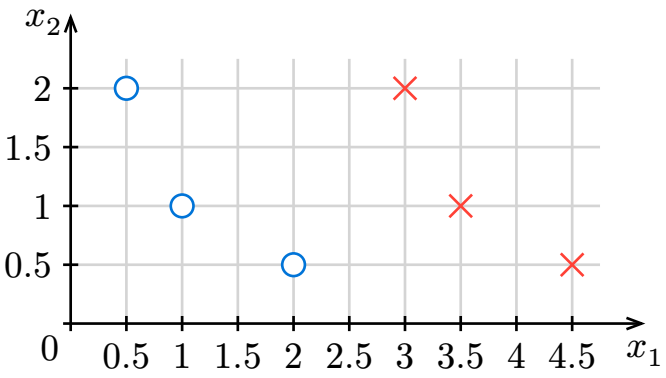
Dove:

- CR sta per Cover Rate
- n è la dimensionalità del problema
- $x_i^{*,\max/\min}$ è il massimo/minimo valore della i-esima componente tra tutte le soluzioni appartenenti al vero insieme di Pareto X^*
- $x_i^{\max/\min}$ è il massimo/minimo valore della i-esima componente tra tutte le soluzioni appartenenti all'insieme di Pareto ottenuto in output X

Più piccoli sono IGD ed IGDX meglio è, e più grande è PSP meglio è; tuttavia, siccome quest'ultimo può essere infinitamente grande, spesso si preferisce utilizzarne il reciproco [54]:

$$\mathrm{RPSP} = \frac{1}{\mathrm{PSP}}$$

Per fare degli esempi, nel caso semplice in cui $n = 2$:



- Insieme di Pareto reale [10] : X^*
- Insieme di Pareto calcolato: X

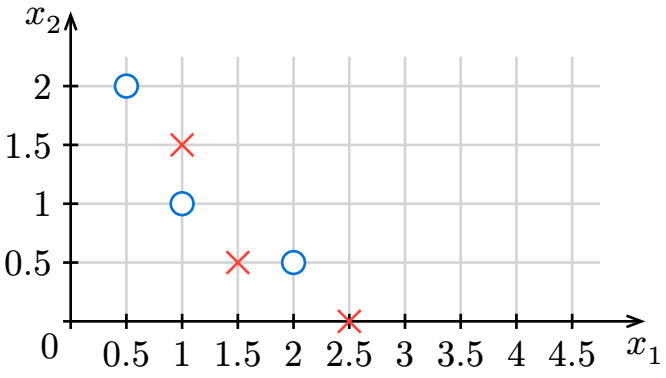
$\sigma_1 = 0$

$\sigma_2 = 1$

$\mathrm{CR}(X) = 0$

$\mathrm{IGDX}(X) \approx 2.1$

$\mathrm{RPSP}(X) = \infty$



- Insieme di Pareto reale [10]: X^*
- Insieme di Pareto calcolato: X

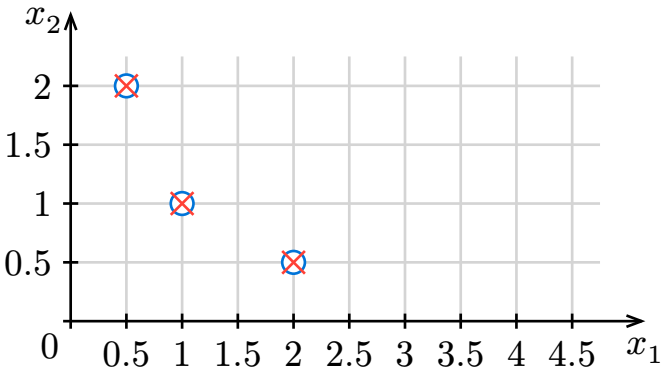
$\sigma_1 = \frac{2}{3}$

$\sigma_2 = \frac{2}{3}$

$\mathrm{CR}(X) \approx 0.816$

$\mathrm{IGDX}(X) \approx 0.638$

$\mathrm{RPSP}(X) \approx 0.782$



- Insieme di Pareto reale [10]: X^*
- Insieme di Pareto calcolato: X

$\sigma_1 = 1$

$\sigma_2 = 1$

$\mathrm{CR}(X) = 1$

$\mathrm{IGDX}(X) = 0$

$\mathrm{RPSP}(X) = 0$

In conclusione, tutte le metriche elencate in precedenza necessitano di conoscere sia il vero insieme di Pareto che il vero fronte di Pareto del problema usato come benchmark, cosa difficile per problemi ingegneristici del mondo reale. In aggiunta, non è ancora stato proposto un singolo indicatore che, senza richiedere parametri aggiuntivi, misuri allo stesso tempo la diversità delle soluzioni e la convergenza al fronte di Pareto [45].

[10] Campionato uniformemente da un ipotetico problema

4.6. Stato dell'arte

I seguenti paragrafi sono basati sul lavoro di ricerca svolto da *W. Li et al.* [45], in cui vengono studiati, in termini di IGD, IGDX, RPSP e costo computazionale, 15 tra i vari algoritmi evolutivi proposti negli ultimi anni per la risoluzione di problemi multi-obiettivo multi-modali.

4.6.1. Algoritmi

Algoritmo	Framework	Locale	Data	Rif.
Omni-optimizer	GA		2008/03	[55]
DN-NSGAII	GA		2016/06	[56]
MO_Ring_PSO_SCD	PSO		2017/09	[57]
MO_PSO_MM	PSO		2018/06	[58]
DNEA	GA		2018/08	[59]
Tri-MOEA&TAR	Decompose-based		2018/11	[60]
DNEA-L	GA	Si	2019/06	[61]
CPDEA	One-by-one		2019/08	[62]
SS-MOPSO	PSO		2019/10	[63]
MMODE_CSCD	DE		2020/10	[64]
MP-MMEA	GA		2020/12	[65]
MMOEA/DC	GA	Si	2021/01	[66]
MMODE_ICD	DE		2021/04	[67]
MMEA-WI	Indicator-based		2021/12	[68]
HREA	GA	Si	2022/03	[69]

Si può subito osservare che la strategia di base più popolare è quella degli algoritmi genetici (GA), seguita da Particle Swarm Optimization (PSO), e da altri approcci meno comuni.

In particolare, tre algoritmi si concentrano sul risolvere MMOPLs, ossia sul trovare insiemi di Pareto locali, preferendo la diversità delle soluzioni alla qualità della convergenza.

4.6.2. Benchmark

Le suite di test utilizzate in [45] sono quattro, ognuna delle quali studia il comportamento dei vari algoritmi su un insieme di problemi con particolari caratteristiche:

1. **IEEE CEC 2020** : È un sottoinsieme della suite di test proposta nell'IEEE CEC 2020 (Congress on Evolutionary Computation), l'evento principale nel campo della computazione evolutiva. I problemi di cui è composta sono caratterizzati da insiemi di Pareto molto complessi, e per questo è considerata come rappresentativa dei MMOPs del mondo reale.

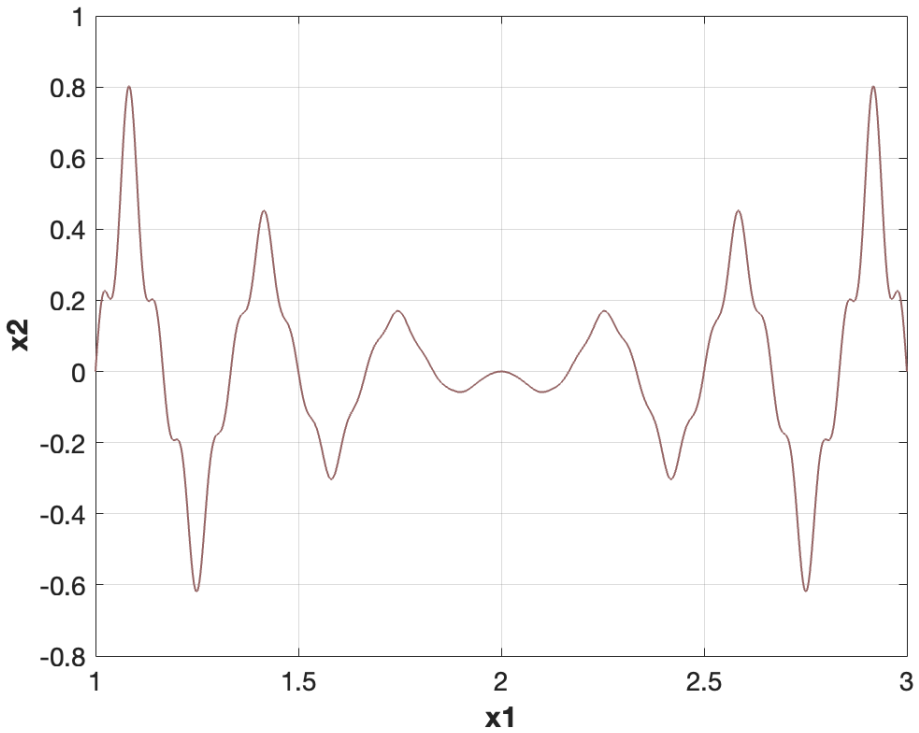
   Ad esempio, il problema denominato "MMF7" [70]:

   $$\begin{cases} f_1 = |x_1 - 2| \\ f_2 = 1 - \sqrt{|x_1 - 2|} + (x_2 - g(x_1))^2 \end{cases}$$
   $$x_1 \in [1, 3],\ x_2 \in [-1, 1]$$

   Dove:

   $$g(x_1) = \left(0.3|x_1 - 2|^2 \cdot \cos(24\pi|x_1 - 2| + 4\pi) + 0.6|x_1 - 2|\right) \cdot \sin(6\pi|x_1 - 2| + \pi)$$

   Ha come insieme di Pareto (in questo caso uno solo) $x_2 = g(x_1)$:



2. **IDMP**: Contiene problemi costruiti appositamente in modo da avere diversi livelli di difficoltà nel trovare i vari insiemi di Pareto; gli algoritmi con meccanismi di niching poco avanzati sono dunque portati a convergere verso gli insiemi di Pareto considerati facili da trovare

3. **MMOPLs**: Contiene problemi tratti da parte della suite IEEE CEC 2020 e dall'estensione della suite IDMP (IDMP_e), i quali testano in modo specifico l'abilità degli algoritmi nel trovare insiemi e fronti di Pareto locali

4. **Multi-Polygon**: È composta da problemi altamente flessibili, con un ampio numero di funzioni obiettivo e di variabili decisionali, studiati appositamente per testare la scalabilità dei vari algoritmi.

   Per fare un esempio, tratto da [71], si considerino quattro esagoni regolari di raggio r identici, posti a distanza l l'uno dall'altro [11], come illustrato in Figura 8, allora, un possibile problema di ottimizzazione multi-obiettivo e multimodale è il seguente:

$$\min_x \{F(x)\} \mid F(x) = \begin{cases} f_1(x) = \min(d(x, A_1), d(x, B_1), d(x, C_1), d(x, D_1)) \\ \cdots \\ f_6(x) = \min(d(x, A_6), d(x, B_6), d(x, C_6), d(x, D_6)) \end{cases}$$

   In questo esempio, ogni punto interno ad uno dei quattro esagoni è Pareto ottimo e, di conseguenza, sono presenti quattro insiemi di Pareto. È immediato vedere come in problemi di questo tipo sia facile aumentare il numero di funzioni obiettivo o di variabili decisionali.

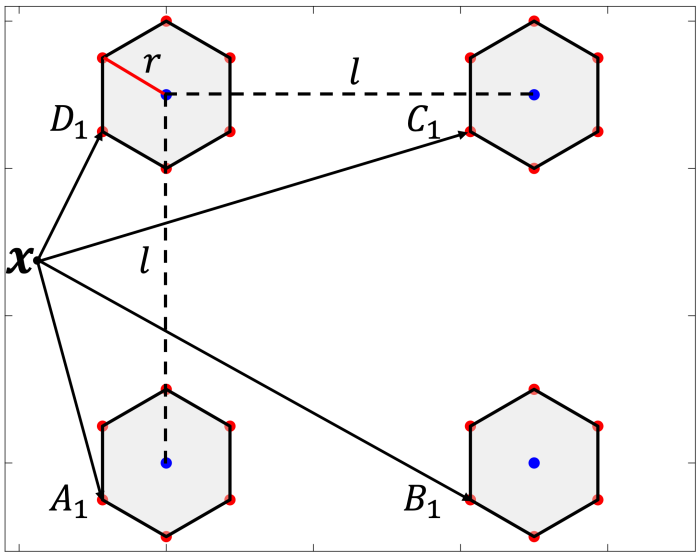


Figura 8: Illustrazione dell'esempio di *multi-polygon problem* tratto da [71]

[11] l deve essere maggiore o uguale di $4r$ [72]

4.6.3. Metodologia di test

Per quanto riguarda la metodologia di test, la strategia adottata in [45] è la seguente:

- Ogni problema viene eseguito da ogni algoritmo 30 volte (numero scelto per minimizzare l'aleatorietà dei risultati, mantenendo un tempo di esecuzione ragionevole)
- Per ognuna delle 30 esecuzioni di ogni problema, viene stilata una classifica in termini di IGD, IGDX e RPSP dei vari algoritmi, indicando con r_i^j il rank dell'algoritmo i nel problema j per la specifica metrica selezionata
- Infine, scelta una metrica, il rank finale dell'algoritmo i su un insieme di J problemi è calcolato come [12]

$$R_i = \frac{\sum_{j=1}^{J} r_i^j}{J}$$

4.6.4. Risultati sperimentali

In generale, come per il caso singolo-obiettivo, vale il "no free lunch theorem" [42], ossia non c'è un singolo algoritmo che performi sempre meglio degli altri in ogni problema. Ci sono però algoritmi che performano estremamente bene su particolari problemi:

- Nella suite IEEE CEC 2020 CPDEA vince nettamente sugli altri, mentre MMODE_CSCD e MO_PSO_MM sono relativamente stabili e competitivi
- In IDMP, come accennato precedentemente, viene testata l'abilità nel trovare insiemi di Pareto di difficoltà crescente, di conseguenza, gli algoritmi locali (HREA, MMOEA/DC, DNEA-L), i quali si concentrano sull'esplorare bene lo spazio di decisione, performano meglio in questo tipo di problemi, con CPDEA e MMEWA-WI che comunque tengono il passo
- Infine, in MMOPLs, i tre algoritmi locali, essendo studiati apposta per tale tipologia di problemi, dominano in modo assoluto sugli altri

Si può osservare in Figura 9 che, mediamente, HREA e MMOEA/DC sono competitivi in tutti i test, mentre algoritmi più vecchi come Omni-Optimizer e DN-

[12] Questo processo è noto come test di Friedman [73]

NSGAII, o algoritmi specifici come Tri-MOEA&TAR [13] e MP-MMEA [14], performano costantemente peggio.

Infine, per quanto riguarda il costo computazionale, la maggior parte degli algoritmi ricade approssimativamente nello stesso ordine di grandezza, ad esclusione di SS-MOPSO, che è notevolmente [15] più oneroso, e di DNEA, Tri-MOEA&TAR e MP-MMEA, che sono nettamente [15] più efficienti degli altri. Curiosamente, il costo computazionale non varia con il numero degli obbiettivi, ed in generale sale lievemente con il numero delle variabili decisionali (tranne per alcuni algoritmi, nei quali, contro-intuitivamente, diminuisce).

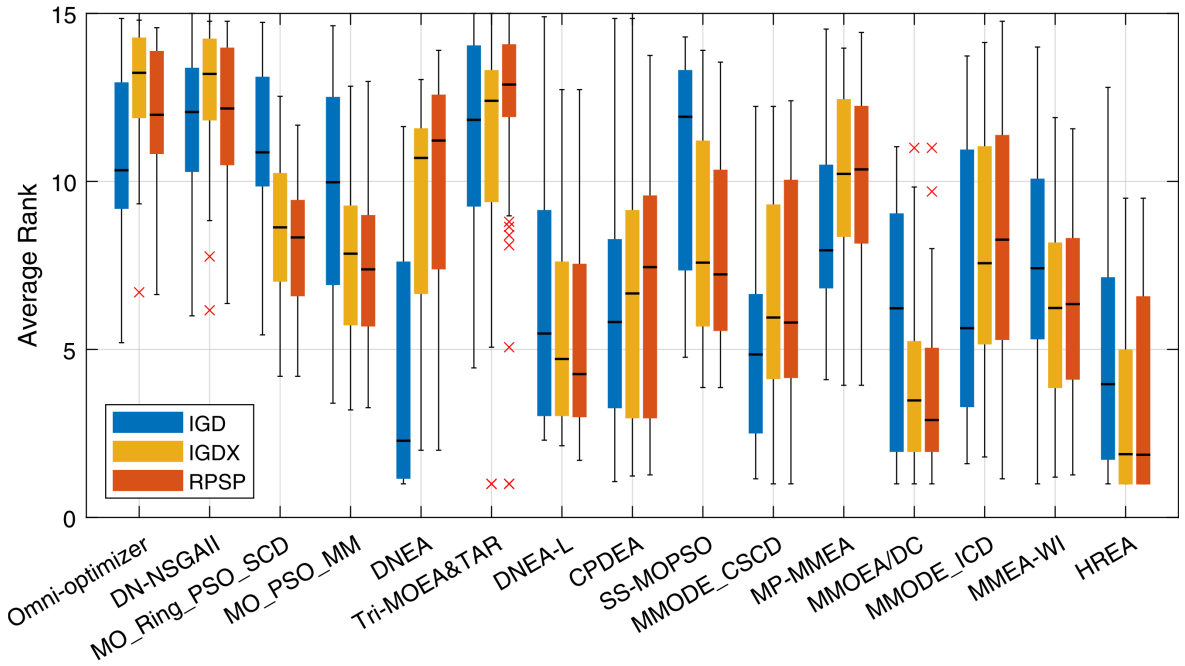


Figura 9: Rank medio di ciascun algoritmo considerando tutti i test [45]

[13] Sfrutta le proprietà delle variabili decisionali e le relazioni tra di esse per inizializzare il processo di minimizzazione

[14] Progettato per problemi con molte variabili decisionali e con insiemi di Pareto sparsi

[15] Circa 1-2 ordini di grandezza

5. Conclusioni

In questo lavoro di tesi è stato studiato lo stato dell'arte nel campo dell'ottimizzazione stocastica multi-modale.

In particolare, in un primo momento sono state analizzate le tecniche di ottimizzazione stocastica per i **problemi a singolo obiettivo**, introducendo dunque le generalità sugli algoritmi stocastici ed alcune metaeuristiche di esempio.

Tra le varie metaeuristiche proposte, data la loro importanza storica e rilevanza attuale, è stato scelto di approfondire gli **algoritmi genetici**. Sono state quindi introdotte le possibili codifiche dei cromosomi, gli operatori di selezione, riproduzione, mutazione e rimpiazzamento, le condizioni di terminazione e le più recenti varianti e miglioramenti proposti nella letteratura.

Dopo di che, sempre per quanto riguarda i problemi a singolo obiettivo, è stato introdotto il **concetto di multi-modalità**. È stato dunque spiegato il significato del termine **niching** e sono state studiate le tecniche di niching allo stato dell'arte per gli algoritmi genetici, arricchendo l'esposizione con numerosi esempi pratici dove ritenuto opportuno.

Per completare la trattazione riguardante i problemi multi-modali a singolo obiettivo, sono state infine elencate le possibili metriche di valutazione e funzioni di benchmark, e sono stati analizzati i principali **risultati sperimentali** ottenuti nella letteratura, i quali hanno evidenziato come la tecnica di niching *modified clearing* sia consistentemente una delle migliori, ma sia anche computazionalmente onerosa, mentre le tecniche di *clearing* ed *RTS* siano degli ottimi compromessi.

Per la loro rilevanza nel mondo reale, e per il loro crescente interesse nella comunità scientifica, sono stati studiati anche i **problemi multi-obiettivo**. Sono state dunque fornite al lettore delle definizioni di base, ed è stato illustrato come un algoritmo genetico standard può essere modificato per adattarsi a problemi di tale tipo.

Infine, è stato spiegato il concetto di **multi-modalità per problemi multi-obiettivo**, e, come fatto precedentemente per i problemi singolo-obiettivo, sono state illustrate le principali metriche di valutazione, funzioni di benchmark e risultati sperimentali ottenuti nella letteratura.

In generale, il campo dell'ottimizzazione multi-modale e multi-obiettivo è relativamente nuovo rispetto a quello multi-modale singolo-obiettivo, e dunque presenta ancora importanti possibilità di miglioramento [45], in particolare:

- Gli algoritmi che si concentrano sul trovare insiemi di Pareto locali sembrano performano bene e potrebbero essere un importante area di ricerca futura
- Non è stato ancora formulato un algoritmo in grado di determinare se un dato problema multi-obiettivo sia multi-modale o meno
- Tutti i problemi proposti finora nelle maggiori suite di test sono continui, tuttavia, dato che i problemi del mondo reale sono spesso discreti o misti, sono necessarie suite di test che considerino anche questi aspetti
- Tutti gli algoritmi proposti nella letteratura hanno serie difficoltà su problemi con un grande numero di variabili decisionali
- Manca un singolo indicatore, privo di parametri aggiuntivi, che incorpori efficacemente le misure di IGD ed IGDX, inoltre, sempre in questo ambito, manca un indicatore che non richieda la conoscenza dei veri insiemi e fronti di Pareto del particolare problema usato come benchmark

Bibliografia

[1] «Multimodal Optimization», in *Introduction to Evolutionary Algorithms*, London: Springer London, 2010, pp. 165–191. doi: 10.1007/978-1-84996-129-5_5.

[2] R. Tanabe e H. Ishibuchi, «A Review of Evolutionary Multimodal Multiobjective Optimization», *IEEE Transactions on Evolutionary Computation*, vol. 24, fasc. 1, pp. 193–200, feb. 2020, doi: 10.1109/TEVC.2019.2909744.

[3] X. Yao, W. Li, X. Pan, e R. Wang, «Multimodal multi-objective evolutionary algorithm for multiple path planning», *Computers & Industrial Engineering*, vol. 169, p. 108145–108146, 2022, doi: 10.1016/j.cie.2022.108145.

[4] C. Blum e A. Roli, «Metaheuristics in combinatorial optimization: Overview and conceptual comparison», *ACM Comput. Surv.*, vol. 35, fasc. 3, pp. 268–308, set. 2003, doi: 10.1145/937503.937505.

[5] W. Jakob, «Applying evolutionary algorithms successfully: a guide gained from real-world applications», *arXiv preprint arXiv:2107.11300*, 2021, doi: 10.48550/arXiv.2107.11300.

[6] E. Galván-López, J. McDermott, M. O'Neill, e A. Brabazon, «Towards an understanding of locality in genetic programming», in *Proceedings of the 12th Annual Conference on Genetic and Evolutionary Computation*, in GECCO '10. Portland, Oregon, USA: Association for Computing Machinery, 2010, pp. 901–908. doi: 10.1145/1830483.1830646.

[7] S. Katoch, S. S. Chauhan, e V. Kumar, «A review on genetic algorithm: past, present, and future», *Multimedia Tools and Applications*, vol. 80, fasc. 5, pp. 8091–8126, feb. 2021, doi: 10.1007/s11042-020-10139-6.

[8] C. Neely, «Using Genetic Algorithms to Find Technical Trading Rules: A Comment on Risk Adjustment», *SSRN Electronic Journal*, 1999, doi: 10.2139/ssrn.189488.

[9] M. D. McKay, R. J. Beckman, e W. J. Conover, «A Comparison of Three Methods for Selecting Values of Input Variables in the Analysis of Output from a Computer Code», *Technometrics*, vol. 21, fasc. 2, pp. 239–245, 1979, Consultato: 20 agosto 2024. [Online]. Disponibile su: http://www.jstor.org/stable/1268522

[10] K. Jebari, «Selection Methods for Genetic Algorithms», *International Journal of Emerging Sciences*, vol. 3, pp. 333–344, 2013.

[11] D. E. Goldberg e K. Deb, «A Comparative Analysis of Selection Schemes Used in Genetic Algorithms», vol. 1. in Foundations of Genetic Algorithms, vol. 1. Elsevier, pp. 69–93, 1991. doi: 10.1016/B978-0-08-050684-5.50008-2.

[12] K. Sastry, D. E. Goldberg, e G. Kendall, «Genetic Algorithms», in *Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques*, E. K. Burke e G. Kendall, A c. di, Boston, MA: Springer US, 2014, pp. 93–117. doi: 10.1007/978-1-4614-6940-7_4.

[13] D. Whitley, «The GENITOR algorithm and selection pressure: why rank-based allocation of reproductive trials is best», in *Proceedings of the Third International Conference on Genetic Algorithms*, George Mason University, USA: Morgan Kaufmann Publishers Inc., 1989, pp. 116–121.

[14] K. Deb, R. B. Agrawal, e others, «Simulated binary crossover for continuous search space», *Complex systems*, vol. 9, fasc. 2, pp. 115–148, 1995.

[15] Z. Michalewicz, *Genetic algorithms + data structures = evolution programs (2nd, extended ed.)*. Berlin, Heidelberg: Springer-Verlag, 1994.

[16] K. Deb e D. Deb, «Analysing mutation schemes for real-parameter genetic algorithms», *Int. J. Artif. Intell. Soft Comput.*, vol. 4, fasc. 1, pp. 1–28, feb. 2014, doi: 10.1504/IJAISC.2014.059280.

[17] S. M. Lim, A. B. M. Sultan, M. N. Sulaiman, A. Mustapha, e K. Y. Leong, «Crossover and mutation operators of genetic algorithms», *International journal of machine learning and computing*, vol. 7, fasc. 1, pp. 9–12, 2017.

[18] G. Wu, R. Mallipeddi, e P. N. Suganthan, «Ensemble strategies for population-based optimization algorithms – A survey», *Swarm and Evolutionary Computation*, vol. 44, pp. 695–711, 2019, doi: 10.1016/j.swevo.2018.08.015.

[19] K. Deb e S. Agrawal, «A Niched-Penalty Approach for Constraint Handling in Genetic Algorithms», in *Artificial Neural Nets and Genetic Algorithms*, Vienna: Springer Vienna, 1999, pp. 235–243.

[20] E. Cantú-Paz, «Efficient and Accurate Parallel Genetic Algorithms», in *Genetic Algorithms and Evolutionary Computation*, 2000. [Online]. Disponibile su: https://api.semanticscholar.org/CorpusID:42893265

[21] P. Moscato e others, «On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms», *Caltech concurrent computation program, C3P Report*, vol. 826, fasc. 1989, p. 37–38, 1989.

[22] E. K. Burke, J. P. Newall, e R. F. Weare, «Initialization strategies and diversity in evolutionary timetabling», *Evol. Comput.*, vol. 6, fasc. 1, pp. 81–103, mar. 1998, doi: 10.1162/evco.1998.6.1.81.

[23] E. Burke, D. Elliman, e R. Weare, «Specialised recombinative operators for timetabling problems», in *Evolutionary Computing*, T. C. Fogarty, A c. di, Berlin, Heidelberg: Springer Berlin Heidelberg, 1995, pp. 75–85.

[24] M. Srinivas e L. Patnaik, «Adaptive probabilities of crossover and mutation in genetic algorithms», *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 24, fasc. 4, pp. 656–667, 1994, doi: 10.1109/21.286385.

[25] P. Aspar, P. Kerschke, V. Steinhoff, H. Trautmann, e C. Grimme, «Multi3: Optimizing Multimodal Single-Objective Continuous Problems in the Multi-objective Space by Means of Multiobjectivization», in *Evolutionary Multi-Criterion Optimization: 11th International Conference, EMO 2021, Shenzhen, China, March 28–31, 2021, Proceedings*, Shenzhen, China: Springer-Verlag, 2021, pp. 311–322. doi: 10.1007/978-3-030-72062-9_25.

[26] G. Hornby, A. Globus, D. Linden, e J. Lohn, «Automated Antenna Design with Evolutionary Algorithms», *Collection of Technical Papers - Space 2006 Conference*, vol. 1, 2006, doi: 10.2514/6.2006-7242.

[27] Y. Li, Y. Chen, J. Zhong, e Z. Huang, «Niching particle swarm optimization with equilibrium factor for multi-modal optimization», *Information Sciences*, vol. 494, pp. 233–246, 2019, doi: 10.1016/j.ins.2019.01.084.

[28] S. W. Mahfoud, «Niching methods for genetic algorithms», University of Illinois at Urbana-Champaign, USA, 1995. [Online]. Disponibile su: http://www.leg.ufpr.br/~leonardo/artigos/tese_mahfoud.pdf

[29] D. E. Goldberg e J. Richardson, «Genetic algorithms with sharing for multimodal function optimization», in *Proceedings of the Second International Conference on Genetic Algorithms on Genetic Algorithms and Their Application*, Cambridge, Massachusetts, USA: L. Erlbaum Associates Inc., 1987, pp. 41–49.

[30] G. Singh e K. Deb, «Comparison of multi-modal optimization algorithms based on evolutionary algorithms», in *Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation*, in GECCO '06. Seattle, Washington, USA: Association for Computing Machinery, 2006, pp. 1305–1312. doi: 10.1145/1143997.1144200.

[31] S. Das, S. Maity, B.-Y. Qu, e P. Suganthan, «Real-parameter evolutionary multimodal optimization — A survey of the state-of-the-art», *Swarm and Evolutionary Computation*, vol. 1, fasc. 2, pp. 71–88, 2011, doi: 10.1016/j.swevo.2011.05.005.

[32] K.-C. Wong, «Evolutionary multimodal optimization: A short survey», *arXiv preprint arXiv:1508.00457*, 2015.

[33] A. Petrowski, «A clearing procedure as a niching method for genetic algorithms», in *Proceedings of IEEE International Conference on Evolutionary Computation*, 1996, pp. 798–803. doi: 10.1109/ICEC.1996.542703.

[34] X. Yin e N. Germay, «A Fast Genetic Algorithm with Sharing Scheme Using Cluster Analysis Methods in Multimodal Function Optimization», in *Artificial Neural Nets and Genetic Algorithms*, R. F. Albrecht, C. R. Reeves, e N. C. Steele, A c. di, Vienna: Springer Vienna, 1993, pp. 450–457.

[35] K. A. De Jong, «An analysis of the behavior of a class of genetic adaptive systems.», University of Michigan, USA, 1975.

[36] S. W. Mahfoud, «Crowding and Preselection Revisited», in *Parallel Problem Solving from Nature*, 1992. [Online]. Disponibile su: https://api.semanticscholar.org/CorpusID:5438739

[37] O. J. Mengshoel e D. E. Goldberg, «Probabilistic crowding: Deterministic crowding with probabilistic replacement», 1999.

[38] G. R. Harik, «Finding Multimodal Solutions Using Restricted Tournament Selection», in *Proceedings of the 6th International Conference on Genetic Algorithms*, San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1995, pp. 24–31.

[39] J.-P. Li, M. E. Balazs, G. T. Parks, e P. J. Clarkson, «A Species Conserving Genetic Algorithm for Multimodal Function Optimization», *Evolutionary Computation*, vol. 10, fasc. 3, pp. 207–234, 2002, doi: 10.1162/106365602760234081.

[40] K.-C. Wong, «Evolutionary multimodal optimization: A short survey», *arXiv preprint arXiv:1508.00457*, 2015.

[41] E. Yu e P. Suganthan, «Ensemble of niching algorithms», *Information Sciences*, vol. 180, fasc. 15, pp. 2815–2833, 2010, doi: 10.1016/j.ins.2010.04.008.

[42] D. Wolpert e W. Macready, «No free lunch theorems for optimization», *IEEE Transactions on Evolutionary Computation*, vol. 1, fasc. 1, pp. 67–82, 1997, doi: 10.1109/4235.585893.

[43] D. Beasley, D. R. Bull, e R. R. Martin, «A Sequential Niche Technique for Multimodal Function Optimization», *Evolutionary Computation*, vol. 1, fasc. 2, pp. 101–125, 1993, doi: 10.1162/evco.1993.1.2.101.

[44] K. Deb e K. Deb, «Multi-objective Optimization», in *Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques*, E. K. Burke e G. Kendall, A c. di, Boston, MA: Springer US, 2014, pp. 403–449. doi: 10.1007/978-1-4614-6940-7_15.

[45] W. Li, T. Zhang, R. Wang, S. Huang, e J. Liang, «Multimodal multi-objective optimization: Comparative study of the state-of-the-art», *Swarm and Evolutionary Computation*, vol. 77, p. 101253–101254, 2023, doi: 10.1016/j.swevo.2023.101253.

[46] M. Muaafa, «Multi-Criteria Decision-Making Frameworks for Surveillance and Logistics Applications», 2015. doi: 10.13140/RG.2.1.1346.3280.

[47] R. Wang *et al.*, «Preference-inspired coevolutionary algorithm with active diversity strategy for multi-objective multi-modal optimization», *Information Sciences*, vol. 546, pp. 1148–1165, 2021, doi: 10.1016/j.ins.2020.09.075.

[48] K. Deb, A. Pratap, S. Agarwal, e T. Meyarivan, «A fast and elitist multiobjective genetic algorithm: NSGA-II», *IEEE Transactions on Evolutionary Computation*, vol. 6, fasc. 2, pp. 182–197, 2002, doi: 10.1109/4235.996017.

[49] Y. Peng e H. Ishibuchi, «Niching Diversity Estimation for Multi-modal Multi-objective Optimization», in *Evolutionary Multi-Criterion Optimization*, H. Ishibuchi, Q. Zhang, R. Cheng, K. Li, H. Li, H. Wang, e A. Zhou, A c. di, Cham: Springer International Publishing, 2021, pp. 323–334.

[50] C. A. C. Coello, *Evolutionary algorithms for solving multi-objective problems*. Springer, 2007. [Online]. Disponibile su: http://refhub.elsevier.com/S2210-6502(23)00027-5/sb49

[51] M. Javadi e S. Mostaghim, «Using Neighborhood-Based Density Measures for Multimodal Multi-objective Optimization», in *Evolutionary Multi-Criterion Optimization*, H. Ishibuchi, Q. Zhang, R. Cheng, K. Li, H. Li, H. Wang, e A. Zhou, A c. di, Cham: Springer International Publishing, 2021, pp. 335–345.

[52] A. Zhou, Q. Zhang, e Y. Jin, «Approximating the set of Pareto-optimal solutions in both the decision and objective spaces by an estimation of distribution algorithm», *IEEE transactions on evolutionary computation*, vol. 13, fasc. 5, pp. 1167–1189, 2009, doi: 10.1109/TEVC.2009.2021467.

[53] C. Yue, B. Qu, e J. Liang, «A Multi-objective Particle Swarm Optimizer Using Ring Topology for Solving Multimodal Multi-objective Problems», *IEEE Transactions on Evolutionary Computation*, vol. 22, pp. 805–817, 2017, doi: 10.1109/TEVC.2017.2754271.

[54] C. Yue, B. Qu, K. Yu, J. Liang, e X. Li, «A novel scalable test problem suite for multimodal multiobjective optimization», *Swarm and Evolutionary Computation*, vol. 48, pp. 62–71, 2019, doi: 10.1016/j.swevo.2019.03.011.

[55] K. Deb e S. Tiwari, «Omni-optimizer: A generic evolutionary algorithm for single and multi-objective optimization», *European Journal of Operational Research*, vol. 185, fasc. 3, pp. 1062–1087, 2008, doi: 10.1016/j.ejor.2006.06.042.

[56] J. J. Liang, C. T. Yue, e B. Y. Qu, «Multimodal multi-objective optimization: A preliminary study», in *2016 IEEE Congress on Evolutionary Computation (CEC)*, 2016, pp. 2454–2461. doi: 10.1109/CEC.2016.7744093.

[57] C. Yue, B. Qu, e J. Liang, «A Multiobjective Particle Swarm Optimizer Using Ring Topology for Solving Multimodal Multiobjective Problems», *IEEE Transactions on Evolutionary Computation*, vol. 22, fasc. 5, pp. 805–817, 2018, doi: 10.1109/TEVC.2017.2754271.

[58] B. Qu, C. Li, J. Liang, L. Yan, K. Yu, e Y. Zhu, «A self-organized speciation based multi-objective particle swarm optimizer for multimodal multi-objective problems», *Applied Soft Computing*, vol. 86, p. 105886–105887, 2020, doi: 10.1016/j.asoc.2019.105886.

[59] Y. Liu, H. Ishibuchi, Y. Nojima, N. Masuyama, e K. Shang, «A Double-Niched Evolutionary Algorithm and Its Behavior on Polygon-Based Problems», in *Parallel Problem Solving from Nature – PPSN XV*, A. Auger, C. M. Fonseca, N. Lourenço, P. Machado, L. Paquete, e D. Whitley, A c. di, Cham: Springer International Publishing, 2018, pp. 262–273.

[60] Y. Liu, G. G. Yen, e D. Gong, «A Multimodal Multiobjective Evolutionary Algorithm Using Two-Archive and Recombination Strategies», *IEEE Transactions on Evolutionary Computation*, vol. 23, fasc. 4, pp. 660–674, 2019, doi: 10.1109/TEVC.2018.2879406.

[61] Y. Liu, H. Ishibuchi, Y. Nojima, N. Masuyama, e Y. Han, «Searching for Local Pareto Optimal Solutions: A Case Study on Polygon-Based Problems», in *2019 IEEE Congress on Evolutionary Computation (CEC)*, 2019, pp. 896–903. doi: 10.1109/CEC.2019.8790066.

[62] Y. Liu, H. Ishibuchi, G. Yen, Y. Nojima, e N. Masuyama, «Handling Imbalance Between Convergence and Diversity in the Decision Space in Evolutionary Multi-Modal Multi-Objective Optimization», *IEEE Transactions on Evolutionary Computation*, p. 1–2, 2019, doi: 10.1109/TEVC.2019.2938557.

[63] B. Qu, C. Li, J. Liang, L. Yan, K. Yu, e Y. Zhu, «A self-organized speciation based multi-objective particle swarm optimizer for multimodal multi-objective problems», *Applied Soft Computing*, vol. 86, p. 105886–105887, 2020, doi: 10.1016/j.asoc.2019.105886.

[64] J. Liang *et al.*, «A clustering-based differential evolution algorithm for solving multimodal multi-objective optimization problems», *Swarm and Evolutionary Computation*, vol. 60, p. 100788–100789, 2021, doi: 10.1016/j.swevo.2020.100788.

[65] Y. Tian, R. Liu, X. Zhang, H. Ma, K. C. Tan, e Y. Jin, «A Multipopulation Evolutionary Algorithm for Solving Large-Scale Multimodal Multiobjective Optimization Problems», *IEEE Transactions on Evolutionary Computation*, vol. 25, fasc. 3, pp. 405–418, 2021, doi: 10.1109/TEVC.2020.3044711.

[66] Q. Lin, W. Lin, Z. Zhu, M. Gong, J. Li, e C. A. C. Coello, «Multimodal Multiobjective Evolutionary Optimization With Dual Clustering in Decision and Objective Spaces», *IEEE Transactions on Evolutionary Computation*, vol. 25, fasc. 1, pp. 130–144, 2021, doi: 10.1109/TEVC.2020.3008822.

[67] C. Yue *et al.*, «Differential evolution using improved crowding distance for multimodal multiobjective optimization», *Swarm and Evolutionary Computation*, vol. 62, p. 100849–100850, 2021, doi: 10.1016/j.swevo.2021.100849.

[68] W. Li, T. Zhang, R. Wang, e H. Ishibuchi, «Weighted Indicator-Based Evolutionary Algorithm for Multimodal Multiobjective Optimization», *IEEE Transactions on Evolutionary Computation*, vol. 25, fasc. 6, pp. 1064–1078, 2021, doi: 10.1109/TEVC.2021.3078441.

[69] W. Li, X. Yao, T. Zhang, R. Wang, e L. Wang, «Hierarchy Ranking Method for Multimodal Multiobjective Optimization With Local Pareto Fronts», *IEEE Transactions on Evolutionary Computation*, vol. 27, fasc. 1, pp. 98–110, 2023, doi: 10.1109/TEVC.2022.3155757.

[70] J. Liang, P. Suganthan, B. Qu, D. Gong, e C. Yue, «Problem Definitions and Evaluation Criteria for the CEC 2020 Special Session on Multimodal Multiobjective Optimization». 2019. doi: 10.13140/RG.2.2.31746.02247.

[71] Y. Peng, H. Ishibuchi, e K. Shang, «Multi-modal Multi-objective Optimization: Problem Analysis and Case Studies», in *2019 IEEE Symposium Series on Computational Intelligence (SSCI)*, 2019, pp. 1865–1872. doi: 10.1109/SSCI44817.2019.9002937.

[72] H. Ishibuchi, Y. Peng, e K. Shang, «A Scalable Multimodal Multiobjective Test Problem», in *2019 IEEE Congress on Evolutionary Computation (CEC)*, 2019, pp. 310–317. doi: 10.1109/CEC.2019.8789971.

[73] M. Friedman, «A Comparison of Alternative Tests of Significance for the Problem of m Rankings», *The Annals of Mathematical Statistics*, vol. 11, fasc. 1, pp. 86–92, 1940, Consultato: 6 agosto 2024. [Online]. Disponibile su: http://www.jstor.org/stable/2235971