

# Algoritmi metaeuristici

Algoritmi per l'intelligenza artificiale

Vincenzo Bonnici

Corso di Laurea Magistrale in Scienze Informatiche

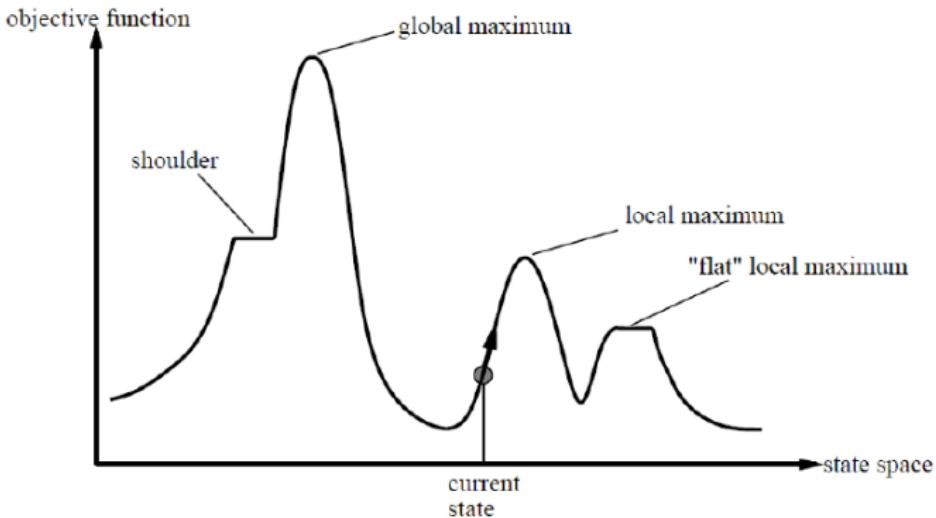
Dipartimento di Scienze Matematiche, Fisiche e Informatiche

Università degli Studi di Parma

2025-2026

- Utilizzati in problemi di ottimizzazione
- Tengono traccia solo dello stato corrente e si spostano su stati adiacenti
- Necessario il concetto di vicinato di uno stato
- Non si tiene traccia dei cammini
- Ad ogni stato è associata una funzione obiettivo
- Nessuna garanzia di raggiungere l'ottimo globale ma prestazioni efficienti
- Garanzia di raggiungere un ottimo locale

# Panorama dello spazio degli stati



- Uno stato ha una posizione sulla superficie e una altezza che corrisponde al valore della funzione obiettivo
- Un algoritmo di ottimizzazione provoca movimento sulla superficie
- Trovare l'avallamento più basso o il picco più alto

Dato lo **stato attuale**  $i$  e la **funzione obiettivo**  $f$ :

- Considerato l'insieme  $N(i)$  degli stati “vicini” di  $i$ , occorre definire un criterio per scegliere il **successivo stato**:
  - il migliore, cioè quello che migliora di più la  $f$ : Hill climbing a salita rapida
  - uno a caso tra quelli che migliorano: Hill climbing stocastico
  - il primo: Hill climbing con prima scelta
- **Criteri di arresto**: per evitare che l'algoritmo impieghi troppo tempo in caso di spazi ad alta dimensione, occorre definire dei criteri di arresto:
  - Numero massimo di iterazioni
  - Miglioramenti esigui
    - Terminiamo quando nessuno degli stati vicini migliora la  $f$  di una quantità superiore ad un fissato valore  $\epsilon$
  - Soluzione ottima localmente
    - Nessuna soluzione nel vicinato migliora quella attuale (massimo locale)

```
function Hill-climbing ( $S, N, f$ )
    // S spazio degli stati
    // N : S → P(S) funzione di vicinato
    // f : S → R funzione obiettivo
     $s \leftarrow x \text{ in } S$  // s Stato iniziale.
    loop do
         $vicino \leftarrow x \text{ in } N(S)$  // x scelto con criterio (i)
        if  $f(vicino) \leq f(s)$  then
            return  $s$  // interrompe la ricerca. s è massimo locale
         $s = vicino$ 
```

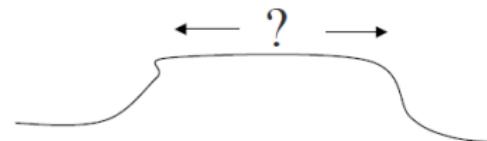
## Ricerca in salita (Hill climbing): problemi tipici

Se la f. è da massimizzare, si cercano i picchi (massimi locali o globali), tuttavia possiamo essere bloccati da problemi come

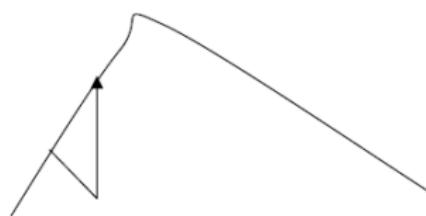
- massimi locali



- pianori o spalle



- crinali (o creste)



- Hill-climbing **stocastico**: si sceglie a caso tra le mosse in salita (che migliorano  $f$ )
  - converge più lentamente ma a volte trova soluzioni migliori
- Hill-climbing **con scelta casuale**
  - generare le mosse a caso ed essere più efficace quando i successori sono molti
  - Convergenza lenta ma può evitare massimi locali non buoni
- Hill-Climbing **riavvio casuale** (random restart): ripartire da un punto scelto a caso
  - Se la probabilità di successo è  $p$  saranno necessarie in media  $1/p$  ripartenze per trovare la soluzione
  - Se funziona o no dipende molto dalla forma del panorama degli stati

Analogia con il processo di solidificazione di un metallo fuso (Physical Annelaling).

- A partire dal metallo fuso, la **temperatura** viene **abbassata lentamente** e il sistema **transita** da uno **stato energetico** al successivo fino a quando il metallo solidifica nello stato di **minima energia** che rappresentata l'**ottimo globale**.

Passi:

- Descrizione delle configurazioni
- Il sistema viene portato alla **temperatura di fusione**
- Generazione casuale di configurazioni
- il sistema si trova in uno stato  $i$  e può **transitare** in uno stato  $j$  ottenuto perturbando lo stato  $i$
- Calcolo della funzione obiettivo
- la transizione avviene seguendo il **criterio di Metropolis**  $P = e^{\frac{E_i - E_j}{K_b T}}$
- la temperatura deve essere abbassata lentamente in modo che il solido raggiunga l'**equilibrio termico** ad ogni temperatura

Assumendo che lo stato  $X \in \{-1, 1\}^n$ .

Dato  $x(t)$ , stato al tempo  $t$ , scegliamo casualmente  $i \in \{1, 2, \dots, n\}$  e settiamo  $x_i = x_i(\text{flip})$  ottenendo  $x(t + 1)$ .

Una scelta alternativa può essere di scegliere in un ordine casuale tutti gli  $i \in \{1, 2, \dots, n\}$  in  $n$  scelte consecutive, cioè evitando di scegliere di aggiornare due volte la stessa componente prima di aver aggiornato una volta tutte le altre componenti.

# Simulated annealing: massimizzazione del criterio di Metropolis

Calcolo della funzione obiettivo del nuovo stato  $x(t + 1)$  e confronto con il precedente stato  $x(t)$  per il calcolo della probabilità di accettazione  $P_T\{(t + 1)\}$  di  $x(t + 1)$ .

$$P_T\{(t + 1)\} = \begin{cases} 1 & \text{se } E(x(t - 1)) \geq E(x(t)) \\ e^{\frac{E(x(t + 1)) - E(x(t))}{T}} & \text{se } E(x(t - 1)) < E(x(t)) \end{cases}$$

Generazione di un numero random  $r \in [0, 1]$  e confronto con  $P_t\{t + 1\}$ : se  $P_T\{x(t + 1)\} < r$  allora la nuova configurazione viene scartata.

Ad ogni passo  $t + 1$  si sceglie il nuovo stato a caso:

- se migliora lo stato corrente  $x(t)$  allora viene espanso
- altrimenti (caso in cui  $E(x(t+1)) - E(x(t)) < 0$ ) lo stato  $x(t+1)$  viene scelto comunque con probabilità  $e^{\frac{E(x(t+1)) - E(x(t))}{T}}$

$T$  decresce col progredire di  $t$  (quindi anche la probabilità di accettare peggioramenti) secondo un piano definito (valore iniziale e decremento sono parametri).

Accettare peggioramenti all'inizio permette di esplorare meglio lo spazio delle soluzioni.

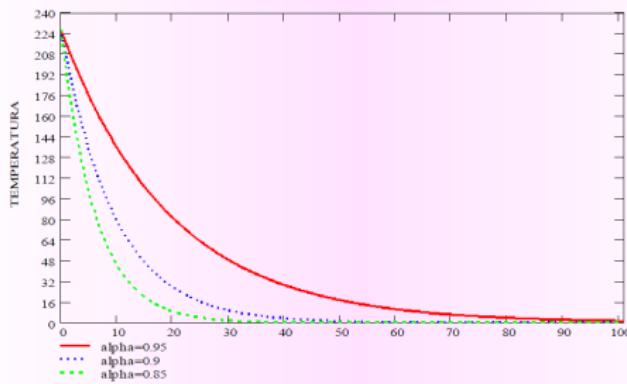
# Simulated annealing: schema di raffreddamento

**Parametri per il raffreddamento:**

- temperatura iniziale
- numero di passi a temperatura costante
- funzione di decremento
- criterio di arresto

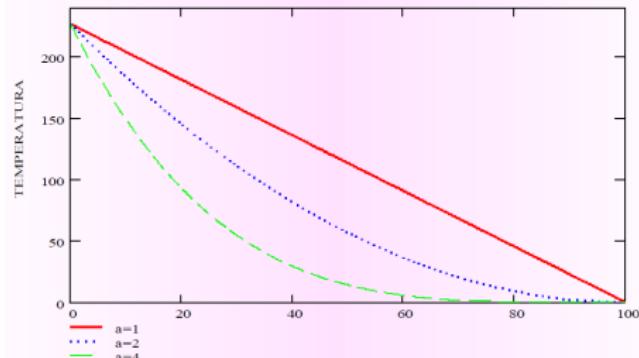
Il concetto fondamentale che guida la scelta di questi parametri è quello di **quasi equilibrio**

$$T_{k+1} = \alpha T_k \quad 0.8 \leq \alpha \leq 0.99$$



$$T_k = T_0 (1 - k / K)^a \quad a = 1, 2 \text{ o } 4$$

$K = \text{numero max decrementi}$



---

## Funzione simulatedAnnealing(*I*)

---

```
1: s = init(I), T = initT(), t = 0;
2: do                                     // Raffredda il sistema gradualmente
3:   do                               // Data T, prova qualche soluzione
4:     selezione una soluzione s' nell'intorno di s;
5:     v = m(I, s) - m(I, s');
6:     if (v > 0) then s = s'; // Soluzioni migliori si accettano sempre
7:     else                                // le altre talvolta
8:       if (rnd[0, 1] <  $e^{\frac{v}{T}}$ ) then s = s';
9:   while (! termCond());
10:  T = update(T, t); t++;
11: while (! haltCond());
12: return s;
```

---

Criterio di **arresto**:

- al raggiungimento del valore ottimo (noto)
- numero massimo di iterazioni, o temperatura sotto un certo valore
- miglioramento di energia al di sotto di un valore prefissato
- combinazione di criteri

**Idea:** rispetto a SA, scegli una soluzione vicina da esplorare basata sul privilegiare quelle che sono in regioni non ancora esplorate **proibendo** (tabú) di ritornare, anche in modo parziale, sui propri passi nell'immediato.

L'obiettivo è raggiunto **memorizzando** le scelte fatte circa le componenti della soluzione corrente e proibendo di riusare tali scelte.

Data una soluzione, si ricerca quindi una nuova soluzione, anche peggiore, nell'intorno di quella corrente evitando quelle che hanno portato a quest'ultima anche se sono migliori.

Esistono diversi tipi di **proibizioni** (tabu):

- Proibizione **fissa**: si impediscono le mosse recentemente fatte per un periodo fissato, poi si riammettono.
- Proibizione **randomizzata**: si impediscono le mosse recentemente fatte per un periodo casuale, poi si riammettono.
- Proibizione **reattiva**: si impediscono le mosse recentemente fatte per un periodo dipendente dalla qualità delle mosse; Date due mosse, se la prima aveva portato un miglioramento della soluzione maggiore della seconda, il suo tempo di proibizione è inferiore a quello della seconda.

- $M = \{\mu_1, \dots, \mu_p\}$  l'insieme delle possibili mosse
- $S_t$  la soluzione al tempo  $t$  e  $I(S_t)$  le soluzioni vicine a  $S_t$   
 $I(S_t) = \{S | S \text{ è ottenibile da } S_t \text{ applicando 1 mossa } \mu_i \in M\}$
- $T$ , parametro di proibizione fissa, è la quantità di tempo di proibizione di riuso di una mossa
- $I_P(S_t) \subset I(S_t)$  è l'insieme delle soluzioni permesse, determinate a partire da  $S_t$  applicando una mossa che non è stata usata durante le ultime  $T$  iterazioni
- la ricerca della soluzione  $S_{t+1}$  avviene all'interno dell'insieme  $I_P(S_T)$
- $UU[\mu_i]$  è l'ultima iterazione  $t$  nella quale è stata usata  $\mu_i$
- $\mu_i^{-1}$  la mossa inversa a  $\mu_i$
- ...continua...

- Criterio della migliore mossa permessa:
  - $I_P(S_t) = \{S | S \text{ è ottenibile da } S_t \text{ applicando 1 mossa } \mu_i \in M \text{ tale che } UU[\mu_i^{-1}] < t - T\}$
  - $S_{t+1} = S \in I_P(S_t)$  tale che  $m(I, S) \leq m(I, S') \forall S' \in I_P(S_t)$
- non è garantito  $m(I, S_{t+1}) \leq m(I, S_t)$ : ci sono mosse proibite
- se  $T = 0$  non ci sono proibizioni, che equivale alla ricerca locale classica
- maggiore è  $T$  maggiore è il numero di iterazioni prima di rivisitare una soluzione
- $T$  non può essere troppo grande: tutte le mosse diventerebbero proibite e la ricerca si bloccherebbe
- $T$  deve garantire almeno 2 mosse per iterazione

Anziché determinare valori buoni per  $T$ , si può tentare un approccio causale:

- ogni  $n$  iterazioni, si scegli casualmente un nuovo  $T_f$  (parametro di proibizione fissa): in questo modo un cattivo valore influenza un numero finito di iterazioni
- per rendere efficace l'approccio, l'algoritmo viene eseguito più volte, partendo sempre dalla soluzione greedy. La soluzione finale è data dalla soluzione migliore trovata durante tutte le esecuzioni.
- Prove sperimentali hanno mostrato che eseguendo  $100n$  volte il programma e, per ciascuna volta, eseguendo  $10n$  iterazioni, le soluzioni trovate sono paragonabili a quelle trovate dalla ricerca a tabu fissa con il miglior  $T_f$

Soluzioni migliori si possono ottenere con la ricerca reattiva:

- anzichè scegliere in modo casuale il valore  $T_f$ , si cerca di scegliere un buon valore il più spesso possibile
- la qualità di  $T_f$  è determinata in due modi
  - mediante una fase di preprocessamento che analizza le proprietà del problema
  - misurando i miglioramenti delle soluzioni determinate con un certo  $T_f$  durante l'esecuzione dell'algoritmo stesso
- la fase di preprocessamento prevede di eseguire delle prove con valori di  $T_f$  fissati e con diversi valori di iterazioni e di classificare i diversi  $T_f$  in funzione della qualità della soluzione che hanno determinato
- al termine del preprocessamento, la proibizione più piccola con voto maggiore, insieme alle altre, vengono usate in ordine nell'algoritmo vero e proprio con regole precise.

# Local beam search (ricerca locale a raggio/fascio)

**Idea:** rispetto a SA, si considerano  $k$  stati precedenti invece che 1.

Si scelgono i top  $k$  stati come successori.

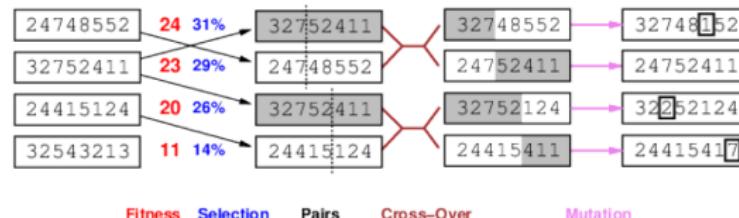
Ogni dei  $k$  successori può essere eseguito in parallelo.

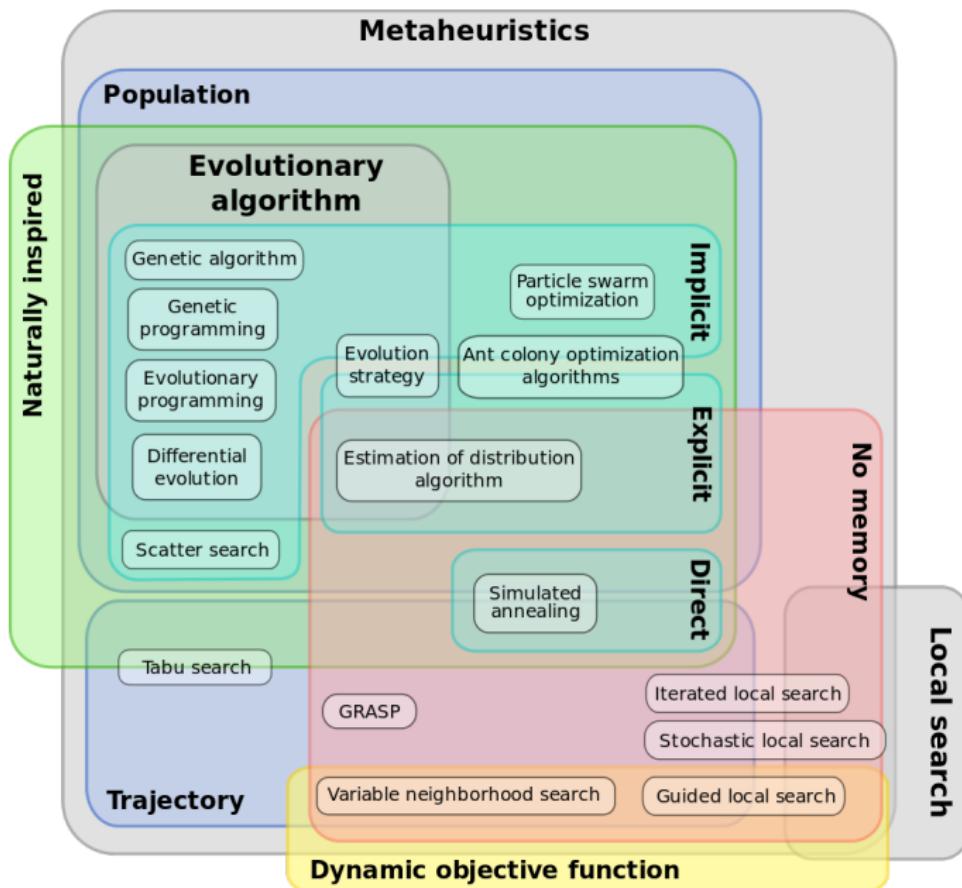
Le ricerche su tali successori che portano a buone soluzioni vengono reclutate per creare nuove soluzioni tramite la loro "unione".

**Problema** spesso tutti i  $k$  successori finiscono sullo stesso ottimo locale

**Idea:** scegli i  $k$  successori randomicamente con una preferenza verso quelli migliori.

**Osservazione:** tale processo assomiglia molto a quello della selezione naturale! Algoritmi genetici = local beam search + generazione dei successori da coppie di stati

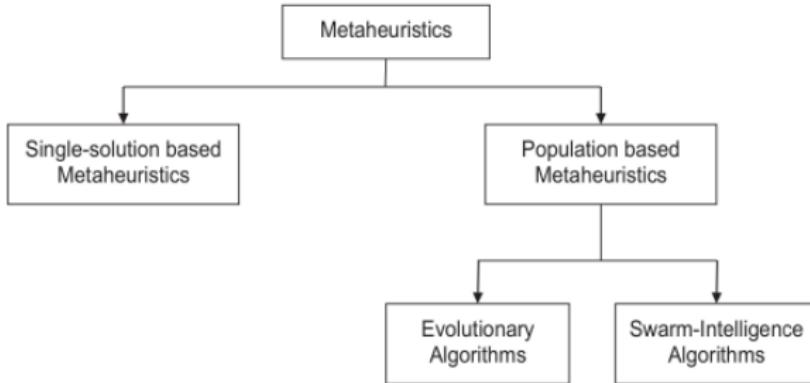




Una **metaeuristica** è una procedura di livello superiore o euristica progettata per trovare, generare o selezionare un'euristica (algoritmo di ricerca parziale) che può fornire una soluzione sufficientemente buona a un problema di ottimizzazione, specialmente con **informazioni incomplete** o imperfette o **capacità di calcolo limitata** e più in generale per risolvere problemi "difficili".

Punti chiave:

- **diversificazione**: ricerca non confinata ad un numero ridotto di regioni.
- **Intensificazione**: esplorazione in profondità delle regioni promettenti nella speranza di trovare una soluzione migliore.
- non è garantito che verrà trovata una soluzione ottima globale



Categorie:

- Basati su **soluzione singola**: singola soluzione candidata che viene migliorata mediante ricerca locale.  
! Possibile blocco in ottimi locali.
- Basati su **popolazione**: molteplici soluzioni candidate durante il processo di ricerca.  
! Diversità nella popolazione.  
! Evita blocco in ottimi locali.

La Natura offre una vastissima gamma di esempi di capacità di **ottimizzazione**.

Da un lato i meccanismi **evolutivi** naturali portano alla creazione di individui particolarmente adatti al loro habitat.

Dall'altro esseri viventi estremamente semplici sono in grado insieme di svolgere compiti complessi (**intelligenza collettiva**).

Molti modelli sono **stocastici** e danno luogo ad algoritmi **randomizzati**. Ovviamente solo alcuni aspetti naturali sono realmente interessanti dal punto di vista algoritmico.

## Algoritmi genetici

- Studiati in maniera approfondita per la prima volta da Holland (1975)
- Delineati a grandi linee già nell'articolo di Turing Computing Machinery and Intelligence (1950)
- Molto utilizzati e studiati sia dal punto di vista teorico che applicativo
- Usando una rappresentazione **binaria** si potrebbero applicare a qualsiasi problema di ottimizzazione
- Si basano sulla teoria darwiniana dell'**evoluzione**

## Applicazioni

Esistono un numero impressionante di applicazioni degli algoritmi genetici. Si possono usare sia per l'ottimizzazione di funzioni a variabile reale sia per ottimizzazione combinatorica.

Gli algoritmi genetici sono anche stati implementati per problemi decisionali (ad esempio SAT)

## Principio darwiniano della selezione naturale

Sopravvive chi si adatta meglio

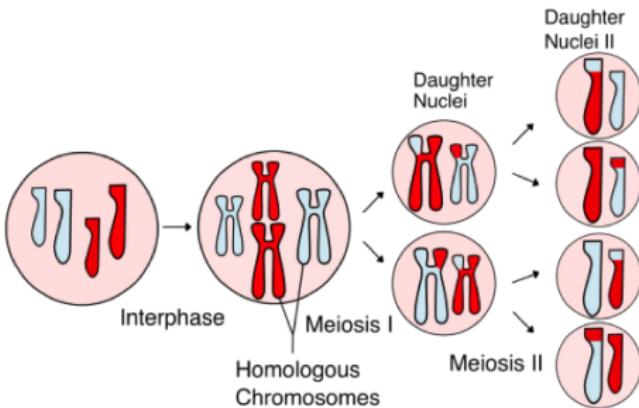
- **Riproduzione**: formazione di nuovi individui che ereditano il **patrimonio genetico** dai genitori
- **Mutazioni genetiche casuali**: indispensabili per l'**evoluzione** della specie
- **Scopo**: ottenere popolazioni di individui adatti all'**ambiente** (che sopravvivono)

Punti chiave:

- **Rappresentazione dei cromosomi:** metodo di codifica dell'informazione all'interno dell'algoritmo.
- **Funzione di fitness:** assegnazione di un valore di "bontà" ad ogni cromosoma.
- **Selezione:** selezione dei cromosomi in base alla funzione di fitness.
- **Crossover:** creazione di una nuova discendenza a partire dai cromosomi selezionati.
- **Mutazione:** eventuale modifica casuale dei valori nei cromosomi.

## Rappresentazione

- Individui rappresentati mediante il loro patrimonio genetico (**genotipo**)
- **Cromosoma**: stringa di lunghezza fissa  $L$
- Ogni simbolo (**gene**) è preso da un alfabeto finito di elementi (**alleli**)
- Possibile presenza di **vincoli**: non tutte le stringhe rappresentano cromosomi validi



## Fitness

- Definizione di una **funzione fitness**  $f$
- Individui con alta fitness sono i più **adatti all'ambiente**: hanno maggiori probabilità di sopravvivere e di riprodursi
- Scopo dell'AG (algoritmo genetico): trovare l'individuo (o gli individui) con la **fitness maggiore**
- La fitness potrebbe essere diversa dalla funzione da massimizzare (essere una trasformata monotona crescente)

## Popolazione

- La popolazione viene **inizializzata** in qualche modo (ad esempio con  $N$  individui generati **a caso**)
- Il numero di individui è (tipicamente) mantenuto fisso
- La popolazione è **manipolata** per un certo numero di generazioni mediante
  - Selezione; Riproduzione; Mutazione

# Algoritmi genetici: algoritmo

---

***Input:***

*Population Size, n*

*Maximum number of iterations, MAX*

***Output:***

*Global best solution,  $Y_{bt}$*

---

***begin***

*Generate initial population of n chromosomes  $Y_i$  ( $i = 1, 2, \dots, n$ )*

*Set iteration counter  $t = 0$*

*Compute the fitness value of each chromosomes*

***while*** ( $t < MAX$ )

*Select a pair of chromosomes from initial population based on fitness*

*Apply crossover operation on selected pair with crossover probability*

*Apply mutation on the offspring with mutation probability*

*Replace old population with newly generated population*

*Increment the current iteration  $t$  by 1.*

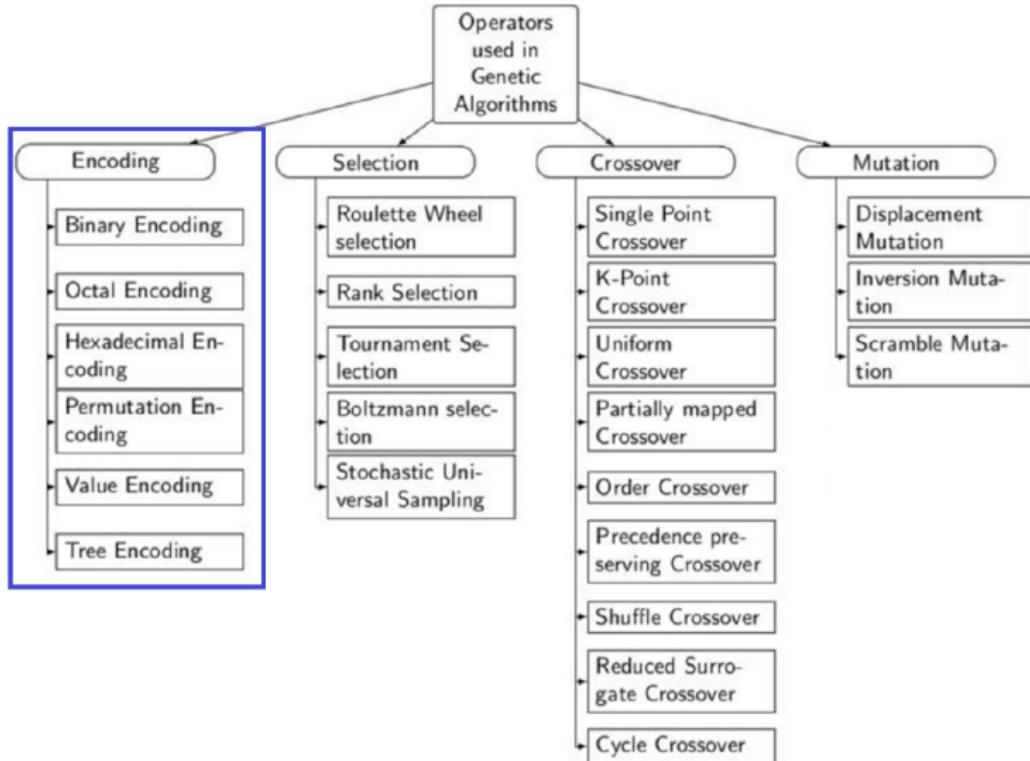
***end while***

*return the best solution,  $Y_{bt}$*

---

***end***

# Algoritmi genetici: operatori



# Algoritmi genetici: codifica

Una cella/posizione = un gene

## Codifica binaria

1	0	0	1	0	1	0	1	1	0
---	---	---	---	---	---	---	---	---	---

- Facile implementazione e rapida esecuzione.
- Accuratezza dipendente dalla precisione della conversione.
- Per problemi che supportano la codifica binaria.

## Codifica ottale e esadecimale

3	2	5	1	5	0	0	1	7	4	A	2	E	B	6	9	F	3	1	C
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

- Facile implementazione.
- Valore gene = numero ottale o esadecimale.
- Utilizzo limitato ad applicazioni ad hoc.

## Codifica permutativa

7	4	2	6	1	8	3	9	10	5
---	---	---	---	---	---	---	---	----	---

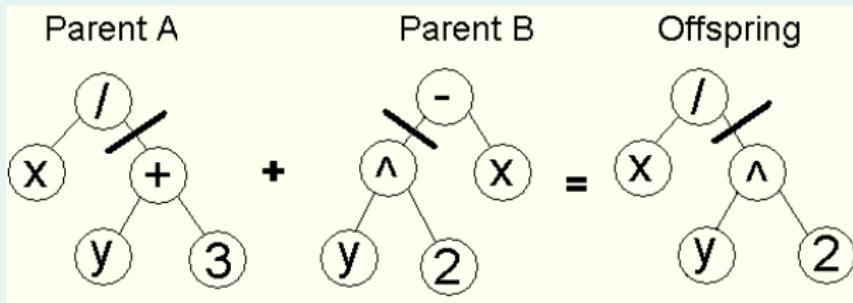
- Valore gene = posizione all'interno della sequenza.
- Utilizzata nei problemi di ordinamento es. TSP.

## Codifica ottale e esadecimale

0.5	1.2	2.0	0.3	1.6	0.7	0.5	1.1	2.9	0.9
-----	-----	-----	-----	-----	-----	-----	-----	-----	-----

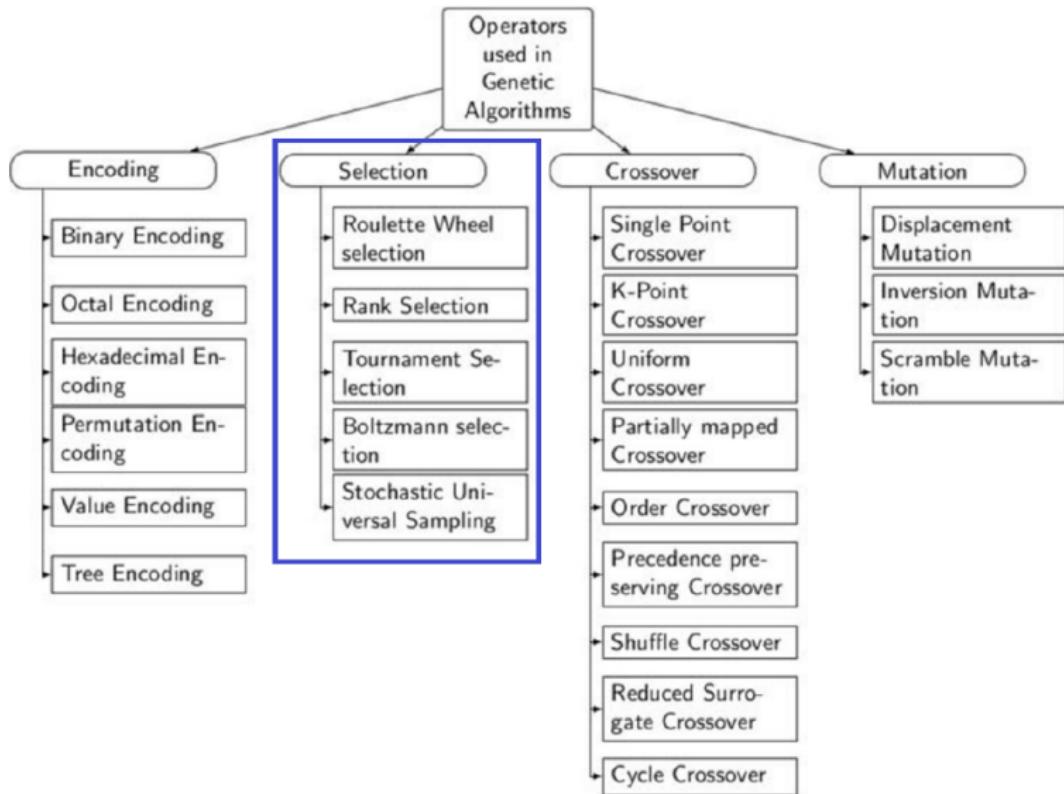
- Valore gene = intero, reale, carattere, ...
- Utilizzato in diverse applicazioni reali con valori complessi es. ricerca di pesi per le reti neurali.

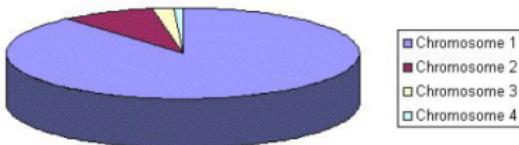
## Codifica ad albero



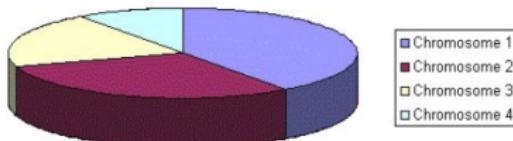
- Valore gene = nodo di un albero di funzioni o comandi.
- Utilizzato per programmazione evolutiva.

# Algoritmi genetici: selezione



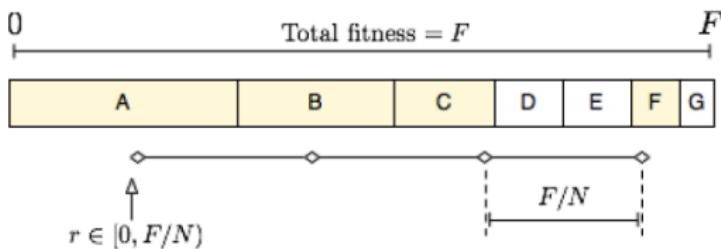


- **Funzionamento:** mappa i cromosomi su una ruota, assegnandogli una porzione proporzionale al loro valore di fitness, poi si effettua una selezione casuale "girando la ruota".
- **Pro:** metodo semplice e di facile implementazione.
- **Contro:** rischio di convergenza prematura influenzato dalla varianza nella funzione di fitness.



- **Funzionamento:** variante del metodo precedente che utilizza il concetto di rank: ordina i cromosomi in base al valore di fitness ed assegna una porzione di ruota proporzionale alla posizione in classifica. Dati  $n$  individui, il peggiore ha rank 1, il migliore ha rank  $n$ .
- **Pro:** preserva diversità nella popolazione.
- **Contro:** convergenza lenta, computazionalmente costoso.

- **Funzionamento:**
  - ① Selezionare  $k =$  numero di cromosomi desiderati
  - ② Selezionare  $j =$  numero di partecipanti al torneo
  - ③ Selezionare  $p =$  pressione selettiva (probabilità)
  - ④ Effettuare un torneo con classifica basata sulla fitness
  - ⑤ Selezionare un cromosoma in base a  $p$ :  $i$ -esimo classificato selezionato con probabilità  $p(1 - p)^{i-1}$
  - ⑥ Ripetere dal passo 4 finché non si hanno  $k$  cromosomi
- **Pro:** preserva diversità nella popolazione, possibile parallelizzazione.
- **Contro:** perdita della diversità al crescere di  $j$ .

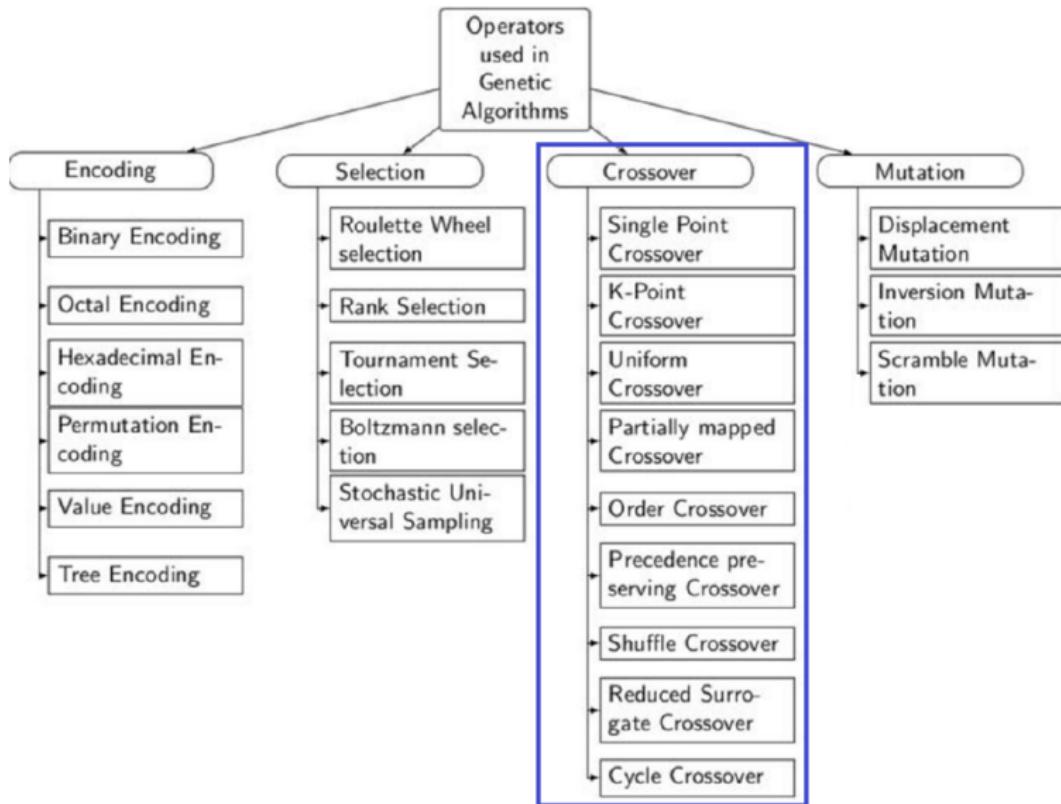


- **Funzionamento:** Estensione del metodo della roulette; con  $k$  individui da scegliere, posiziono un punto casuale sulla ruota ed altri  $k - 1$  punti equidistanti tra loro.
- **Pro:** metodo semplice e di facile implementazione.
- **Contro:** convergenza prematura, non ha buona prestazioni su alcuni problemi es. TSP di grandi dimensioni.

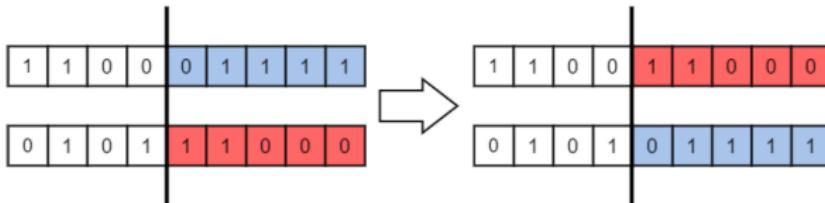
- **Funzionamento:** Basato su entropia e metodi di campionamento. Pressione di selezione controllata da un parametro  $T$  (temperatura), che varia continuamente in base a un piano preimpostato.
  - Inizio: valori alti per  $T \mapsto$  pressione selettiva bassa.
  - Fine: valori bassi per  $T \mapsto$  pressione selettiva alta.
- **Pro:** diversità nella popolazione mantenuta, alta precisione.
- **Contro:** Computazionalmente oneroso, soggetto a perdita di informazione → Soluzione: selezione elitaria.

- **Funzionamento:** Meccanismo ausiliario per migliorare le performance degli operatori di selezione: assicura che l'individuo migliore venga sempre selezionato per la creazione della generazione successiva, aggiungendolo manualmente qualora il processo normale non lo selezioni.

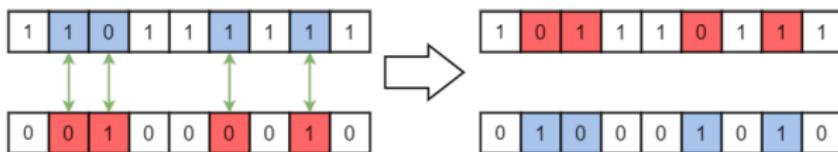
# Algoritmi genetici: crossover



# Crossover - Punto singolo



- **Funzionamento:** Scambio dell'informazione genetica dei genitori a partire da un punto di sovrapposizione scelto casualmente.  
Generalizzazione: crossover a  $k$  punti.
- **Pro:** facile implementazione.
- **Contro:** Possibile ottenere una prole poco differenziata rispetto ai genitori.



- **Funzionamento:** Ogni gene di un genitore viene scambiato con una certa probabilità con l'elemento corrispondente dell'altro genitore.
- **Pro:** No bias nell'esplorazione, buon potenziale ricombinatorio.
- **Contro:** Possibile ottenere una prole poco differenziata rispetto ai genitori.

- **Funzionamento:**

- ① Scegli un segmento di  $P1$  da copiare nel figlio
- ② A partire dal primo punto di sovrapposizione, guarda quale elemento di  $P2$  non è stato copiato
- ③ Per ciascuno di questi geni  $i$ , guarda quale elemento  $j$  è stato copiato da  $P1$  al suo posto
- ④ Metti  $i$  nella posizione occupata da  $j$  in  $P2$
- ⑤ Se la posizione occupata da  $j$  è già occupata da un altro valore  $k$ , metti  $i$  nella posizione occupata da  $k$  in  $P2$
- ⑥ Ripeti finché non trovi una posizione valida per  $i$
- ⑦ Riempì le restanti posizioni del figlio con i valori corrispondenti da  $P2$

- **Pro:** Velocità di convergenza bilanciata.

# Crossover - Match parziale (PMX)

- Step 1

1	2	3	4	5	6	7	8	9
---	---	---	---	---	---	---	---	---



			4	5	6	7		
--	--	--	---	---	---	---	--	--

- Step 2

9	3	7	8	2	6	5	1	4
---	---	---	---	---	---	---	---	---

1	2	3	4	5	6	7	8	9
---	---	---	---	---	---	---	---	---



i	j
8	4
2	5

	2	4	5	6	7	8		
--	---	---	---	---	---	---	--	--

- Step 3

9	3	7	8	2	6	5	1	4
---	---	---	---	---	---	---	---	---

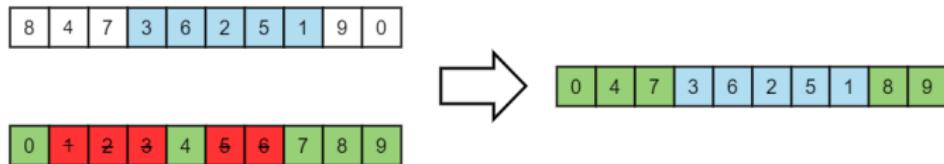
1	2	3	4	5	6	7	8	9
---	---	---	---	---	---	---	---	---



i	j	k
2	5	7

9	3	2	4	5	6	7	1	8
---	---	---	---	---	---	---	---	---

# Crossover - Ordinato (OX)

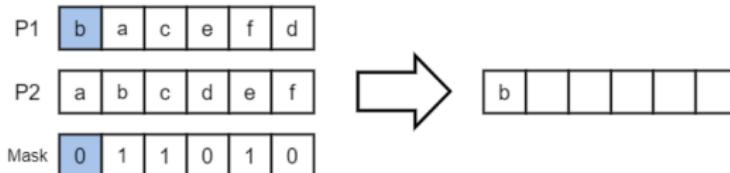


- **Funzionamento:** Copia una o più parti di un genitore, poi riempie le restanti posizioni con valori non già presenti presi dal secondo genitore, seguendo un criterio d'ordine (es. in ordine di comparsa).
- **Pro:** Buona esplorazione nello spazio di ricerca.
- **Contro:** Perdita di informazione sugli individui delle generazioni precedenti.

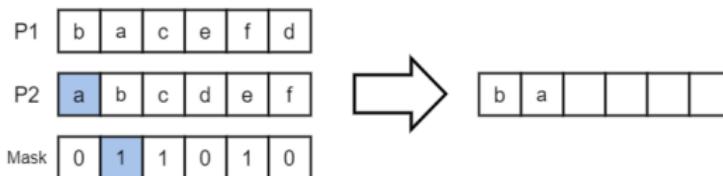
- **Funzionamento:** Utilizza una maschera che specifica quale dei genitori dovrà contribuire all' $i$ -esimo gene.  
Per ciascun gene della prole, va a prendere il primo gene del genitore in ordine di comparsa che non è ancora stato utilizzato all'interno del figlio.
- **Pro:** Buona generazione della prole.
- **Contro:** Possibili ridondanze.

# Crossover - Precedence preserving (PPX)

Step 1

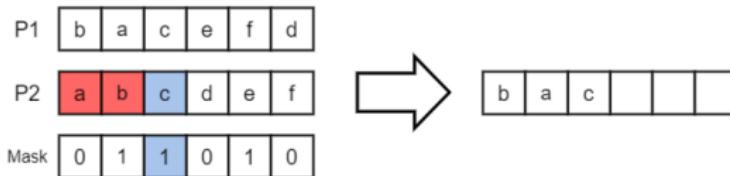


Step 2

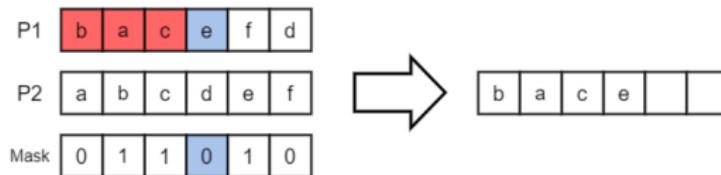


# Crossover - Precedence preserving (PPX)

Step 3

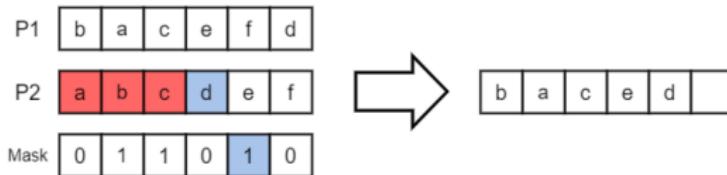


Step 4

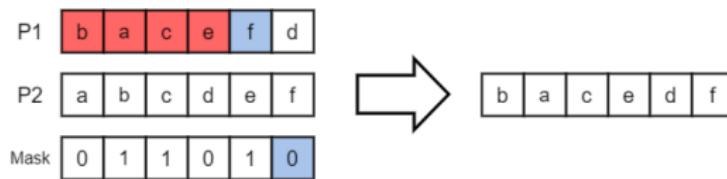


# Crossover - Precedence preserving (PPX)

Step 5



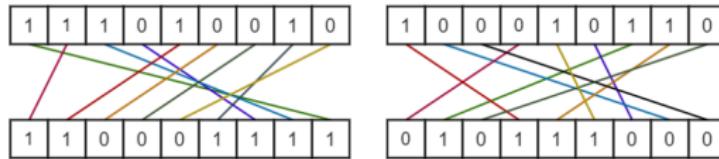
Step 6



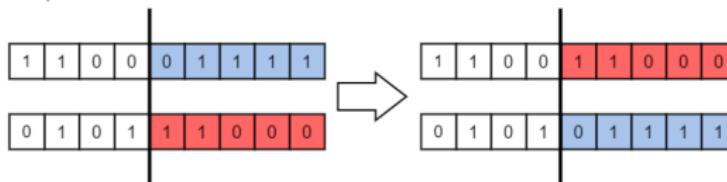
- **Funzionamento:**
  - ① Mescola i geni di ciascun genitore secondo uno schema
  - ② Effettua crossover a punto singolo
  - ③ Riordina i geni di ciascun genitore secondo lo schema usato nel passo 1.
- **Pro:** Aiuta la diversificazione nella popolazione.
- **Contro:** Uso in tempi recenti limitato.

# Crossover - Precedence preserving (PPX)

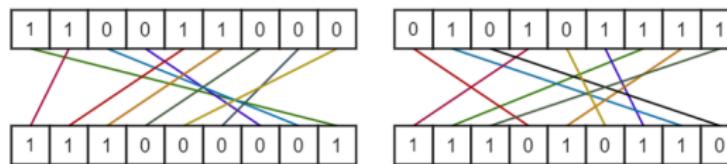
Step 1



Step 2



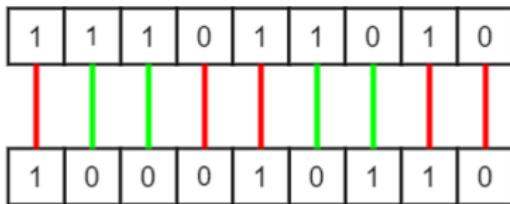
Step 3



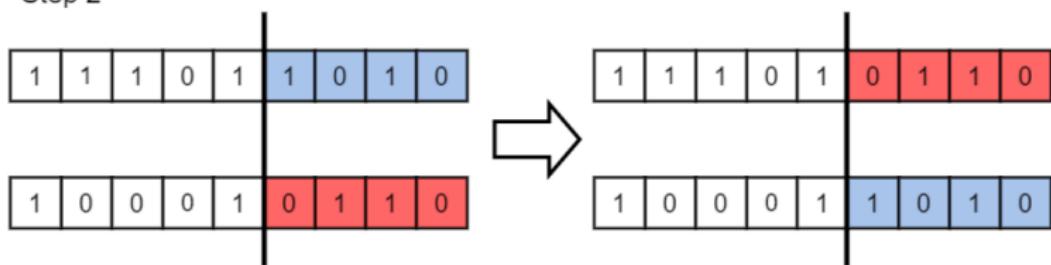
- **Funzionamento:**
  - ① Ispeziona i genitori e crea una lista di tutte le posizioni in cui i genitori hanno geni diversi
  - ② Seleziona un punto casuale tra quelli individuati
  - ③ Effettua un crossover a punto singolo
- **Pro:** Mantiene la diversità nella popolazione e ha buone performance su problemi di ottimizzazione piccoli.
- **Contro:** Convergenza prematura.

# Crossover - Precedence preserving (PPX)

Step 1



Step 2



- **Funzionamento:**

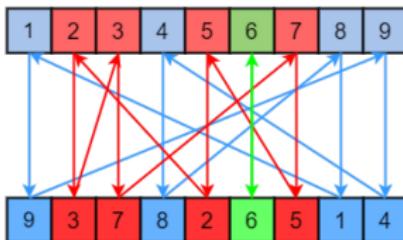
- ① Individua i cicli nei genitori  $P_1$  e  $P_2$ 
  - ① Prendi il primo elemento  $i$  di  $P_1$  non ancora appartenente ad un ciclo
  - ② Guarda l'elemento  $j$  di  $P_2$  nella stessa posizione di  $i$
  - ③ Vai all'elemento di  $P_1$  contenente lo stesso valore di  $j$
  - ④ Ripeti i passi 2 e 3 finché non torni ad  $i$
- ② Metti gli elementi del ciclo 1 appartenenti a  $P_1$  nel primo figlio e quelli appartenenti a  $P_2$  al secondo figlio
- ③ Metti gli elementi del ciclo 2 appartenenti a  $P_2$  nel primo figlio e quelli appartenenti a  $P_1$  al secondo figlio
- ④ Ripeti l'alternanza per ogni ciclo rimasto

- **Pro:** Esplorazione senza bias.

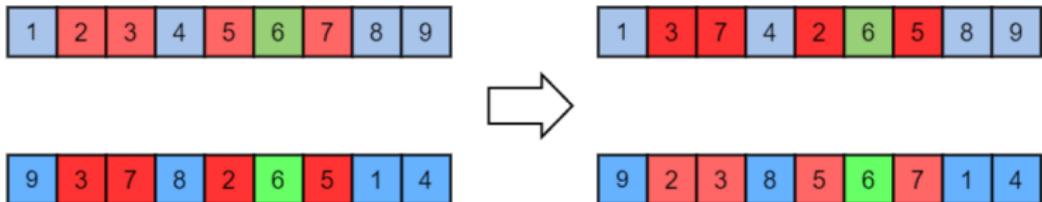
- **Contro:** Convergenza prematura.

# Crossover - Precedence preserving (PPX)

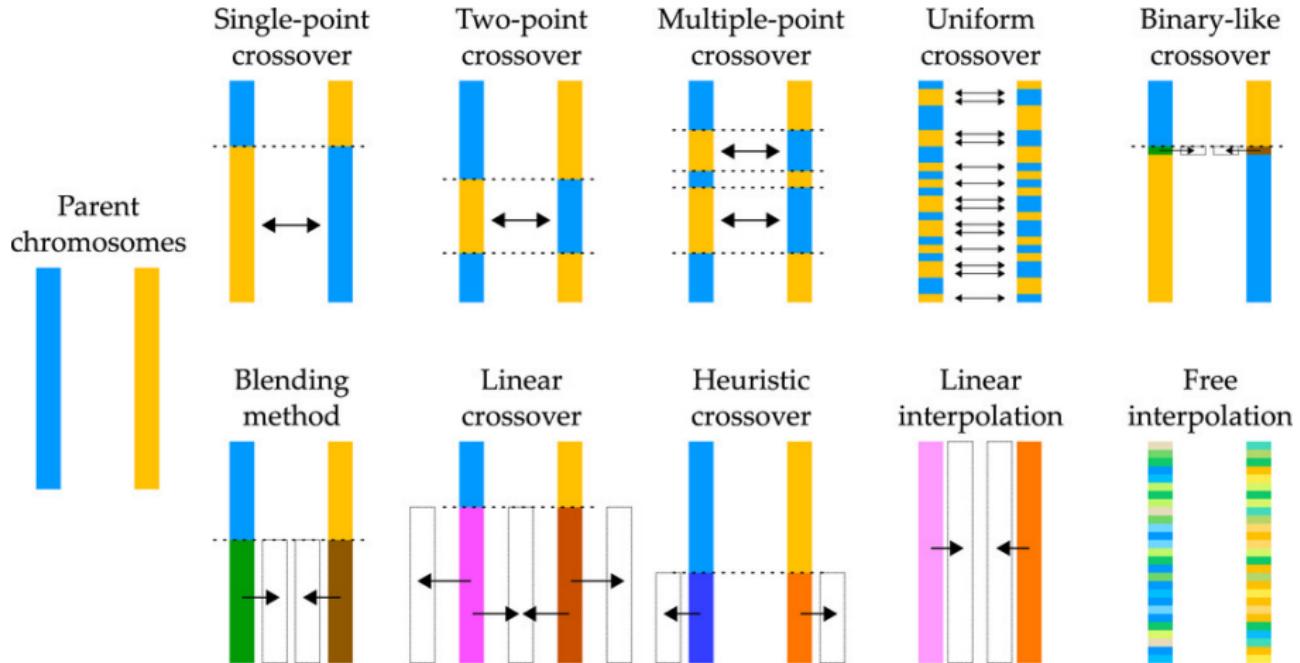
Step 1



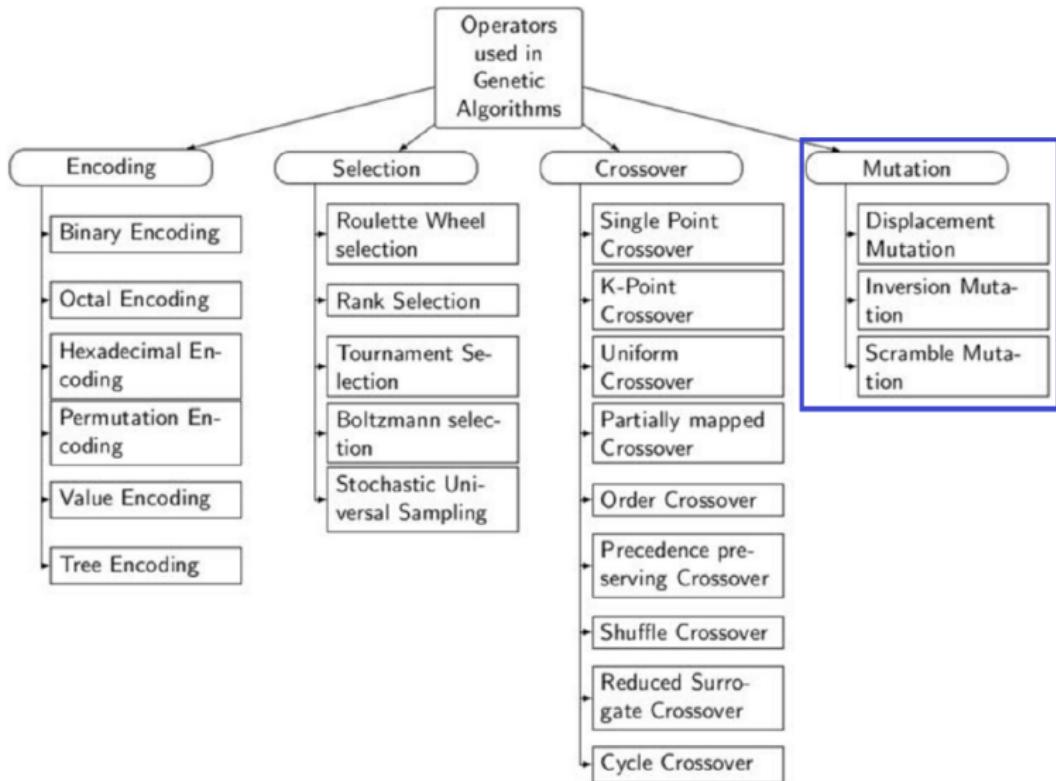
Step 2



# Crossover -



# Algoritmi genetici: mutazione



# Mutazione - Operatori principali

## Dislocamento



## Inversione



## Scramble



## Binary Coded Genetic Algorithm

- Variante più nota, basata su rappresentazione binaria dei cromosomi.
- Implementazione veloce.
- Richiesta conversione in formato binario.
- Picchi di Hamming e schema impari.
- Bassa precisione.

## Real Coded Genetic Algorithm

- Rappresentazione basata sul valore (numeri reali).
- Algoritmo semplice, robusto, efficiente ed accurato.
- Convergenza prematura.
- Necessario ridefinire operatori di crossover e di mutazione.

## Multi-Objective Genetic Algorithm

- Ideato per problemi di ottimizzazione multiobiettivo.
- Obiettivi: convergenza, diversità, copertura.
- varianti:
  - Basati su Pareto:
    - Basati sul concetto di efficienza paretiana.
    - Performance scarse su problemi reali.
  - Basati su decomposizione:
    - Decompongono il problema in sottoproblemi più semplici.
    - Sottoproblemi risolvibili in parallelo.
    - Possibile cooperazione tra sottoproblemi simili mediante scambio di soluzioni.

## Chaotic Genetic Algorithm

- Basato sulla teoria del caos.
- Utilizza funzioni caotiche per operazioni randomiche all'interno degli operatori genetici.
- Alleviano il problema della convergenza prematura.
- Ottimi per problemi di ottimizzazione.

## Hybrid Genetic Algorithm

- Combinazione di GA e altre tecniche.
- Obiettivo: creare un algoritmo efficiente ed efficace mantenendo i punti forti dei singoli componenti.
- Esempi
  - GA + logica fuzzy.
  - item GA + ricerca locale (TS, SA, ...).
  - GA + intelligenza di sciamo (ACO, PSO, ...).

# Swarm intelligence (Intelligenza di sciame)

Sono algoritmi ispirati al comportamento di **insetti/animali**.

Caratteristiche tipiche:

- **Fase esplorativa**: esplorano individualmente lo spazio di ricerca aumentando la loro conoscenza.
- **Meccanismo di condivisione**: condividono la conoscenza acquisita con gli altri membri.  
Permette di far dirigere lo sciame verso posizioni (caratteristiche di una soluzione) migliori.

Algoritmi (più famosi):

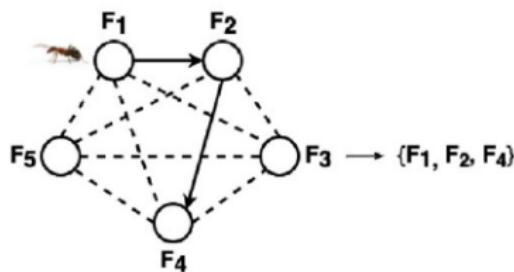
- ACO (Ant Colony Optimization)
- PSO (Particle Swarm Optimization)
- ABC (Artificial Bee Colony optimization)
- ...
- Pigeon Inspired Optimization (PIO)

# Ant Colony Optimization (ACO)

Algoritmi che usano popolazioni di **formiche** per risolvere problemi di ottimizzazione in uno spazio di ricerca a **grafo** (completo).

A partire da un grafo:

- ogni formica può muoversi lungo gli archi che connettono due nodi
- ad ogni arco è associato un livello di **feromone**
- il **percorso** fatto da una formica rappresenta una **soluzione**, valutata con una funzione di **fitness**
- la bontà della soluzione fa aggiornare il feromone
- le successive formiche saranno attratte dagli archi con più feromone



Ogni feature viene rappresentata da un nodo nel grafo.

Ad ogni iterazione, un insieme di formiche costruisce simultaneamente il proprio sottoinsieme di feature:

- una formica inizia con un nodo casuale
- per una formica al nodo  $i$ , il prossimo nodo è deciso dalla probabilità:

$$p_{i,j} = \begin{cases} \frac{\tau_{i,j}^\alpha \cdot \nu_{i,j}^\beta}{\sum_{k \in J_i} \tau_{i,k}^\alpha \cdot \nu_{i,k}^\beta} & \text{per } j \in J_i \\ 0 & \text{altrimenti} \end{cases}$$

dove  $J_i$  è l'insieme dei vicini di  $i$  non ancora visitati dalla specifica formica,  $\tau_{i,j}^\alpha$  e  $\nu_{i,j}^\beta$  sono rispettivamente il feromone e una informazione euristica (spesso information gain) associati all'arco  $i - j$  controllati dai parametri  $\alpha$  e  $\beta$

- si ripete finché un criterio di terminazione non è soddisfatto.

I sottoinsiemi vengono valutati da una fitness function.  
Solo la formica migliore aggiorna i valori di feromone.

Criterio di terminazione:

- numero di feature da selezionare
- numero di feature da selezionare random
- numero di feature da selezionare + accuratezza della classificazione

Problema convergenza prematura:

- Modifica delle regole di aggiornamento feromone e valore euristico: regola locale + regola globale; Aggiornamento in due fasi diverse
- Bilanciamento tra exploration/exploitation: utilizzo di due soluzioni (migliore globale e migliore locale)

Tecniche per la riduzione della complessità:

- livello di feromone e il valore euristico assegnati a ciascun nodo
- si forza un ordine tra le feature
- raggruppando per similarità le feature

Tecniche classiche per l'aggiornamento:

- aggiornamento valori feromone: performance della classificazione
- aggiornamento valori euristici: filtro misura come informazione mutuale e F-score.

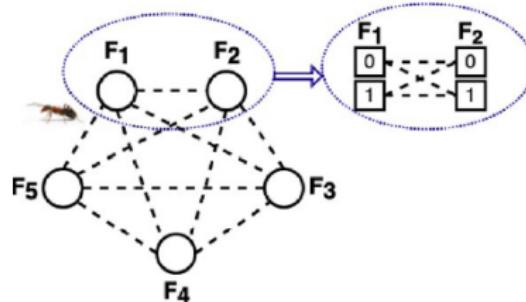
Alternative:

- valore feromone e valore euristico calcolati sulla percentuale di guadagno
- valore euristico come misura di similarità
- valore euristico calcolato attraverso la correlazione di Pearson

Feature selection formulata come un insieme di decisioni binarie. Ogni feature come due sottonodi:

- sottonodo 0 (feature non selezionata)
- sottonodo 1 (feature selezionata)

Quindi, tra due feature ci sono quattro archi.



Funzionamento::

- una formica attraversa tutti i nodi
- ad ogni nodo, una formica può visitare il sottonodo 0 o il sottonodo 1
- in base a quale visita indica se quella feature viene selezionata oppure no

Rappresentazione standard:

- richiede un criterio di terminazione
- spazi di ricerca limitati

Rappresentazione binaria:

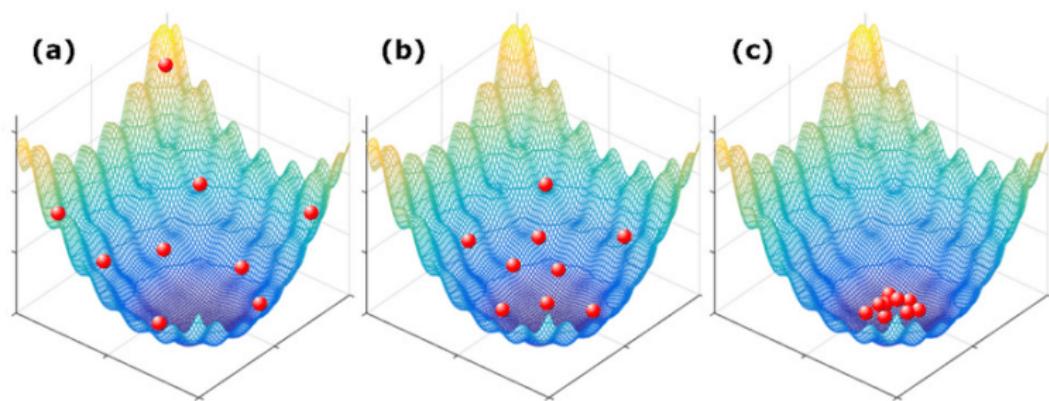
- non necessita di un criterio di terminazione
- grande spazio di ricerca avendo quattro archi che connettono coppie di feature.

ACO in generale:

- il feromone e i valori euristici sono definiti tra due feature
- nel mondo reale interazioni tra tre o più feature sono comuni
- multi-objective: poco studiati

# Particle swarm optimizzazione (PSO)

- Originariamente implementato per l'ottimizzazione numerica
- Utilizza uno sciame di particelle che si muovono simulando uno stormo di uccelli
- Ogni particella rappresenta una possibile soluzione



# Particle swarm optimizzazione (PSO)

- Ogni particella si muove liberamente all'interno dello spazio di ricerca
- Ad ogni iterazione, la posizione viene aggiornata utilizzando il concetto di **velocità** che considera
  - $g_{best}$ : la migliore soluzione globale
  - $p_{best}$ : la migliore soluzione personale
- La topologia descrive il sottoinsieme di particelle prese in considerazione durante l'aggiornamento della posizione (es. totalmente connessa, kNN, ...)
- Ad ogni iterazione, la posizione viene aggiornata utilizzando la velocità

$$V = wV + \phi_p r_p(p_{best} - x) + \phi_g r_g(g_{best} - x)$$

con  $w$  valore di inerzia;  $V$  velocità attuale;  $x$  posizione attuale;  $\phi_p$  coefficiente cognitivo;  $\phi_g$  coefficiente sociale;  $r_p, r_g \in U(0, 1)$ .

- Le feature diventano dimensioni nello spazio di ricerca
- La posizione in una dimensione è il valore della feature corrispondente
  - Standard/continua: un valore reale per ogni feature
  - Binaria: un valore booleano per ogni feature
- La feature selection viene eseguita selezionando le feature il cui valore è sopra una certa soglia  $\theta$  oppure 1 nel caso binario.
- Problema di convergenza prematura
  - Controllo dinamico dei parametri: modificare il valore di inerzia a seconda della diversità della sciame (distanza euclidea). Quando la diversità è bassa l'inerzia aumenta e viceversa
  - Competitive swarm optimizer:  $g_{best}$  e  $p_{best}$  sono rimossi. Le particelle competono per passare alla popolazione successiva. I perdenti aggiornano la posizione in base ai vincenti ed entrano nella popolazione successiva

- Approccio che cerca di massimizzare/minimizzare due o più funzioni contrastanti
- Non è presente una singola soluzione ottima ma una serie di soluzioni non dominate, cioè soluzioni che, migliorando uno dei due obiettivi, peggiorano l'altro e viceversa.
- Un'archivio contiene l'insieme delle soluzioni non dominate trovate durante l'esecuzione
- Versione standard: come PSO a singolo obiettivo ma  $g_{best}$  viene scelto tra gli elementi presenti nell'archivio
- Alternativa: applicazione di tre operatori sulle soluzioni in archivio: inserimento, rimozione e scambio

Versione standard:

- Si utilizza la funzione di aggiornamento classica
- La posizione viene normalizzata in un valore compreso tra 0 e 1 tramite una funzione(es. sigmoide)
- La nuova posizione viene convertita in binario confrontandola con un valore random o una soglia prestabilita

In uno spazio binario non c'è direzione, il movimento è rappresentato dall'inversione dei bit:

- La velocità è rappresentata dalla probabilità di invertire i bit e il momento dalla tendenza a rimanere nella posizione attuale

Tre **componenti** principali:

- **Risorse** (di cibo): la cui qualità è espressa da un valore (fitness);
- Api **employed**: associate ad una risorsa;
- Api **unemployed**: alla ricerca di risorse. Possono essere:
  - onlooker : usano le informazioni dell'alveare
  - scout: si muove a random

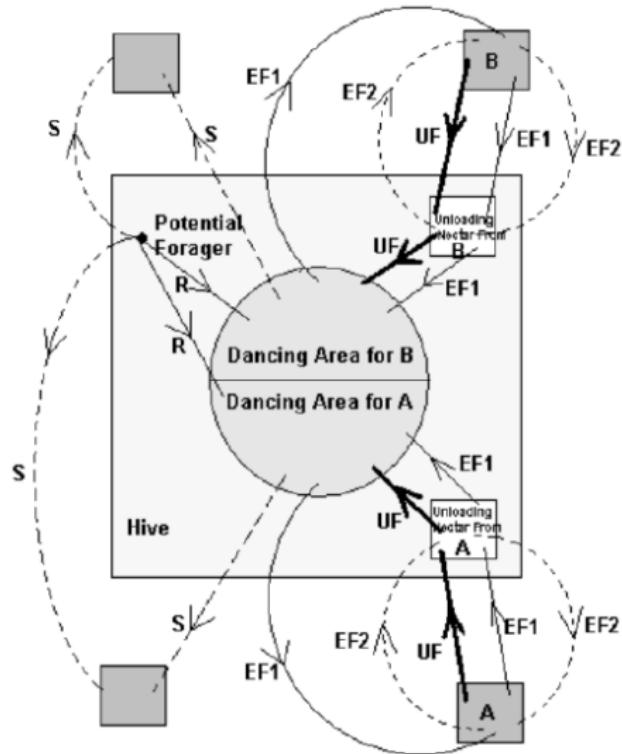
Due **comportamenti** principali:

- **Assunzione** (recruitment) di una risorsa
- **Abbandono** di una risorsa

Meccanismo di **condivisione**  $\mapsto$  danza.

- Una ape è inizialmente unemployed. Può essere scout o recluta
- Dopo aver identificato la risorsa di cibo, l'ape diventa impiegata, quindi
  - esplora la risorsa
  - memorizza le informazioni su di essa
  - Prende un carico di nettare e ritorna all'alveare
- ora ha tre opzioni:
  - Abbandonare la risorsa e diventare uncommitted follower (UF)
  - Danzare per reclutare altre api prima di ritornare alla stessa risorsa (EF1)
  - Continuare ad utilizzare la stessa risorsa senza reclutare altre api (EF2)

# Artificial Bee Colony (ABC)



Posizione di una risorsa di cibo  $\mapsto$  una possibile soluzione  $x \in \mathbb{R}^D$ .

Quantità di nettare di una risorsa  $\mapsto$  qualità (fitness) della soluzione.

Algoritmo:

- Inizializzazione random di  $x_i$  ( $i = 1, 2, \dots, SN$ ) soluzioni (popolazione).
- Cicli ripetuti di ricerca sulla popolazione da parte di:
  - api employed: producono una modifica alla posizione e ne testano la fitness; condividono le informazioni con la danza nell'alveare;
  - api unemployed: scelgono una nuova risorsa con probabilità legate alla fitness;
  - api scout: rimpiazzano risorse abbandonate (no miglioramenti per  $n$  tentativi) con delle nuove.

ABC utilizza una rappresentazione basata su vettori  $x \in \mathbb{R}^D$  per risolvere problemi di ottimizzazione che risulta naturale per la feature selection (D: numero di features).

Ogni risorsa (soluzione)  $\mapsto$  un sottoinsieme di feature.

(Come PSO) ABC può utilizzare rappresentazioni continue o binarie.

Rispetto agli altri algoritmi di SI:  
chiara separazione tra esplorazione (scout) e miglioramento (onlooker, employed) di soluzioni

# Artificial Bee Colony (ABC)

4 differenti processi di selezione:

- Selezione (probabilistica) globale delle regioni più promettenti da parte delle onlooker :

$$p_i = \frac{fit_i}{\sum_{n=1}^{SN} fit_i}$$

- Selezione (probabilistica) locale delle nuove risorse candidate nel vicinato di una risorsa:

$$V_{ij} = x_{ij} + \phi_{ij}(x_{ij} - x_{kj})$$

- Scelta greedy nella valutazione di quale risorsa usare (vecchia soluzione o nuova candidata);
- Selezione randomica di una nuova risorsa fatta dalle scout:

$$x_i^j = x_{min}^j + U(0, 1)(x_{max}^j - x_{min}^j)$$

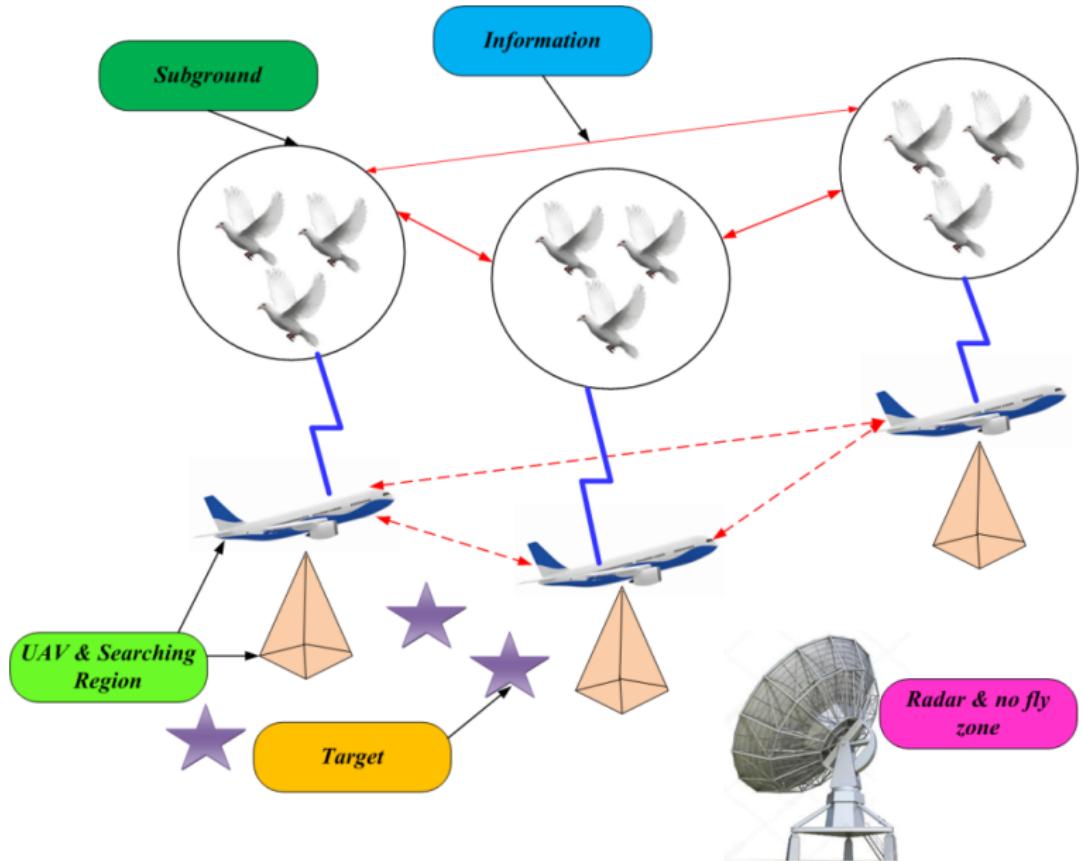
- La **pianificazione del percorso** è il problema di progettare il percorso che un veicolo dovrebbe seguire in modo tale da massimizzare e/o raggiungere un determinato obiettivo.
- I **piccioni** sono l'uccello più popolare al mondo e un tempo erano usati per inviare il messaggio dagli egiziani, cosa che si verificava anche in molti affari militari.
- I piccioni viaggiatori possono facilmente trovare la loro casa utilizzando tre strumenti di ricerca: **campo magnetico, sole e punti di riferimento**.



- In biologia, una popolazione può dividersi in qualche **sottogruppo**.
  - Di fronte ai nemici naturali, i sottogruppi **coopereranno** per resistere.
  - Tuttavia, **competono** anche tra loro nell'interesse del cibo, dell'accoppiamento, del territorio e così via.
  - La cooperazione e la concorrenza fanno sì che la popolazione sopravviva e si **evolva** meglio
- Per simulare questi comportamenti naturali, gli **UAV** lavorano insieme per completare l'attività di ricerca tramite l'interazione delle informazioni, nel frattempo gli UAV competono tra loro per cercare le cellule vitali specifiche.

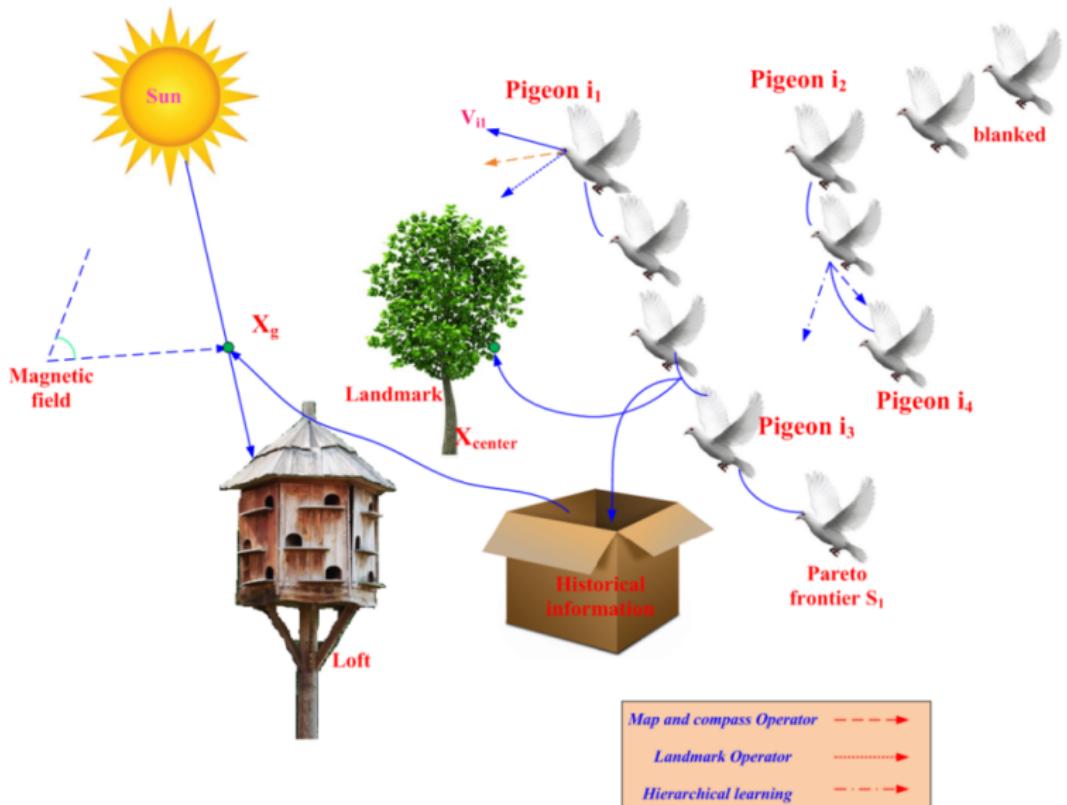


# Swarm intelligence - Pigeon Inspired Optimization (PIO)



- Analizzando i dati di volo raccolti dal GPS in miniatura durante più voli in stormo di piccioni, è stata scoperta una rete **gerarchica** nelle relazioni **leader-seguace** in volo dei piccioni.
  - In uno stormo di piccioni, eccetto il capo generale il cui movimento non sarà influenzato dall'altro piccione, ogni piccione ha il suo **rango** nella gerarchia.
  - Durante il volo, i piccioni cercheranno di **seguire** quelli nei ranghi superiori e **guidare** quelli nei ranghi inferiori.
  - Si ipotizza che la gerarchia di leadership sia il risultato di un **feedback** tra apprendimento e competenza.

# Swarm intelligence - Pigeon Inspired Optimization (PIO)



- I piccioni possono percepire il campo magnetico terrestre e prendono la posizione del sole come una bussola per formare una **mappa nei loro ricordi** che li guida nella giusta direzione.
- Nel frattempo, i piccioni hanno anche la capacità di riconoscere i **punti di riferimento** che hanno incontrato prima in modo da poter ottenere il **percorso migliore** verso la loro destinazione.
- L'algoritmo PIO riproduce completamente questi processi. Mentre si trovano nelle vicinanze dei punti di riferimento, i piccioni aggiornano le loro posizioni utilizzando la migliore posizione centrale di ogni iterazione. Attraverso queste due parti di aggiornamenti, i piccioni troveranno presto la migliore posizione globale della storia.

- Nel PIO di base, tutti i piccioni correggeranno le loro posizioni  $X_i$  in base
  - al sole e al campo magnetico descritti dall'attuale migliore posizione globale  $X_g$
  - dal messaggio di anteprima dell'immagine del punto di riferimento specificato dalla media pesata delle posizioni  $X_{center}$ .
- Nel PIO modificato tenendo conto della gerarchia, i piccioni sono divisi in due ruoli: uno è il leader generale e l'altro è il normale seguace.
- Inoltre, con l'ordinamento non dominato di Pareto, tutti i piccioni saranno divisi in insiemi diversi: prima frontiera  $S_1$ , seconda frontiera  $S_2$  e così via.
- Inoltre, un operatore di confronto di affollamento continuerà a ordinare i piccioni in ogni set.

Passi:

- ① Inizializza i parametri
- ② Valuta la fitness dei piccioni
- ③ Seleziona l'operatore da applicare
- ④ Aggiorna i piccioni
  - Se è selezionato l'operatore mappa e bussola, la velocità e la posizione di tutti i piccioni è aggiornata, altrimenti aggiorna la posizione individualmente
- ⑤ Aggiorna la posizione migliore globale
  - La posizione del piccione,  $X_g best$  è la migliore posizione globale che ha il valore minimo della funzione fitness nella nuova posizione del piccione.
- ⑥ Termina
  - Infine, i parametri nella compensazione della guida sono ottimizzati per ottenere una maggiore precisione di atterraggio con una minore integrazione dell'errore di altezza. Dopo la progettazione a quattro strati, l'accelerazione normale, nonché la sua oscillazione e la risposta della frequenza di beccheggio vengono verificate in base ai criteri