# Lezione su Euristiche e Meta-Euristiche per Ottimizzazione e Apprendimento

## Problemi nella Vita Reale

Nella vita quotidiana, prendere decisioni è complesso a causa di:

- \*\*Tempo limitato\*\*: Bisogna agire rapidamente.

- \*\*Incertezza\*\*: Dati incompleti o ambigui.

- \*\*Big Data\*\*: Grandi volumi di informazioni da elaborare.

- \*\*Dinamicità\*\*: Ambienti che cambiano velocemente.

\*\*Esempi\*\*:

- Risolvere un Sudoku o il rompicapo "8-puzzle" (ottimizzazione di mosse).

- Decisioni finanziarie o logistiche con vincoli multipli.

## Il Test di Turing e l'Intelligenza Artificiale

Per superare il \*\*Test di Turing\*\*, un sistema deve:

- Interpretare il linguaggio naturale.

- Ragionare automaticamente.

- Adattarsi a contesti incerti.

- Muoversi o manipolare oggetti (per robotica).

## Approcci all'Ottimizzazione

- \*\*Deterministici vs. Stocastici\*\*:

- \*Deterministici\* (es. Tabu Search): Stessi input → stessi output.

- \*Stocastici\* (es. Simulated Annealing): Soluzioni variabili per lo stesso input.

- \*\*Iterativi vs. Greedy\*\*:

- \*Iterativi\* (es. GA): Partono da soluzioni complete e le modificano.

- \*Greedy\* (es. algoritmi costruttivi): Costruiscono la soluzione passo-passo.

## Esempi Pratici

- \*\*8-puzzle\*\*: Trova la sequenza minima di mosse per ordinare le tessere.

- \*\*Spazio di ricerca\*\*: Tutte le possibili sequenze di mosse.

- \*\*Funzione di costo\*\*: Numero di mosse effettuate.

- \*\*Sudoku\*\*: Risolvere la griglia rispettando vincoli di righe/colonne/sottogriglie.

# Lezione su Ottimizzazione Complessa e Gestione dei Vincoli

## Introduzione

Il corso esplora le sfide legate alla risoluzione di problemi complessi, con particolare attenzione all'ottimizzazione vincolata e alla gestione dello spazio di ricerca. Gli algoritmi euristici e meta-euristici sono strumenti fondamentali per affrontare problemi NP-difficili o con vincoli stringenti.

---

## Complessità dei Problemi

Un problema è considerato \*\*complesso\*\* quando:

- La dimensione delle istanze di input è elevata.

- La struttura delle istanze è irregolare (es. molti ottimi locali).

- Il tempo di ricerca è critico (es. applicazioni in tempo reale).

\*\*Esempi\*\*:

- \*\*Proteine (PSP)\*\*: Lo spazio di ricerca può avere milioni di conformazioni con livelli di energia diversi (Tabella a pag. 9).

- \*\*Landscape combinatorio\*\*: Problemi "facili" hanno pochi picchi, mentre quelli "difficili" presentano superfici discontinue o plateau.

---

## Spazio di Ricerca e Ottimizzazione Vincolata

### Definizioni

- \*\*Spazio di ricerca (S)\*\*: Insieme di tutte le possibili soluzioni, anche non ammissibili.

- \*\*Spazio ammissibile (F)\*\*: Sottoinsieme di S che soddisfa i vincoli.

- \*\*Funzione obiettivo\*\*: Assegna un valore a ogni soluzione (es. minimizzare \( f(x) \)).

\*\*Esempio\*\*:

- \*\*Problema SAT\*\*:

- Configurazioni: Assegnazioni di verità per \( A \) e \( B \).

- Funzione obiettivo: Numero di clausole soddisfatte (pag. 3).

---

## Ottimizzazione Vincolata

Formulazione generale:

\[

\min\_x f(x) \quad \text{t.c.} \quad g\_i(x) \leq 0, \quad h\_j(x) = 0

\]

Dove:

- \( g\_i(x) \): Vincoli di disuguaglianza.

- \( h\_j(x) \): Vincoli di uguaglianza.

### Tecniche di Gestione dei Vincoli

1. \*\*Funzioni di Penalità\*\*:

- \*\*Statiche\*\*: Penalità fisse (es. \( \psi(x) = f(x) + r \sum g\_i(x)^2 \)).

- \*\*Dinamiche\*\*: Penalità crescono con le generazioni (es. \( \psi(x) = f(x) + (Ct)^\alpha \sum |g\_i(x)|^\beta \)).

- \*\*Adattive\*\*: Penalità si adattano in base alla fattibilità delle soluzioni (es. metodo Bean e Hadj-Alouane).

2. \*\*Algoritmi di Riparazione\*\*:

- Mappano soluzioni non ammissibili in soluzioni ammissibili (es. interpolazione lineare con un riferimento \( I\_r \)).

---

## Landscape Complesso e Ottimi Locali

### NK-Model

Modello teorico per studiare la rugosità del landscape:

- \*\*N\*\*: Numero di geni.

- \*\*K\*\*: Interdipendenza tra geni.

- All’aumentare di \( K \), il numero di ottimi locali cresce esponenzialmente (grafici a pag. 13-14).

### Differenze tra Problemi Facili e Difficili

- \*\*Facili\*\*:

- Superfici lisce, pochi ottimi locali.

- Soluzioni vicine hanno fitness simili.

- \*\*Difficili\*\*:

- Molti ottimi locali, plateau.

- Soluzioni vicine possono avere fitness molto diversi.

---

## Esempi Pratici

1. \*\*8-Puzzle\*\*:

- \*\*Spazio di ricerca\*\*: Sequenze di mosse.

- \*\*Vincoli\*\*: Solo mosse legali.

2. \*\*Progettazione di Reti\*\*:

- \*\*Vincoli\*\*: Latenza massima, costi infrastrutturali.

---

## Conclusioni

- \*\*Key Concepts\*\*:

- Codifica del problema, rappresentazione delle soluzioni, esplorazione dello spazio di ricerca.

- \*\*Strumenti\*\*:

- Penalità, riparazione, ibridazione di metodi.

- \*\*Obiettivo\*\*:

- Trovare soluzioni ammissibili e ottimali in tempi accettabili, anche per problemi NP-difficili.

# Lezione su Meta-Euristiche e Ottimizzazione Ispirata alla Natura

## Introduzione

Questo materiale didattico esplora l'uso di algoritmi bio-ispirati per risolvere problemi complessi di ottimizzazione, con particolare attenzione alle meta-euristiche evolutive e all'intelligenza di sciame.

## Problemi Complessi e Ottimizzazione

### Caratteristiche dei Problemi Complessi

- \*\*NP-completi\*\*: Problemi computazionalmente intrattabili per dimensioni elevate

- \*\*Molti ottimi locali\*\*: Difficoltà nell'individuare l'ottimo globale (pag. 3-4)

- \*\*Scenari dinamici\*\*: Ambienti con incertezza e vincoli temporali (pag. 5-6)

### Esempio Concreto

- \*\*Proteine (PSP)\*\*: Spazio di ricerca con oltre 83 milioni di conformazioni possibili a diversi livelli energetici (pag. 4)

## Ispirazione dalla Natura

### Teoria Evolutiva Applicata (pag. 10-11)

- \*\*Selezione naturale\*\*: Sopravvivenza dei più adatti

- \*\*Meccanismi chiave\*\*:

- Adattamento

- Crossover (ricombinazione)

- Mutazione

- Evoluzione

### Algoritmi Bio-Ispirati (pag. 12-14)

- \*\*Sistemi immunitari artificiali\*\*

- \*\*Intelligenza di sciame\*\*:

- Comportamento collettivo e decentralizzato

- Auto-adattamento

- Esempi: colonie di formiche, particle swarm optimization

## Confronto tra Approcci (pag. 21-24)

### Metodi Classici vs. Meta-Euristiche

| Caratteristica | Metodi Classici | Meta-Euristiche |

|---------------|----------------|----------------|

| \*\*Natura\*\* | Deterministici | Stocastici |

| \*\*Soluzioni\*\* | Riproducibili | Variabili |

| \*\*Approccio\*\* | Greedy | Iterativo |

| \*\*Focus\*\* | Singola soluzione | Popolazione di soluzioni |

### Vantaggi delle Meta-Euristiche

1. \*\*Robustezza\*\*: Gestione efficace dell'incertezza

2. \*\*Decentralizzazione\*\*: Nessun punto di controllo centrale

3. \*\*Adattabilità\*\*: Risposta a cambiamenti dinamici

4. \*\*Flessibilità\*\*: Applicabilità a diversi domini

5. \*\*Approccio black-box\*\*: Non richiedono conoscenza specifica del dominio

## Perché Sviluppare Nuovi Algoritmi? (pag. 20-22)

- I metodi greedy tradizionali spesso falliscono in problemi complessi

- Esempio TSP: soluzione greedy con costo 53 vs soluzione ottimale con costo 19

- Le meta-euristiche offrono:

- Migliore esplorazione dello spazio di ricerca

- Capacità di evitare ottimi locali

- Adattabilità a scenari reali complessi

## Conclusione

Le meta-euristiche bio-ispirate rappresentano un potente strumento per:

- Affrontare problemi NP-completi

- Gestire scenari dinamici e incerti

- Ottenere soluzioni approssimate in tempi accettabili

# Lezione su Algoritmi Genetici e NP-Completezza

## Parte 1: Algoritmi Genetici (GA)

### Introduzione agli Algoritmi Genetici

Gli algoritmi genetici sono meta-euristiche ispirate ai principi dell'evoluzione naturale:

- \*\*Selezione naturale\*\*: Sopravvivenza dei più adatti

- \*\*Operatori principali\*\*:

- Crossover (ricombinazione)

- Mutazione

- Selezione

### Schema di Funzionamento (pag. 17-18)

1. \*\*Inizializzazione\*\*: Generazione casuale di una popolazione iniziale

2. \*\*Valutazione\*\*: Calcolo del fitness per ogni individuo

3. \*\*Selezione\*\*: Scelta dei genitori basata sul fitness

4. \*\*Riproduzione\*\*: Applicazione di crossover e mutazione

5. \*\*Sostituzione\*\*: Creazione di una nuova generazione

6. \*\*Terminazione\*\*: Ripetizione fino a soddisfare i criteri di arresto

### Operatori Chiave (pag. 19-21)

1. \*\*Selezione\*\*: Roulette-wheel, tornei

2. \*\*Crossover\*\*:

- Single-point

- Two-point

- Uniforme

3. \*\*Mutazione\*\*: Alterazioni casuali per diversificazione

## Parte 2: Teoria della NP-Completezza

### Problemi e Complessità (pag. 3-15)

- \*\*Problema NP-completo\*\*: Problema decisionale per cui:

- Non si conosce un algoritmo polinomiale

- Ogni problema in NP è riducibile ad esso

- \*\*Esempi classici\*\*:

- TSP (Traveling Salesman Problem)

- Problema della colorazione di grafi

- Problema del ciclo hamiltoniano

### Classi di Complessità (pag. 16-21)

- \*\*P\*\*: Problemi risolvibili in tempo polinomiale

- \*\*NP\*\*: Problemi verificabili in tempo polinomiale

- \*\*NP-completi\*\*: I problemi più difficili in NP

- \*\*NP-hard\*\*: Problemi almeno tanto difficili quanto quelli NP-completi

### Riduzioni e Teoremi Fondamentali (pag. 26-35)

- \*\*Riduzione polinomiale\*\*: Trasformazione di un problema in un altro

- \*\*Teorema di Cook-Levin (pag. 42)\*\*:

- SAT è NP-completo

- Base per dimostrare la NP-completezza di altri problemi

### Esempi di Problemi NP-Completi (pag. 36-50)

1. \*\*Problema della Clique\*\*:

- Input: Grafo G e intero k

- Domanda: Esiste un sottografo completo di k nodi?

2. \*\*Vertex Cover\*\*:

- Input: Grafo G e intero k

- Domanda: Esiste un insieme di k vertici che tocchi tutti gli archi?

3. \*\*3-SAT\*\*:

- Variante del problema SAT con clausole di 3 letterali

## Collegamento tra le Due Parti

Gli algoritmi genetici sono particolarmente utili per:

- Affrontare problemi NP-completi quando le soluzioni esatte sono impraticabili

- Fornire soluzioni approssimate in tempi ragionevoli

- Gestire spazi di ricerca complessi e multidimensionali

\*\*Esempio applicativo\*\*:

- Utilizzo di GA per risolvere istanze del TSP:

- Rappresentazione: Permutazione di città

- Funzione di fitness: Lunghezza del tour

- Operatori specializzati per mantenere soluzioni ammissibili

## Conclusioni

- \*\*Algoritmi genetici\*\*: Strumento potente per problemi complessi, specialmente quando combinati con tecniche specifiche per vincoli

- \*\*NP-completezza\*\*: Quadro teorico per comprendere i limiti computazionali

- \*\*Approccio pratico\*\*: Per problemi NP-completi, spesso è preferibile sviluppare:

- Algoritmi approssimati

- Euristiche

- Meta-euristiche (come gli algoritmi genetici)

\*\*Introduzione ai Metodi di Risoluzione dei Problemi\*\*

I metodi per risolvere problemi di ottimizzazione si dividono principalmente in \*\*metodi esatti\*\* (come Branch and Bound, Programmazione Dinamica) e \*\*metodi approssimati\*\*. I metodi approssimati includono gli \*\*algoritmi approssimati\*\* (con garanzie teoriche sul rapporto qualità/soluzione ottima) e gli \*\*algoritmi euristici\*\*.

Le \*\*euristiche\*\* sono ulteriormente suddivise in:

\* \*\*Euristiche specifiche per il problema:\*\* Progettate appositamente per un problema particolare.

\* \*\*Metaeuristiche:\*\* Metodi generali applicabili a una vasta gamma di problemi.

Le metaeuristiche si dividono in:

\* \*\*Basate su una singola soluzione (S-metaeuristiche):\*\* Come Ricerca Locale, Simulated Annealing, Tabu Search.

\* \*\*Basate su popolazioni di soluzioni:\*\* Come Algoritmi Genetici, PSO, ACO.

\*\*Concetti Chiave Comuni:\*\*

\* \*\*Codifica del problema:\*\* Come rappresentiamo una soluzione?

\* \*\*Rappresentazione della soluzione:\*\* La struttura dati usata per descrivere una soluzione.

\* \*\*Progettazione del metodo di ricerca:\*\* Come esploriamo lo spazio delle soluzioni?

\* \*\*Definizione della funzione obiettivo:\*\* Cosa vogliamo minimizzare/massimizzare?

\*\*Algoritmi Greedy (Ingordi)\*\*

Gli algoritmi greedy costruiscono una soluzione passo dopo passo, partendo da una soluzione vuota. Ad ogni passo, selezionano un elemento (secondo una regola euristica locale) da aggiungere alla soluzione parziale, senza mai tornare indietro (no backtracking). Una volta scelto, un elemento non viene mai sostituito.

\* \*\*Template:\*\* Inizializza soluzione `s` come vuota. Ripeti: scegli un elemento `e` tramite euristica locale tra quelli non ancora selezionati (`E \ s`). Se `s U {e}` è una soluzione ammissibile (`∈ F`), aggiungilo a `s`. Finché `s` non è completa.

\* \*\*Vantaggi:\*\* Semplici da progettare e generalmente a bassa complessità computazionale.

\* \*\*Limiti:\*\* La scelta locale ottima non garantisce l'ottimo globale. Possono essere "miope". Esempio classico: il TSP con l'euristica del vicino più prossimo può produrre soluzioni molto lontane dall'ottimo.

\*\*Metodi Iterativi (S-Metaeuristiche)\*\*

Questi metodi partono da una soluzione iniziale e iterativamente la migliorano. Ad ogni iterazione, generano soluzioni candidate (spesso "vicine" alla soluzione corrente) e selezionano una di esse per diventare la nuova soluzione corrente. Il processo continua fino al raggiungimento di un criterio di arresto.

\* \*\*Template:\*\* Parti da una soluzione iniziale `s0`. Ripeti: Genera un insieme di soluzioni candidate `C(s)` (vicinate a `s`). Seleziona una soluzione `s'` da `C(s)` per diventare la nuova soluzione corrente. Aggiorna il contatore. Finché non si verifica il criterio di arresto. Restituisci la migliore soluzione trovata.

\* \*\*Componenti comuni:\*\* Definizione della \*\*struttura di vicinato (neighborhood)\*\* e scelta della \*\*soluzione iniziale\*\*.

\*\*Definizione del Vicinato (Neighborhood)\*\*

Il vicinato di una soluzione `s`, indicato con `N(s)`, è l'insieme delle soluzioni "vicine" a `s`. La definizione del vicinato è cruciale per le performance di un algoritmo iterativo. Dipende fortemente dalla \*\*rappresentazione\*\* della soluzione.

\* \*\*Importanza:\*\* Un vicinato inadeguato può portare al fallimento dell'algoritmo. La "località" indica che piccole modifiche alla rappresentazione dovrebbero produrre piccole modifiche alla soluzione.

\* \*\*Esempi comuni:\*\*

\* \*\*Rappresentazione binaria:\*\* Flipping di un singolo bit (distanza di Hamming 1). Dimensione del vicinato: `n` (dove `n` è la lunghezza della stringa).

\* \*\*Rappresentazione a permutazione:\*\* Scambio (swap) di due elementi. Dimensione del vicinato: `n(n-1)/2`.

\* \*\*Operatori specifici:\*\*

\* \*\*Swap:\*\* Scambia due elementi.

\* \*\*Insertion/Move:\*\* Rimuovi un elemento e inseriscilo in un'altra posizione.

\* \*\*Inversione:\*\* Inverti una sotto-sequenza.

\* \*\*k-opt (es. 2-opt, 3-opt):\*\* Rimuovi `k` archi e ricollegali in modo diverso (usato spesso nel TSP).

\* \*\*Nota:\*\* L'efficienza di un vicinato dipende anche dal problema. Ad esempio, gli operatori k-opt non sono ideali per problemi di scheduling dove l'ordine è importante, mentre sono efficaci per il TSP dove l'adiacenza conta.

\*\*Soluzioni Locali vs Globali\*\*

Un \*\*ottimo locale\*\* è una soluzione che non può essere migliorata considerando solo i suoi vicini immediati (`N(s)`). Un \*\*ottimo globale\*\* è la migliore soluzione in tutto lo spazio delle soluzioni. La qualità dell'ottimo locale trovato da un metodo iterativo dipende fortemente dalla \*\*soluzione iniziale\*\* e dalla \*\*struttura del vicinato\*\*. Un ottimo locale per un vicinato `N1` potrebbe non esserlo per un vicinato `N2`.

\*\*Vicinati Molto Grandi (Very Large Neighborhoods - VLN)\*\*

Quando si progetta una S-metaeuristica, c'è un compromesso tra la \*\*dimensione del vicinato\*\* e la \*\*qualità delle soluzioni\*\* ottenibili, e la \*\*complessità computazionale\*\* per esplorarlo.

\* \*\*Vantaggio dei VLN:\*\* Esplorare un vicinato più grande può portare a soluzioni di migliore qualità, poiché si considerano più candidati ad ogni iterazione.

\* \*\*Svantaggio:\*\* Richiede tempo computazionale aggiuntivo per generare e valutare molti vicini. La dimensione del vicinato può crescere in modo polinomiale di alto grado o addirittura esponenzialmente con la dimensione del problema, aumentando drasticamente la complessità della ricerca.

\* \*\*Sfida:\*\* Progettare \*\*procedure efficienti\*\* per esplorare i VLN, identificando vicini miglioranti o il migliore vicino \*\*senza enumerare l'intero vicinato\*\*.

---

\*\*Heuristics & Metaheuristics for Optimization & Learning - Lezione del 28/04/2025\*\*

\*\*Ricerca Locale (Local Search - LS)\*\*

La Ricerca Locale è probabilmente la metaeuristica più antica e semplice. Parte da una soluzione iniziale e, ad ogni iterazione, cerca di migliorarla sostituendola con un \*\*vicino migliore\*\*.

\* \*\*Template:\*\* Genera una soluzione iniziale `s0`. Finchè non si raggiunge un criterio di arresto: Genera i vicini `N(s)`. Se nessun vicino è migliore di `s`, ferma la ricerca. Altrimenti, seleziona un vicino migliore `s'` e poni `s = s'`.

\* \*\*Interpretazione:\*\* Può essere vista come una \*\*discesa\*\* nello spazio delle soluzioni rappresentato come un grafo, fermandosi quando si raggiunge un "fondo di valle" (ottimo locale).

\* \*\*Strategie di Selezione del Vicino:\*\*

\* \*\*Miglioramento Migliore (Best Improvement/Steepest Descent):\*\* Esamina tutto il vicinato e sceglie il vicino che migliora di più la funzione obiettivo. Esplorazione esaustiva, ma costosa per vicinati grandi.

\* \*\*Primo Miglioramento (First Improvement):\*\* Esamina il vicinato in un ordine prefissato e sceglie il \*\*primo\*\* vicino che migliora la soluzione. Esplorazione parziale, generalmente più veloce.

\* \*\*Selezione Casuale:\*\* Sceglie casualmente un vicino tra quelli che migliorano.

\* \*\*Compromesso:\*\* La scelta della strategia influenza il tempo di ricerca e la qualità della soluzione finale. Ad esempio, usare "Primo Miglioramento" con una soluzione iniziale casuale può bilanciare tempo e qualità.

\*\*Tabu Search (TS)\*\*

La Tabu Search è una metaeuristica deterministica che estende la Ricerca Locale per sfuggire agli ottimi locali. Il suo tratto distintivo è l'uso della \*\*memoria\*\*.

\* \*\*Ispirazione:\*\* Simile alla Ricerca Locale "a discesa più ripida", ma accetta anche soluzioni \*\*peggiorative\*\* quando tutti i vicini sono peggiorativi, per uscire dai minimi locali.

\* \*\*Esplorazione del Vicinato:\*\* Solitamente esplora l'intero vicinato in modo deterministico ad ogni iterazione.

\* \*\*Scelta della Soluzione:\*\* Quando non si è in un ottimo locale, sceglie il vicino migliore (come LS). Quando si è in un ottimo locale (tutti i vicini sono peggiori), sceglie comunque il \*\*migliore vicino\*\* tra tutti, anche se è peggiore della soluzione corrente.

\* \*\*Problema del Ciclismo:\*\* Questa strategia può portare a cicli, ritornando su soluzioni già visitate.

\* \*\*Soluzione: Lista Tabù (Tabu List):\*\* Una struttura dati (memoria a breve termine) che memorizza le \*\*mosse\*\* o \*\*attributi di mosse\*\* recentemente applicate. Le soluzioni/mosse che riportano a stati recenti sono dichiarate "tabù" e (generalmente) rifiutate.

\* \*\*Efficienza:\*\* Memorizzare \*tutte\* le soluzioni visitate sarebbe troppo costoso. Si memorizzano solo gli attributi delle ultime `k` mosse (dove `k` è una costante).

\* \*\*Limitazione:\*\* Memorizzare solo attributi può portare a \*\*scarti troppo restrittivi\*\* (rifiutare soluzioni non ancora generate).

\* \*\*Criteri di Aspirazione (Aspiration Criteria):\*\* Eccezioni alla regola tabù. Ad esempio, una mossa tabù può essere accettata se genera una soluzione migliore di quella migliore trovata finora.

\* \*\*Memoria a Lungo Termine (Diversificazione):\*\* Memorizza informazioni sulle soluzioni visitate durante la ricerca per esplorare aree non ancora visitate dello spazio. Ad esempio, scoraggia gli attributi delle soluzioni di alta qualità per spostare la ricerca altrove.

\* \*\*Template TS:\*\* Inizializza soluzione `s`, lista tabù, memorie a medio/lungo termine. Ripeti: Trova il miglior vicino \*\*ammesso\*\* (non tabù o che soddisfa un criterio di aspirazione). Poni `s = s'`. Aggiorna lista tabù, criteri di aspirazione, memorie. Se necessario, applica Intensificazione/Diversificazione. Finché non si verifica il criterio di arresto. Restituisci la migliore soluzione trovata.

\* \*\*Intensificazione:\*\* Concentra la ricerca su aree promettenti dello spazio, ad esempio basandosi sulle informazioni della memoria a lungo termine.

\* \*\*Diversificazione:\*\* Spinge la ricerca verso aree inesplorate, ad esempio scoraggiando attributi frequenti.

\* \*\*Punti Chiave nella Progettazione:\*\*

\* Definizione del vicinato e della soluzione iniziale (comune a tutte le S-metaeuristiche).

\* Gestione della \*\*lista tabù\*\*.

\* Definizione dei \*\*criteri di aspirazione\*\*.

\* Gestione della \*\*memoria a lungo termine\*\* (per diversificazione).

---

\*\*Heuristics & Metaheuristics for Optimization & Learning - Lezione del 30/04/2025\*\*

\*\*Iterated Local Search (ILS)\*\*

L'Iterated Local Search è un framework che combina la potenza della Ricerca Locale con una strategia di perturbazione per sfuggire agli ottimi locali.

\* \*\*Principio:\*\* 1) Applica una Ricerca Locale a una soluzione iniziale. 2) Iterativamente: Perturba la soluzione ottima locale trovata. 3) Applica nuovamente la Ricerca Locale alla soluzione perturbata. 4) Accetta la nuova soluzione ottima locale come soluzione corrente secondo un certo criterio. 5) Ripeti fino al criterio di arresto.

\* \*\*Template ILS:\*\* Applica Ricerca Locale a `s0` ottenendo `s\*`. Ripeti: Perturba `s\*` (possibilmente usando la storia della ricerca) ottenendo `s'`. Applica Ricerca Locale a `s'` ottenendo `s'\*`. Accetta `s'\*` come nuova `s\*` (eventualmente usando la memoria di ricerca). Finché non si verifica il criterio di arresto. Restituisci la migliore soluzione trovata.

\* \*\*Elementi Chiave:\*\*

\* \*\*Ricerca Locale:\*\* Può essere qualsiasi S-metaeuristica (LS, TS, SA). È trattata come una "scatola nera".

\* \*\*Metodo di Perturbazione:\*\* Come modificare la soluzione ottima locale corrente? La grandezza della perturbazione è cruciale:

\* Troppo \*\*piccola\*\*: rischio di cicli, bassa probabilità di esplorare nuovi bacini di attrazione.

\* Troppo \*\*grande\*\*: si perde l'informazione acquisita, si saltano le buone proprietà della soluzione locale.

\* \*\*Criterio di Accettazione:\*\* Quando accettare la nuova soluzione ottima locale `s'\*` come soluzione corrente `s\*`? Spesso si accetta sempre, oppure si usa una strategia come quella del Simulated Annealing (accettare peggioramenti con una certa probabilità).

\* \*\*Nota:\*\* La progettazione di un ILS efficace dipende principalmente dal \*\*metodo di perturbazione\*\* scelto e dal \*\*criterio di accettazione\*\*.

# Algoritmi Genetici: Ispirazione dalla Natura per Risolvere Problemi Complessi

## Introduzione

Gli algoritmi genetici (GA) sono metodi euristici ispirati ai principi dell'evoluzione naturale, come la selezione, il crossover e la mutazione. Sono particolarmente utili per problemi complessi dove soluzioni tradizionali falliscono a causa di:

- \*\*Tempo\*\*: Problemi che richiedono troppo tempo per essere risolti con metodi deterministici.

- \*\*Incertezza\*\*: Ambienti con dati incompleti o variabili.

- \*\*Big Data\*\*: Situazioni con enormi quantità di dati da analizzare.

---

## Come Funzionano?

I GA simulano l'evoluzione di una popolazione di soluzioni candidate, migliorandole iterativamente. Ecco i passaggi principali:

1. \*\*Inizializzazione\*\*: Si genera una popolazione casuale di soluzioni, rappresentate come stringhe (es. binario, numeri reali, permutazioni).

2. \*\*Valutazione\*\*: Ogni soluzione viene valutata tramite una \*funzione di fitness\* che misura la sua qualità.

3. \*\*Selezione\*\*: Le soluzioni migliori hanno maggiori probabilità di essere selezionate come "genitori" per la generazione successiva.

4. \*\*Crossover e Mutazione\*\*:

- \*\*Crossover\*\*: Combina parti di due genitori per creare nuove soluzioni (es. single-point, uniforme).

- \*\*Mutazione\*\*: Introduce piccole variazioni casuali per esplorare nuove aree dello spazio di ricerca.

5. \*\*Sostituzione\*\*: Le nuove soluzioni sostituiscono quelle meno performanti.

6. \*\*Terminazione\*\*: Il processo si ripete fino a raggiungere un criterio di arresto (es. numero massimo di generazioni o fitness soddisfacente).

---

## Operatori Chiave

- \*\*Selezione\*\*: Basata su fitness, privilegia le soluzioni migliori (es. roulette-wheel, tornei).

- \*\*Crossover\*\*: Esempi:

- \*\*Single-point\*\*: Taglio in un punto e scambio delle parti.

- \*\*Uniforme\*\*: Scambio casuale di geni tra genitori.

- \*\*Mutazione\*\*: Piccole modifiche casuali (es. scambio, inversione, inserimento).

## Applicazioni

- \*\*Problemi di ottimizzazione\*\*: Funzioni matematiche, scheduling.

- \*\*Sudoku\*\*: Codificando la griglia come permutazione e minimizzando i vincoli violati.

- \*\*Machine Learning\*\*: Ottimizzazione di iperparametri.

## Vantaggi

- \*\*Robustezza\*\*: Gestisce incertezza e rumore.

- \*\*Flessibilità\*\*: Adattabile a diversi problemi.

- \*\*Parallelizzabile\*\*: Può esplorare più soluzioni contemporaneamente.

### Conclusione

I GA sono strumenti potenti per problemi complessi, combinando esplorazione (crossover) e sfruttamento (mutazione) per trovare soluzioni ottimali in modo efficiente. La chiave è una buona rappresentazione del problema e una funzione di fitness ben definita.

\*\*Come funziona un Algoritmo Genetico (GA)?\*\*

Un Algoritmo Genetico è una \*\*metaeuristica basata su popolazioni\*\*. Invece di lavorare su una singola soluzione che viene iterativamente migliorata (come nelle S-metaeuristiche viste finora), un GA lavora simultaneamente su un insieme (popolazione) di soluzioni candidate. Queste soluzioni evolvono nel tempo attraverso meccanismi ispirati all'evoluzione naturale: le soluzioni più "adatte" (quelle con un valore migliore della funzione obiettivo) hanno maggiori probabilità di "riprodursi" e generare "figli" (nuove soluzioni), mentre le soluzioni meno adatte tendono a scomparire.

Il processo è iterativo:

1. Si parte con una \*\*popolazione iniziale\*\* di soluzioni (spesso generate casualmente).

2. Si \*\*valutano\*\* tutte le soluzioni nella popolazione usando la funzione obiettivo.

3. Si \*\*selezionano\*\* alcune soluzioni dalla popolazione corrente per fungere da "genitori", tipicamente basandosi sulla loro bontà (fitness).

4. Si applicano \*\*operatori genetici\*\* (come l'incrocio e la mutazione) ai genitori selezionati per creare nuove soluzioni (figli).

5. Le nuove soluzioni formano (in tutto o in parte) la \*\*nuova generazione\*\* (popolazione).

6. Il processo si ripete fino al raggiungimento di un criterio di arresto (es: numero massimo di generazioni, qualità della soluzione, tempo).

\*\*Componenti Principali di un Algoritmo Genetico:\*\*

1. \*\*Codifica (Representation):\*\* Come viene rappresentata una soluzione (individuo) all'interno dell'algoritmo (es: stringa binaria, permutazione, vettore reale)? Questa scelta influenza fortemente il design degli operatori.

2. \*\*Popolazione Iniziale:\*\* Il punto di partenza dell'algoritmo, solitamente generato casualmente.

3. \*\*Funzione di Valutazione (Fitness Function):\*\* Corrisponde alla funzione obiettivo del problema. Serve a quantificare la "bontà" di ogni singola soluzione nella popolazione.

4. \*\*Selezione (Selection):\*\* Il meccanismo che determina quali individui della popolazione corrente avranno l'opportunità di riprodursi. Favorisce le soluzioni con fitness più alta (es: selezione per torneo, roulette wheel selection).

5. \*\*Operatori Genetici:\*\*

\* \*\*Incrocio (Crossover):\*\* Combina parti di due (o più) soluzioni genitore per creare uno o più figli. È l'operatore principale di \*\*ricombinazione\*\* dell'informazione.

\* \*\*Mutazione (Mutation):\*\* Introduce piccole modifiche casuali in una singola soluzione. Serve a mantenere la \*\*diversità\*\* nella popolazione e a sfuggire ai minimi locali.

6. \*\*Criterio di Arresto (Stopping Criterion):\*\* Determina quando terminare l'esecuzione dell'algoritmo (es: numero massimo di iterazioni/generazioni, stabilità della soluzione migliore).

D'accordo, ecco una spiegazione generale degli \*\*Algoritmi Immunologici (Artificial Immune Systems - AIS)\*\* basata su conoscenze comuni nel campo dell'ottimizzazione e dell'intelligenza artificiale, anche se l'argomento \*\*non è trattato nei file delle lezioni fornite\*\*.

\*\*Algoritmi Immunologici (AIS)\*\*

Gli Algoritmi Immunologici sono una \*\*classe di metaeuristiche\*\* ispirate al \*\*sistema immunitario biologico\*\* degli organismi viventi. L'obiettivo è sfruttare i principi e i meccanismi di questo sistema naturale per risolvere problemi complessi di ottimizzazione e apprendimento.

\*\*Principio di Funzionamento:\*\*

Il sistema immunitario biologico protegge l'organismo da agenti patogeni (virus, batteri) riconoscendoli come "non-self" rispetto alle cellule "self" dell'organismo stesso. Produce anticorpi specifici per neutralizzare ogni antigene (patogeno) specifico. Caratteristiche chiave del sistema immunitario sono:

\* \*\*Selezione Clonale:\*\* Quando un antigene è riconosciuto, i linfociti (cellule immunitarie) specifici per quell'antigene si replicano (clonano) rapidamente.

\* \*\*Mutazione Somatica:\*\* Durante la replicazione, i linfociti subiscono mutazioni casuali. Queste mutazioni permettono di generare varianti degli anticorpi.

\* \*\*Selezione Affinità:\*\* Tra le varianti mutate, quelle con \*\*maggiore affinità\*\* (capacità di legarsi meglio) all'antigene sopravvivono e si proliferano, mentre quelle con minore affinità vengono eliminate.

\* \*\*Memoria Immunologica:\*\* Il sistema "ricorda" gli antigeni incontrati in passato, permettendo una risposta più rapida e efficace se lo stesso antigene attacca nuovamente.

Gli AIS traslano questi concetti in un contesto computazionale:

1. \*\*Antigeni:\*\* Rappresentano il problema da risolvere o un obiettivo da raggiungere.

2. \*\*Anticorpi / Linfociti:\*\* Rappresentano le \*\*soluzioni candidate\*\* al problema.

3. \*\*Affinità:\*\* È legata alla \*\*qualità\*\* della soluzione, solitamente misurata dal valore della funzione obiettivo. Un'alta affinità indica una soluzione di alta qualità.

4. \*\*Selezione Clonale:\*\* Le soluzioni migliori (anticorpi con alta affinità) vengono selezionate e replicate.

5. \*\*Mutazione Somatica:\*\* Alle soluzioni clonate vengono applicate piccole modifiche (mutazioni) per generare varianti.

6. \*\*Selezione dell'Affinità:\*\* Tra i cloni mutati, si selezionano quelli con affinità più alta (soluzioni migliorate).

7. \*\*Memoria:\*\* L'algoritmo può mantenere traccia delle buone soluzioni trovate per guidare la ricerca futura o per accelerare la convergenza verso soluzioni ottime note.

\*\*Schema Generale di un AIS:\*\*

1. \*\*Inizializzazione:\*\* Genera una popolazione iniziale casuale di soluzioni (anticorpi).

2. \*\*Valutazione:\*\* Calcola l'affinità (fitness) di ogni soluzione rispetto al problema (antigene).

3. \*\*Selezione Clonale:\*\* Seleziona le soluzioni con affinità più alta e crea cloni di queste soluzioni. Il numero di cloni può essere proporzionale all'affinità.

4. \*\*Mutazione:\*\* Applica operatori di mutazione ai cloni. La frequenza o l'intensità della mutazione può essere inversamente correlata all'affinità del clone originale (ipermutazione somatica).

5. \*\*Selezione dell'Affinità (Maturation):\*\* Valuta i cloni mutati e seleziona i migliori (quelli con affinità più alta) per formare la nuova popolazione o parte di essa.

6. \*\*Aggiornamento della Popolazione:\*\* La nuova popolazione può essere composta dai migliori cloni maturati, integrata con nuove soluzioni casuali per mantenere la diversità (simile alla risposta immune innata).

7. \*\*Memoria (opzionale):\*\* Archivia le soluzioni di alta qualità trovate durante il processo.

8. \*\*Iterazione:\*\* Ripete i passi da 2 a 7 fino al raggiungimento di un criterio di arresto (es: numero massimo di iterazioni, qualità della soluzione).

\*\*Applicazioni e Vantaggi:\*\*

\* \*\*Applicazioni:\*\* Problemi di ottimizzazione combinatoria, ottimizzazione continua, riconoscimento di pattern, rilevamento di anomalie, apprendimento automatico.

\* \*\*Vantaggi:\*\* Capacità intrinseca di apprendimento e adattamento grazie alla memoria, buona capacità di ottimizzazione locale grazie alla maturazione dei cloni, mantenimento della diversità nella popolazione.

In sintesi, gli Algoritmi Immunologici sono metaeuristiche che simulano i meccanismi del sistema immunitario biologico (selezione clonale, mutazione, affinità) per evolvere una popolazione di soluzioni verso l'ottimo di un problema.

D'accordo, ecco una descrizione generale dell'\*\*Ottimizzazione mediante Colonia di Formiche (Ant Colony Optimization - ACO)\*\* basata su conoscenze comuni nel campo dell'ottimizzazione, al di fuori del materiale specifico delle lezioni fornite.

\*\*Ottimizzazione mediante Colonia di Formiche (Ant Colony Optimization - ACO)\*\*

L'Ant Colony Optimization è una \*\*metaeuristica basata su popolazioni\*\* ispirata al \*\*comportamento foraggiante distribuito\*\* delle vere colonie di formiche. È particolarmente efficace per risolvere problemi di ottimizzazione su \*\*grafo\*\*, come il problema del commesso viaggiatore (TSP) o problemi di instradamento su reti.

\*\*Principio di Funzionamento:\*\*

Le formiche reali comunicano indirettamente tra loro attraverso \*\*tracce di feromone\*\* che depositano nel terreno mentre si muovono. Quando devono scegliere una strada (ad esempio, verso una fonte di cibo), tendono a seguire il percorso con una \*\*concentrazione più alta di feromone\*\*. A sua volta, una maggiore concentrazione di feromone indica che più formiche hanno seguito quel percorso, suggerendo (ma non garantendo) che sia un buon percorso. Questo meccanismo crea un \*\*feedback positivo\*\*: i percorsi più corti vengono percorsi più rapidamente, quindi il feromone evapora meno e ne viene depositato di più, rendendoli ancora più attraenti.

Gli algoritmi ACO simulano questo processo per risolvere problemi di ottimizzazione:

1. \*\*Formiche Artificiali:\*\* L'algoritmo gestisce una popolazione di \*\*agenti\*\* (formiche artificiali) che costruiscono \*\*soluzioni\*\* al problema (ad esempio, un tour completo nel TSP).

2. \*\*Feromone Artificiale:\*\* Viene associata una \*\*traccia di feromone\*\* artificiale a ciascun componente della soluzione (ad esempio, ad ogni arco/connessione del grafo del problema). Questa traccia influenza le scelte future delle formiche.

3. \*\*Costruzione della Soluzione:\*\* Ogni formica costruisce una soluzione passo dopo passo. La scelta del prossimo componente da aggiungere alla soluzione parziale è \*\*probabilistica\*\* e \*\*basata su due fattori\*\*:

\* \*\*Feromone (τ):\*\* La quantità di traccia di feromone presente sul componente. Una quantità maggiore rende il componente più probabile da scegliere.

\* \*\*Euristica (η):\*\* Una misura della \*\*desiderabilità a priori\*\* del componente (ad esempio, l'inverso della distanza in un problema di routing). Un valore più alto indica un componente intrinsecamente migliore.

4. \*\*Aggiornamento del Feromone:\*\* Dopo che tutte (o una parte) delle formiche hanno costruito una soluzione completa, la traccia di feromone viene aggiornata:

\* \*\*Evaporazione:\*\* Una frazione del feromone su tutti i componenti \*\*evapora\*\* (viene sottratta). Questo è un meccanismo fondamentale per \*\*evitare la convergenza prematura\*\* verso soluzioni sub-ottime e per \*\*dimenticare informazioni obsolete\*\*.

\* \*\*Deposito:\*\* Le formiche \*\*depositano\*\* feromone sui componenti delle soluzioni che hanno costruito. La quantità depositata è solitamente \*\*proporzionale alla qualità\*\* della soluzione trovata (ad esempio, inversamente proporzionale al costo del tour nel TSP: soluzioni migliori depositano più feromone).

\*\*Schema Generale di ACO:\*\*

1. \*\*Inizializzazione:\*\* Inizializza le tracce di feromone su tutti i componenti (spesso con un valore piccolo e costante).

2. \*\*Ciclo Principale:\*\* Ripeti fino al criterio di arresto:

\* \*\*Costruzione delle Soluzioni:\*\* Per ogni formica: Costruisci una soluzione completa scegliendo i componenti uno alla volta in base alla \*\*regola di transizione\*\* (una combinazione della traccia di feromone e dell'informazione euristica).

\* \*\*Aggiornamento del Feromone:\*\* Aggiorna le tracce di feromone su tutti i componenti:

\* \*\*Evapora\*\* una parte del feromone esistente.

\* \*\*Deposita\*\* nuovo feromone basato sulle soluzioni costruite dalle formiche (spesso solo le migliori, o tutte).

3. \*\*Output:\*\* Restituisci la migliore soluzione trovata durante il processo.

\*\*Componenti Chiave di ACO:\*\*

\* \*\*Regola di Transizione (Transition Rule):\*\* Determina come una formica sceglie il prossimo componente della soluzione. Combina la traccia di feromone (`τ`) e l'euristica (`η`).

\* \*\*Aggiornamento del Feromone (Pheromone Update):\*\* Definisce come e quando le tracce di feromone vengono modificate (evaporazione + deposito).

\* \*\*Evaporazione:\*\* Parametro che controlla la velocità di evaporazione del feromone.

\* \*\*Numero di Formiche:\*\* Il numero di agenti che costruiscono soluzioni in parallelo ad ogni iterazione.

\*\*Applicazioni e Vantaggi:\*\*

\* \*\*Applicazioni:\*\* Problemi su grafi come TSP, Vehicle Routing Problem (VRP), assegnamento quadratico, scheduling, network routing.

\* \*\*Vantaggi:\*\* Capacità intrinseca di apprendere e adattarsi grazie al feedback del feromone, buona per esplorare lo spazio delle soluzioni complesso, naturale per problemi discreti e combinatori su grafi.

In sintesi, ACO è una metaeuristica che simula il comportamento collettivo delle formiche per risolvere problemi di ottimizzazione. Le formiche artificiali costruiscono soluzioni guidate da tracce di feromone che vengono aggiornate in base alla qualità delle soluzioni trovate, permettendo all'algoritmo di convergere verso soluzioni di alta qualità.