Algoritmi evolutivi

Francesco Tomaselli

22 gennaio 2021

Indice

1 Introduzione

A seguire un'introduzione alle metaeuristiche, per poi passare all'idea e alle componenti che stanno alla base degli algoritmi evolutivi.

1.1 Metaeuristiche

Una metaeuristica è una tecnica che permette di risolvere problemi di ottimizzazione in maniera approssimata, per iterazioni.

Le metaeuristiche sono spesso applicate a problemi non risolvibili in modo esatto in tempo utile. Con questo approccio si individua una soluzione ammissibile.

L'idea di base sta nell'individuare un insieme di soluzioni possibili e cercare di migliorarle ad ogni iterazione. Si usa una sorta di *ricerca casuale guidata* che prevede l'esplorazione dello spazio di soluzioni con un certo grado di casualità, ma comunque guidata da una certa metrica di qualità.

Si fa notare che le metaeuristiche possono essere interrotte in qualsiasi momento, non serve quindi necessariamente individuare una soluzione ottima. È possibile infatti prevedere un numero massimo di iterazioni dell'algoritmo, sperando che siano state sufficienti a produrre una soluzione "buona".

1.2 Evoluzione simulata

Prendendo spunto dall'evoluzione, si possono applicare concetti naturali alla ricerca di una soluzione per un certo problema.

Definizione 1 *Un problema di ottimizzazione è definito dalla coppia* (Ω, f)*, dove il primo indica lo spazio di soluzioni, e il secondo una funzione di valutazione. L'obbiettivo è quello di individuare il massimo globale in* Ω

Principi

L'evoluzione simulata si basa sui principi di quella naturale, in particolare tenta di generare molta diversità nella generazione iniziale, per favorire un'esplorazione omogenea dello spazio di ricerca.

Utilizza poi il concetto di selezione naturale e di manteninmento di tratti vantaggiosi, ovvero, come in natura, gli individui più forti, o adatti all'ambiente in cui si trovano, hanno piu probabilità di riprodursi e di sopravvivere di generazione in generazione.

Nozioni di base

Parlando di algoritmi evolutivi, bisogna introdurre alcune nozioni di base:

1. *Individuo*: soluzione candidata
2. *Cromosoma*: sequenza di oggetti
3. *Gene*: oggetto, es: bit, carattere, numero, etc.
4. *Allele*: valore di un oggetto
5. *Locus*: posizione di un oggetto
6. *Fenotipo*: applicazione di una soluzione candidata
7. *Genotipo*: codifica di una soluzione candidata
8. *Popolazione*: set di cromosomi
9. *Generazione*: popolazione in un certo istante di tempo
10. *Riproduzione*: creazione di cromosomi da uno o piu cromosomi
11. *Fitness*: qualità di una soluzione candidata

Elementi di un problema evolutivo

Un problema evolutivo prevede la definizione di vari elementi quali:

1. Encoding di una soluzione candidata
2. Generazione della popolazione iniziale
3. Funzione di Fitness
4. Metodo di selezione
5. Operatori genetici di modifica dei cromosomi
6. Criterio di terminazione
7. Valori per i relativi parametri

Schema generale

Generalmente un'algoritmo evolutivo opera seguendo questi passi:

1. Generazione di una popolazione iniziale
2. Valutazione del fitness degli individui
3. Generazione della nuova popolazione:
   (a) Selezione dei genitori basata sul fitness calcolato al punto precedente
   (b) Generazione di discendenti, partendo dai genitori individuati al punto precedente

(c) Mutazione randomica degli individui

(d) Selezione di $|pop|$ individui tra i padri e i figli

4. Ripetizione dal punto 2, fino al raggiungimento del criterio di terminazione

Ragionando sul processo utilizzato, è immediato capire che un algoritmo evolutivo non garantisce la qualità della soluzione trovata ma si spera che, partendo da una popolazione iniziale che copre molte aree dello spazio di ricerca, migliorando iterativamente gli individui si riesca a trovare una soluzione buona.

1.3 Ricerca locale

Come nell'ottimizzazione classica, si possono applicare varie tecniche per migliorare le soluzioni individuate.

Una particolare branca delle tecniche di ottimizzazione consiste nella così detta ricerca locale. Si cerca quindi di migliorare localmente la soluzione individuata, assumendo quindi che punti vicini nello spazio di soluzioni abbiano qualità simili.

Gradient Ascent e Descent

L'applicazione di gradient ascent o descent presuppone che:

1. Lo spazio di soluzioni Ω sia un sottoinsieme di $\mathbb{R}^n$
2. La funzione di valutazione f sia derivabile in ogni punto

Fissati i due prerequisiti, si può procedere con gradient acent (o descent):

1. Si sceglie un punto iniziale
2. Viene calcolato il gradiente del punto corrente
3. Si procede in direzione del gradiente o in senso opposto
4. I punti 2 e 3 si ripetono fino a quando si individua un minimo o massimo locale

La procedura ha il grande difetto di rimanere bloccata in un minimo locale. Bisogna quindi ripetere l'algoritmi cambiando punto di partenza.

Hill Climbing

Se la funzione f non è differenziabile, è possibile verificare l'andamento della funzione nelle prossimità del punto corrente e scegliere il vicino ottimo.

L'algoritmo si può definire in 4 punti:

1. Si parte da un punto casuale s_0 in Ω
2. Viene scelto un punto s nelle vicinanze di s_t
3. s_{t+1} diventa s se $f(s) > f(s_t)$, altrimenti rimane s_t
4. I punti 2 e 3 vengono ripetuti fino alla raggiunta di un criterio di terminazione

Hill Climbing ha tendenza a bloccarsi in un ottimo locale, come Gradient Ascent.

Simulated Annealing

L'idea fondamentale dell'approccio è che una soluzione migliore di quella attuale è sempre accettata, ma è possibile accettarne anche una peggiore con una certa probabilità. Tale mossa potrebbe evitare di rimanere bloccati in un massimo locale.

L'algoritmo è definito come segue:

1. Si parte da un punto casuale s_0 in Ω
2. Viene scelto un punto s nelle vicinanze di s_t
3. s_{t+1} diventa:
   - s se $f(s) > f(s_t)$
   - s con probabilità $p = e^{-\frac{\Delta f}{kT}}$, dove Δf indica la riduzione di qualità della soluzione e T è il parametro temperatura, che cala con il passare del tempo
   - s_t con probabilità $1 - p$
4. I punti 2 e 3 vengono ripetuti fino alla raggiunta di un criterio di terminazione

La temperatura T si riduce di iterazione in iterazione, per valori piccoli l'approccio equivale ad hill climbing, ma per valori grandi di T la probabilità di muoversi in una soluzione peggiore aumenta, riducendo il rischio di bloccarsi in un massimo locale.

Threshold Accepting

L'idea coincide con simulated annealing, ma le soluzioni peggiori sono accettate se la degradazione di qualità è inferiore ad una certa threshold, che si riduce nel tempo.

Great Deluge Algorithm

L'idea coincide con simulated annealing, ma le soluzioni peggiori sono accettate se la qualità è maggiore di un certo lower bound, incrementato nel tempo.

Record-to-Record Travel

L'approccio è molto simile al precedente, questa volta però per calcolare la degradazione di qualità si fa riferimento alla soluzione ottima fino ad ora.

1.4 Altre tecniche

Esistono altre tecniche di ottimizzazione legate al mondo degli evolutionary algorithms, che in alcuni casi sfruttano tecniche di ricerca locale, ma non necessariamente.

Tabu search

L'approccio consiste nel mantenere una coda FIFO nella quale memorizzare soluzioni già esplorate in passato. Alla creazione di un nuovo candidato si eviterà quindi di esplorare soluzioni già viste. Non sono considerate le mutazioni.

Algoritmi memetici

L'idea alla base dell'approccio è quella di combinare i vantaggi degli algoritmi basati su popolazioni e quelli di ricerca locale.

Sarà considerata quindi una popolazione iniziale di soluzioni candidate, e, per velocizzare l'altrimenti lento processo di ottimizzazione, ogni individuo è migliorato localmente fino a raggiungere un massimo.

Nonostante velocizzi il processo di ricerca, è comunque molto probabile rimanere bloccati in massimi locali, e non riuscire ad esplorare a sufficienza lo spazio di soluzioni.

Evoluzione differenziale

Si utilizza un DE-operator, visto come combinazione di mutazione e ricombinazione. L'opertore guida la ricerca, alla creazione di un nuovo individuo, esso può rimpiazzare il padre se la qualità è maggiore.

Scatter search

Si genera una popolazione iniziale e si cerca di effettuare una ricerca locale attorno ad essa, guidando l'esplorazione dell'algoritmo nella direzione dei migliori individui.

La bontà dell'algoritmo risiede nell'esplorazione dello spazio di soluzioni.

2 Elementi base

Nella sezione precedente sono stati introdotti brevemente i principali elementi di un algoritmo evolutivo, a seguire una descrizione più nel dettaglio.

2.1 Encoding

La scelta del tipo di codifica è cruciale per il giusto funzionamento di un algoritmo evolutivo. Nonostante non si sia una soluzione generale per tale scelta, un econding deve rispettare alcune caratteristiche:

1. Fenotipi simili devono essere rappresentati da genotipi simili
2. Soluzioni codificate in modo simile, devono avere fitness simile
3. Lo spazio di soluzioni Ω dovrebbe essere chiuso rispetto agli operatori genetici

Hamming Cliffs

La similarità di due genotipi è definita dalla facilità, espressa in numero di mutazioni, di passaggio da uno all'altro. Il problema delle Hamming Cliffs si incontra quando l'encoding usato non rispetta la prima caratteristica iuntrodotta a inizio sezione.

Non rispettando quel punto infatti, due soluzioni simili sarebbero codificate in maniera molto diversa, finendo per rendere difficile il passaggio da una all'altra, rendendo più difficoltosa l'esplorazione dello spazio di soluzioni. Infatti, per passare dalla prima alla seconda, pur essendo molto vicine nel dominio iniziale, sarebbero necessarie troppe mutazioni.

Epistasi

Il secondo principio da rispettare consiste nell'avere fitness simili per soluzioni codificate in modo simile. Con epistasi si indica l'interazione dei geni in un cromosoma.

Codifiche che presentano alta epistasi sono indesiderabili, poichè andrebbero a impedire un miglioramento graduale delle soluzioni.

Chiusura dello spazio di soluzioni

Fissato un certo spazio di soluzioni, bisogna fare in modo che gli operatori genetici non creino elementi al di fuori di tale spazio. Elementi fuori da Ω non sarebbero interpretabili e risulterebbero inutili.

Per ovviare al problema è possibile:

1. Scegliere un econding differente
2. Scegliere operatori genetici che non creino elementi al di fuori di Ω
3. Usare meccanismi di riparazione per gli individui generati in modo errato
4. Introdurre una penalità che riduca il fitness di tali individui

2.2 Fitness e selezione

Il principio base della selezione è che individui migliori hanno più probabilità di procreare.

Definizione 2 *Con selective pressure si definisce quanto gli individui migliori siano preferiti per produrre la generazione successiva*

La selective pressure dovrebbe idealmente essere bassa nelle prime generazioni, per assecondare l'esplorazione dello spazio di soluzioni, per poi diventare sempre piu alta per migliorare localmente le soluzioni individuate.

Roulette-Wheel Selection

Una delle tecniche base per implementare la selezione è la Roulette-wheel selection. Si definisce il fitness relativo di un individuo come

$$f_{rel}(s) = \frac{f_{abs}(s)}{\sum_{s\prime \in pop(t)} f_{abs}(s\prime)}$$

Tale valore rappresenta la probabilità con cui l'individuo può essere selezionato.

Dominance problem

Una selezione basata sul fitness come quella al punto precedente potrebbe portare a *crowding*, ovvero il sovraffollamento di una certa regione dello spazio Ω. Tale effetto non è desiderabile in quanto non favorisce l'esplorazione dello spazio di soluzioni.

Il problema si manifesta quando esistono individui con fitness troppo alto che dominano gli altri nel passaggio di generazione.

Vanishing selective pressure

In contrapposizione all'avere individui con fitness troppo elevato, avere valori di fitness troppo simili nella popolazione porta all'azzeramento della selective pressure.

Considerando ad esempio la formula per il calcolo del fitness relativo, con valori molto simili di fitness nella popolazione si otterrà $f_{rel}(s) \approx \frac{1}{\mu}$, dove μ è la

dimensione della popolazione. Un comportamento simile porta a non avere di fatto selective pressure.

Una soluzione al problema consiste nello scalare la funzione di fitness, in modo da evitare il comportamento appena descritto:

- $\sigma scaling$: scalind basato su media e varianza
- *linear dynamical scaling*: sottrae minimo della fitness function da tutti gli individui

Variance problem

Durante la fase di selezione è possibile computare il fitness relativo e selezionare probabilisticamente in base a quello. La natura probabilistica dell'approccio potrebbe però scartare individui molto validi.

Una prima soluzione consiste nel determinare il numero di discendenti in base al fitness, considerando media e deviazione standard $\mu_f(t)$ e $\sigma_f(t)$:

- se $f(s) < \mu_t(t) - \sigma_f(t)$: 0 figli
- se $\mu_t(t) - \sigma_f(t) < f(s) < \mu_t(t) + \sigma_f(t)$: 1 figlio
- se $f(s) > \mu_t(t) - \sigma_f(t)$: 2 figli

Un'altra soluzione consiste nel considerare una ruota di selezione, la cui circonferenza è suddivisa in tanti archi quanti gli individui, e con grandezza degli archi proporzionale al fitness relativo di ogni individuo. Vengono fissati poi dei marker e viene fatta girare la ruota. Verranno poi selezionati gli individui puntati dai marker.

Rank-Based selection

Un approccio simile alla roulette-wheel selection è rank selection.

Gli individui vengono inizialmente ordinati per fitness, successivamente viene assegnato loro un rank in base all'ordine ottenuto. La selezione viene poi effettuata probabilisticamente sul rank.

Tale approccio differisce da roulette-wheel selection per il fatto che non vengono considerati direttamente i fitness ma solo l'ordine nella popolazione, di conseguenza il valore assoluto del fitness viene ignorato, evitando che alcuni individui dominino gli altri,

Adattando in base al tempo le probabilità associate ai rank, si puo controllare la selective pressure, mettendo focus sull'esplorazione piuttosto che sul miglioramento della popolazione attuale.

Tournament selection

Un altro approccio per implementare la selezione consiste nello scegliere un certo numero di individui e farli competere, scegliendo come vincitore quello con fitness più alto, o decidendo probabilisticamente in base al fitness.

Facendo così anche gli individui meno forti potrebbero passare alla generazione successiva, nel caso in cui vengano messi a confronto con individui peggiori.

Elitism

Gli approcci probabilistici potrebbero scartare gli individui migliori della generazione corrente.

Per risolvere il problema si possono ammettere alla generazione successiva i primi k individui ordinati per fitness, evitando sempre un peggioramento del fitness massimo.

Sharing

La tecnica dello sharing è pensata per prevenire crowding di una certa zona dello spazio Ω.

Nello specifico, individui che cadono in una zona affollata riceveranno una penalità nel fitness, andando potenzialmente a ridurre l'affollamento.

2.3 Operatori genetici

Gli operatori genetici sono applicati a un sottoinsieme degli individue in una generazione per creare mutazioni e combinazioni di soluzioni esistenti. Nonostante alcune modifiche possano risultare inconvenienti, la speranza è che alcune possano ottenere individui migliori.

Lo spazio di soluzioni Ω può essere o non essere chiuso rispetto a un operatore genetico.

Mutazione

Per mutazione si intende un piccolo cambiamento nel genotipo di un individuo. Si possono applicare vari tipi di mutazione, ad esempio:

- *Standard*: cambiamento del valore di un gene
- *Pair* swap: interscambio dei valori di due geni
- *Shift*: shift di una sottosequenza di geni

- *Permutazione arbitraria*: permutazione di una certa sottosequenza di geni
- *Inversione*: inversione di una sottosequenza di geni
- *Binaria*: inversione di alcuni geni con una certa probabilità
- *Gaussiana*: aggiunta ad ogni gene di un valore pescato da una distribuzione Gaussiana

Crossover

Nella tecnica del crossover, si parte da due o piu genitori, e si ricombinano spezzando le loro sequenze e ricombinando.

Esistono varianti quali:

- *One-point*: individuato un punto di spezzamento, si ricombinano i geni dei due genitori
- *N-point*: si individuano n punti di spezzamento invece che uno
- *Uniform*: si definisce la probabilità di scambio per ogni coppia
- *Shuffle*: viene effettuata una permutazione dei geni, si applica one point crossover, e si riporta all'ordine iniziale
- *Diagonal*: simile a N-point, ma con un numero più elevato di genitori

Edge recombination

Il cromosoma è rappresentato come un grafo. Ogni gene è un vertice che ha archi verso i suoi vicini. Gli archi dei due grafi vengono mischiati. Si preserva l'informazione relativa alla vicinanza.

Problemi degli operatori genetici

Esistono sostanzialmente due problemi negli operatori genetici:

1. *Positional bias*: si manifesta quando la probabilità che due geni siano ereditati dallo stesso padre varia in base alla loro posizione relativa. Ad esempio in one-point crossover, se due geni sono molto vicini, molto probabilmente saranno ereditati assieme.
2. *Distributional bias*: si manifesta se la probabilità che un certo numero di geni di un genitore passino al figlio è differente al variare del numero di geni considerato. Ad esempio, in uniform crossover, la probabilità che un certo numero di geni di uno dei due genitori passi al figlio è data da una binomiale.

3 Algoritmi fondamentali

Introdotti i componenti fondamentali, si discutono ora alcuni tipi e varianti di algoritmi evolutivi, con alcuni fondamenti teorici legati alla bontà dell'evoluzione simulata.

3.1 Fondamenti teorici

Sebbene il funzionamento degli algoritmi evolutivi sia chiaro, risulta poco intuitivo il motivo per cui funzionano, ovvero perchè riescano a migliorare progressivamente le soluzioni individuate.

Schema theorem

Un mezzo teorico per determinare il comportamento di un algoritmo evolutivo è dato dallo schema theorem.

Definizione 3 *Uno schema è definito come una sequenza di uni, zeri e placeholder. L'encoding di una soluzione matcha uno schema sse gli uni e gli zeri dello schema sono presenti nella rappresentazione binaria dell'encoding considerato.*

Si può definire il fitness di uno schema come il fitness medio delle soluzioni che matchano tale schema.

Lo schema theorem dimostra che schemi con elevato fitness si riprodurranno esponenzialmente, mentre schemi con fitness basso andranno a morire nelle generazioni successive. Di conseguenza, sempre più individui andranno a matchare schemi con fitness elevato, aumentando la qualità della popolazione.

No free lunch theorem

Il teorema dimostra che gli algoritmi evolutivi, in media, funzoinano tanto bene quanto qualsiasi altra metaeuristica.

Si sta di fatto enunciando l'impossibilità di stabilire a priori la qualità di una soluzione individuata da un algoritmo evolutivo per un particolare problema del quale non si conosce nulla.

3.2 Swarm e population based optimization

Swarm e population based optimization sono alcune metaeuristiche utilizzate nel mondo degli algoritmi evolutivi.

L'idea di base fa rifermineto a spiecie naturali capaci di interagire e cooperare tra loro, in modo da raggiungere un certo obiettivo.

Particle swarm

Ispirato al comportamento di uccelli e pesci, le soluzioni candidate operano aggregando informazioni creando conoscenze comuni.

L'idea è quella di guidare le soluzioni candidate considerando conoscenze locali e globali, in modo da muoverle in gruppo verso una soluzione ottima.

Ant colony

Facendo riferimento al comportamento delle formiche, le soluzioni candidate operano seguendo un certo decision graph.

Si parte da soluzioni parziali ed effettuando scelte random per renderle complete. Le decisioni prese da un singolo sono influenzate da quelle prese dai compagni.

Population-based incremental learning

Si parte da una popolazione iniziale ma, invece che memorizzarla, si mantengono le statistiche relative ad essa.

Si seguono i seguenti pattern:

- gli individui sono generati pescando dalle statistiche memorizzate. Si effettua quindi un Uniform crossover implicito alla creazione di nuove soluzioni.
- ad ogni iterazione si considerano i migliori individui per aggiornare le statistiche della popolazione.
- le mutazioni sono in modalità bit-flipping sulle statistiche
- si considera un learning rate che regola l'esplorazione e il miglioramento locale

L'approccio presenta alcuni problemi, quali la possibilità di imparare dipendenze tra i singoli bit che rappresentano le statistiche, oppure la possibile rappresentazione di popolazioni differenti tramite le stesse statistiche.

3.3 Algoritmi genetici

La programmazione genetica mira a comporre un programma in grado di risolvere un problema dato. Tale programma è espresso attraverso un *parse tree*.

Processo

Si definiscono due insiemi:

- F: funzioni e operatori
- T: simboli terminali

Una soluzione candidata è rappresentata da un parse tree, che definisce una certa espressione, componendo elementi di F e T.

Il processo è il solito seguito da un algoritmo evolutivo, ovvero creazione, selezione e applicazione di operatori genetici.

Introni

Visto che si stanno considerando parse tree che rappresentano espressioni, bisogna porre attenzione ad eventuali rami inutili nell'albero.

Un esempio, nel contesto delle operazioni matematiche, può essere $a+1-1$, dove la seconda parte dell'espressione è chiaramente inutile. Bisogna quindi tenerne conto in fase di ricombinazione e mutazione, per evitare soluzioni candidate con queste caratteristiche.

3.4 Multi-criteria optimization

È possibile avere problemi con più criteri da ottimizzare. Ad esempio, il problema di acquistare un'auto che sia economica, ma comoda e veloce.

La strada da seguire per ottimizzare non è chiara, infatti, probabilmente, migliorando uno dei tre criteri, gli altri peggioreranno.

Pareto optimal solution

Definite le funzioni da ottimizzare, è possibile individuare quei punti nello spazio per cui il miglioramento di una funzoine porta al peggioramento delle altre.

Tali punti formano il così detto *Pareto front* e concettualmente è proprio uno di quei punti che ottimizza il problema visto nel complesso.

Approccio evolutivo

L'idea è quella di applicare un algoritmo evolutivo al problema, in modo da ottenere quante più possibili soluzioni sul Pareto front.

Un approccio semplice consiste nel definire la funzione di fitness come somma pesata delle funzioni da ottimizzare. Purtroppo questo favorisce soluzioni che

massimizzano uno dei criteri, sfavorendo soluzioni intermedie, ma è possibile risolvere il problema individuando le soluzioni marginali e scartandole in fase di selezione.

Un secondo problema è quello della convergenza in un punto qualsiasi del fronte, si può rimediare applicando tecniche di *power law sharing*, simili a quelle per evitare *crowding*. Tali tecniche, tenendo conto delle zone già coperte del fronte, cercheranno di coprire punti inesplorati del fronte, in modo da garantire una copertura omogenea.

3.5 Algoritmi evolutivi paralleli

Rispetto alle altre metaeuristiche si è osservato che gli algoritmi evolutivi spesso portano a risultati ottimi, ma con il prezzo di un tempo di esecuzione molto lento.

È possibile parallelizzare alcune fasi del processo in modo da velocizzarlo o migliorarne il risultato. Osservando le varie fasi si nota che sono parallelizzabili:

- la generazione iniziale, stando attendi ad eventuali duplicati, che comunque non costituiscono grossi problemi
- il calcolo del fitness degli individi, con l'accortezza di raccogliere i dati in un unico processore per calcolare il fitness relativo
- la selezione se costituita da eventi indipendenti, come ad esempio tournament selection, diventa più complesso gestire etilismo
- l'applicazione degli operatori genetici
- il controllo di raggiungimento del criterio di terminazione

Island model

È possibile sfruttare la parallelizzazione considerando un modello ad isola. Ogni isola avrà una popolazione, ed eseguirà il processo evolutivo.

Si può introdurre migrazione degli individui da un isola all'altra in maniera random o definita da connessione tra le isole.

Cellular evolution

I processori sono organizzati in una griglia. Ogni processore è responsabile di un cromosoma.

Per la selezione ogni processore calcola il massimo dei vicini, gli operatori genetici sono applicabili solo tra vicini e la mutazione è gestita da ogni singolo processore.