Università di Perugia
Dipartimento di Matematica e Informatica

Computational Intelligence

Genetic Programmig Project

Professore
Prof. Marco Baioletti

Studente
Tommaso Romani

Anno Accademico 2022-2023

Indice

1 Introduzione

Il problema analizzato in questa relazione consiste nell'applicare un algoritmo di programmazione genetica per riuscire a trovare una funzione che riesca a rappresentare bene una data funzione sconosciuta, di cui sono noti solo i valori che essa assume in determinati punti.

2 Metodologia

La Programmazione Genetica è una categoria del più ampio argmomento degli *Evolutionary Algorithms*, strategie di ottimizzazione che si rifanno ai principi dell'evoluzione biologica.

La struttura di ognuno di questi algoritmi di ottimizzazione va individuata nei seguenti punti:

- **Rappresentazione**: come sono rappresentati gli individui.
- **Fitness function**: come viene definita la misura di bontà di ogni individuo.
- **Selection**: insieme agli operatori di vairatione sono la forza trainante degli EA, definisce le metodologie di selezione degli individui per la fase di accoppiamento e per la fase di passaggio generazionale
- **Variation**: nsieme alle tecniche di selezione sono la forza trainante degli EA, definiscono come vengono generati nuovi individui, che ricombinando materiale genetico, sono coloro che portano alla scoperta di soluzioni migliori.

2.1 Rappresentazione

Nella Programmazione Genetica ogni individuo viene rappresentato come un albero i cui nodi possono essere suddivisi in due insiemi:

1. **Function Set**: Insieme dei nodi interni, che possono rappresentare le funzioni disponibili nel nostro problema, ogni nodo avrà tanti figli quanti saranno i possibili argomenti presi dalla funzione di tale nodo.
2. **Terminal Set**: Insieme di costanti e valori che saranno rappresentate sulle foglie dell'albero.

Nel nostro caso particolare avremo:

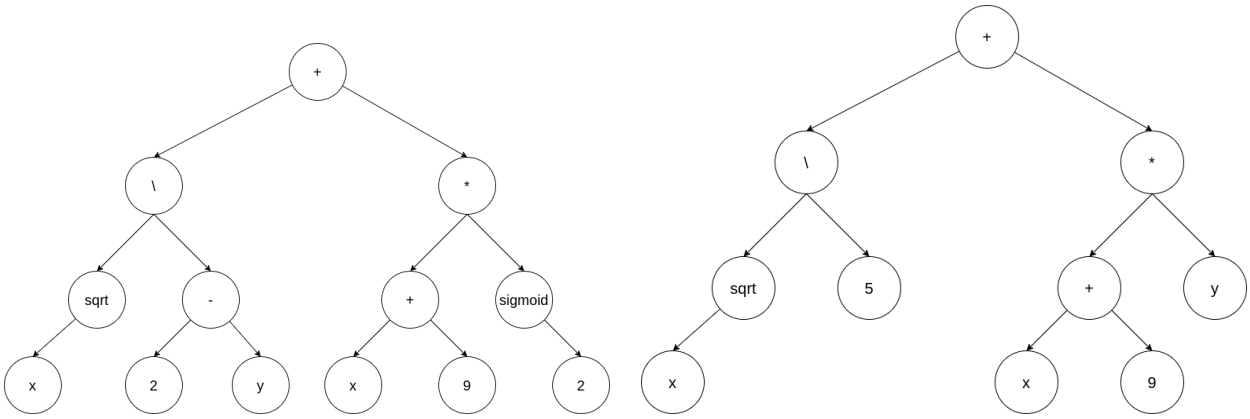
- `+,-,\,*,sigmoid`, come simboli nel **Function set**
- `1, ..., 9, x, y` come simboli nel **Terminal set**, in cui `x` e `y` saranno le nostre variabili.

Gli alberi sono stati generati ricorsivamente con l'approccio *ramped half-and-half*, in cui data un intervallo di profondità, si genera ogni albero seguendo uno di due approcci:

- **Metodo Full**: Ogni nodo interno dell'albero (fino alla profondità massima specificata) è espanso usando un operatore di funzione. Questo significa che l'albero è "pieno" in quanto ogni ramo dell'albero raggiunge la profondità massima.
- **Metodo Grow**: A differenza del metodo Full, il metodo Grow permette di terminare i rami in qualsiasi punto prima della profondità massima. In questo modo, si possono avere nodi terminali (foglie) a diverse profondità, creando alberi di forme più irregolari e varie.

Dunque un esempio delle rappresentazioni ad albero generate usando entrambi i metodi *Full* e *Grow* è la seguente.

Figura 1: Esempio di Grow e Full



2.2 Fitness Function

Nel nostro problema la fitness function è stata espressa in termini di errore quadratico medio, quindi seguendo la seguente formula:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \tilde{y}_i)$$

in cui n era il numero di entries nel dataset associato a una data funzione, y_i il risultato della funzione valutata in due punti x e y, e $\tilde{y}_i$ il risultato ottenuto valutando ricorsivamente un singolo membro della popolazione.

Dato che nella Programmazione Genetica si presenta spesso un fenomeno noto come **bloating**, in cui la dimensione dei membri della popolazione tende a crescere con l'avanzamento delle generazioni, per limitarlo durante il calcolo della fitness function abbiamo utilizzato un sistema di **Parsimony pressure**, andando a penalizzare alberi di grandi dimensione, con la seguente formula:

$$\text{element}.fitness+ = depth\_penalty \times \text{element}.depth$$

Per tutti i nostri test è stato usato un valore di *depth_penalty* pari a 200.

2.3 Selection

I meccanismi di selection utilizzati sono stati di due tipi:

- **Stocastico** per la creazione della mating pool
- **Deterministico** per la selezione della nuova generazione.

2.3.1 Parent Selection

La parent selection è stata implementata utilizzando il metodo **Stochastic Universal Sampling**(SUS). Questo metodo è progettato per migliorare la parent selection in modo più equo ed efficiente rispetto ad altri metodi come la selezione a roulette wheel.

- **Valutazione di Fitness**: Come prima cosa si calcola la fitness per ogni genitore.

- **Scala di Fitness Cumulativa**: Si calcola una scala di fitness cumulativa sommando i punteggi di fitness di tutti gli individui. Ogni individuo occupa una porzione di questa scala proporzionale al proprio fitness.
- **Puntatori Equidistanti**: Invece di selezionare i genitori casualmente, SUS usa un insieme di puntatori equidistanti. Il numero di puntatori è uguale al numero di genitori che si desidera selezionare. Ad esempio, se si desidera selezionare quattro genitori, si posizionano quattro puntatori lungo la scala di fitness a intervalli uguali.
- **Offset Casuale**: Si inizia con un offset casuale per il primo puntatore. Questo passaggio assicura che la selezione mantenga un elemento di casualità, ma riduce la varianza rispetto alla selezione a roulette wheel.
- **Selezione dei Genitori**: Ogni puntatore seleziona un genitore. A causa della disposizione equidistante dei puntatori, gli individui con fitness più elevato hanno una maggiore probabilità di essere selezionati, ma la selezione è più equa rispetto alla selezione a roulette wheel. Questo perché, in SUS, un individuo ad alta fitness non può essere scelto molteplici volte per lo stesso set di puntatori, il che riduce il rischio di selezione eccessiva degli individui più adatti.
  In particolare il numero di copie per un paricolare individuo i non sarà mai minore della parte intera di $\lambda \cdot P_{sel}(i)$ e mai maggiore di uno in più.

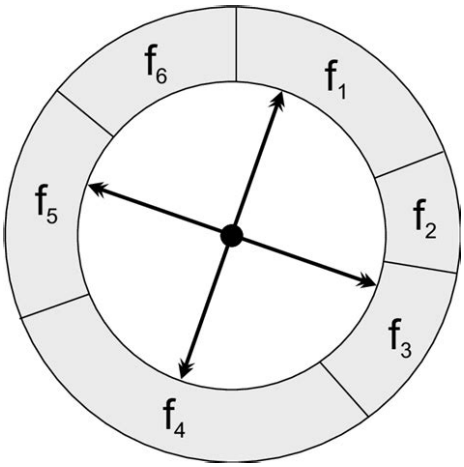


Figura 2: Esempio grafico di SUS

Se si utilizzano popolazioni molto grandi è stato anche implementato il meccanismo di **overselection**, selezionabile con un parametro.

2.3.2 Survivor Selection

Il meccanismo di Survivor Selection utilizzato è il (μ, λ).
La scelta è stata fatta per cercare di rendere il modelle più resistente possibile ai minimi locali.

2.4 Variation

Gli operatori di variazione negli algoritmi di GP sono solitamente in modo esclusivo (o applico mutation o crossover) in base alla probabilità p_{mut}, e sono funzionano nel seguente modo:

- **Crossover**: In ciascuno dei genitori viene selezionato un punto di crossover, che può essere o un nodo interno dell'albero o una foglia, e successivamente vengono scabiati i punti di crossover individuati, compresi i sottoableri radicati in loro.
- **Mutation**: Si trova un nodo casuale nell'individuo, e dopo aver generato un nuovo sottoalbero, si scarta il nodo selezionato con il sottoalbero radicato in esso, e lo si rimpiazza con in nuovo albero appena generato

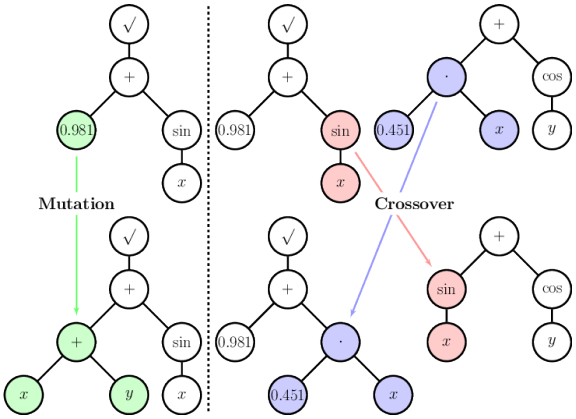


Figura 3: Mutation e Crossover visualizzati su un albero

3 Test

I test sono stati condotti utilizzando le seguenti 4 equazioni:

1. $\frac{\sqrt{9sigmoid(x)}+(yx)}{4} \cdot \frac{6x+4y}{5x+\sqrt{5xsigmoid(y)}}$
2. $6x + \frac{\sqrt{x}}{4} + \frac{9sigmoid(y)}{2} - \frac{4x-7}{y}$
3. $8sigmoid(x \cdot y) - \sqrt{\frac{7x}{2}}y + 6\frac{x}{y-3}$
4. $9x + \frac{x}{6y} - \sqrt{xy} + sigmoid(y)$

Ognuna delle quali è stata valutata su 80 diversi valori di input casuali, con cui è stato generato un dataset.

Tutti i test sono stati condotti usando i seguenti parametri:

1. $\mu = 300$
2. $\lambda = 10 * \mu$
3. `mating_pool_size` $= \frac{\mu}{2}$
4. `max_depth` $= 6$
5. `depth_penaly` $= 200$
6. $N = 200$
7. $p_{mut} = 0.1$

	MSE	**Num Generazioni**	**Generazioni senza migliorie**
funzione 1	9374.360	200	-
funzione 2	0.309	56	30
funzione 3	10.697	200	-
funzione 4	2.068	73	18

Tabella 1