Algoritmi Genetici

e programmazione genetica

Algoritmi Genetici

- Algoritmi motivati dall'analogia con l'evoluzione biologica
  - Lamarck: le specie "trasmutano" nel tempo
  - Darwin e Wallace: variazioni consistenti ereditabili si osservano negli individui di una popolazione; selezione naturale dei più sani
  - Mendel e la genetica: esiste un meccanismo per ereditare tratti genetici
- A partire da un insieme di ipotesi (**popolazione**), generano **successori** delle ipotesi producendo **perturbazioni** su di esse che si spera producano risultati migliori
- Caratteristiche:
  - Gli AG possono effettuare ricerche nello spazio di ipotesi i cui elemento interagiscono in modo complesso (ovvero laddove è difficile valutare l'impatto di un singolo elemento)
  - Gli AG sono ottimizzatori, non sono veri "apprendisti"
  - Gli AG sono facilmente parallelizzabili (si avvantaggiano del basso costo dell'hardware)

Algoritmi Genetici

- Una **Funzione Fitness** che assegna un valore di "benessere" a ogni ipotesi h
- Una **Soglia di Fitness** che specifica un criterio di terminazione
- **p** numero di ipotesi da includere nella popolazione
- **r** la frazione della popolazione che deve essere rimpiazzata dall'**operatore di incrocio (crossover)**
- **m** il **tasso di mutazione**

Algoritmo Tipo

GA(Fitness, Soglia-Fitness, p, r, m)

- **Inizializza**: P = insieme di p ipotesi
  - generate a caso o specificate a mano
- **Valuta**: per ogni h in P, calcola Fitness(h)
- **While** $\max_h$ Fitness(h) < Soglia-Fitness
  - **Seleziona**: seleziona (1-r)·p membri di P da aggiungere a una nuova generazione P_S
  - **Crossover**: Seleziona r·p/2 coppie di ipotesi da P
    - Per ogni coppia (h_1, h_2) produci due **successori** (**offspring**) applicando l'operatore di crossover
  - **Muta**: Inverti un bit a caso di m% individui di P_S scelti a caso
  - **Aggiorna**: P = P_S
  - **Valuta**: per ogni h in P, calcola Fitness(h)
- **Return** $\text{argmax}_{h \in P}$ Fitness(h)

Rappresentazione delle Ipotesi (1)

- La rappresentazione di base in GA è la stringa binaria (**cromosoma**)
  - La mappatura dipende dal dominio e dal progetto
- La stringa di bit rappresenta una ipotesi
- Elementi dell'ipotesi (es. clausole di una regola, caratteristiche, ecc.) sono rappresentati ciascuno da una sottostringa che si trova in una posizione specifica (se la stringa ha lunghezza fissa)

Rappresentazione delle Ipotesi (2)

- Esempio 1: rappresentiamo l'ipotesi

  (Outlook = **Overcast** ∨ **Rain**) ∧ (Wind = **Strong**)

  con:

Outlook	**Wind**
011	**10**

- Esempio 2: rappresentiamo l'ipotesi

  IF Wind = **Strong** **THEN** PlayTennis = **yes**

  con:

Outlook	**Wind**	**PlayTennis**
111	**10**	**10**

- Esempio 3: rappresentiamo un'architettura di rete neurale definita da un grafo codificato mediante una stringa binaria

Operatori per gli Algoritmi Genetici

	Stringhe iniziali	*Maschera di crossover*	*Successori (offspring)*
Single-point crossover	11101001000 00001010101	11111000000	11101010101 00001001000
Two-point crossover	11101001000 00001010101	00111110000	00101000101 11001011000
Uniform crossover	11101001000 00001010101	10011010011	10001000100 01101011001
Point mutation	11101001000		11101011000

- Gli **operatori di crossover** creano dei discendenti di una coppia di ipotesi "mescolando" le caratteristiche dei due genitori
- L'**operatore di mutazione** crea un discendente da un'ipotesi invertendo un bit scelto a caso
- Altri operatori specializzati possono essere definiti sulla base del problema specifico

Selezione delle ipotesi migliori (fittest hypotheses)

- La selezione degli (1-r)·p membri avviene sulla base della probabilità:

$$\Pr(h_i) = \frac{Fitness(h_i)}{\sum_{j=1}^{p} Fitness(h_j)}$$

- Può causare **crowding** (alcuni individui tendono a prendere il sopravvento, riducendo drasticamente la diversità della popolazione)

GABIL (DeJong et al., 1993)

- Un sistema per apprendere insiemi di regole proposizionali
- Ogni ipotesi corrisponde a un insieme di regole
- La rappresentazione a stringhe di ciascuna regole viene concatenata alle altre
  - La lunghezza complessiva della stringa cresce con l'aumentare delle regole
    - **IF** $a_1 = T \wedge a_2 = F$ **THEN** $c = T$; **IF** $a_2 = T$ **THEN** $c = F$

a_1	a_2	c	a_1	a_2	c
10	01	1	11	10	0

  - Fitness(h) = correct(h)^2
    - Correct(h) è il numero di esempi di addestramento correttamente classificati da h
  - Point mutation standard e operatore crossover che preservi la ben formatezza dell'ipotesi

GABIL: operatore di crossover

- L'operatore di crossover è un'estensione del two-point crossover
- Date due ipotesi h_1 e h_2 si seleziona per h_1 un intervallo di bit (m_1, m_2)
- Si calcola la distanza d_1 (d_2) di m_1 (m_2) dal confine della regola immediatamente alla sua sinistra
- Per h_2 si sceglie un intervallo di bit tale che preservi le stesse distanze d_1 e d_2
- Esempio:

	a_1 a_2 c	a_1 a_2 c	a_1 a_2 c
h_1:	1[0 01 1	11 1]0 0	$d_1 = 1$, $d_2 = 3$
h_2:	0[1 1]1 0	10 01 0	$d_1 = 1$, $d_2 = 3$

ottenendo come offspring:

h_3:	11 10 0		
h_4:	00 01 1	11 11 0	10 01 0

GABIL: estensioni

- Aggiungiamo due operatori specializzati da applicare con una certa probabilità:
  - AggiungiAlternativa (AA)
    - Generalizza il vincolo su un attributo cambiando uno 0 in 1 nella sottostringa corrispondente all'attributo
  - EliminaCondizione (EC)
    - Generalizzazione ancora più drastica: rimpiazza tutti i bit di un attributo con 1
- Possiamo anche aggiungere due bit alla fine di ogni ipotesi, AA e EC, per indicare se su quell'individuo si potranno applicare i due operatori oppure no
  - Questi bit possono essere modificati da una generazione all'altra come tutti gli altri mediante crossover e mutazione
  - **Anche la strategia di apprendimento evolve!**

Ricerca nello spazio delle ipotesi

- Gli AG effettuano **ricerche randomizzate** sullo spazio delle ipotesi
  - Differente rispetto agli altri metodi di apprendimento (es. BackPropagation si muove molto più lentamente da un’ipotesi all’altra)
- E’ meno probabile che gli AG cadano in un minimo locale
- Il **crowding** è una difficoltà reale: alcuni individui con buon fitness si riproducono velocemente e copie simili a queste prendono il sopravvento sulla popolazione, riducendone la diversità
  - Soluzione: creare perturbazioni o alterazioni della funzione di selezione
- Tempi di apprendimento lunghi (necessario hw “ad hoc”)
- Prestazioni paragonabili a C4.5

Risolvere il problema del crowding

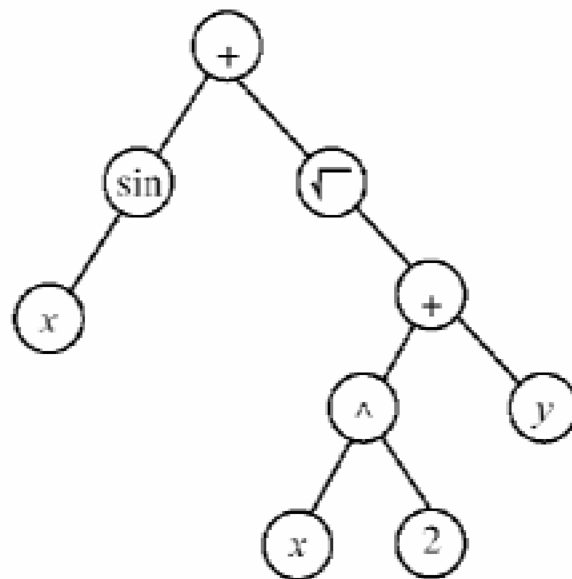
- **Tournament selection** (selezione per torneo):
  - Scegli h_1 e h_2 casualmente con probabilità uniforme
  - Con probabilità p, seleziona l'ipotesi migliore delle due (la peggiore con probabilità 1-p)
- **Rank selection (selezione per grado)**:
  - Ordina tutte le ipotesi per fitness
  - La probabilità di selezione è proporzionale al rank

Programmazione Genetica

- Metodologia di **programmazione evolutiva** in cui gli individui nella popolazione in evoluzione sono programmi informatici
- La PG utilizza la stessa struttura algoritmica degli AG
- I programmi manipolati dalla PG sono rappresentati da alberi che corrispondono agli alberi sintattici (**parse tree**) dei programmi

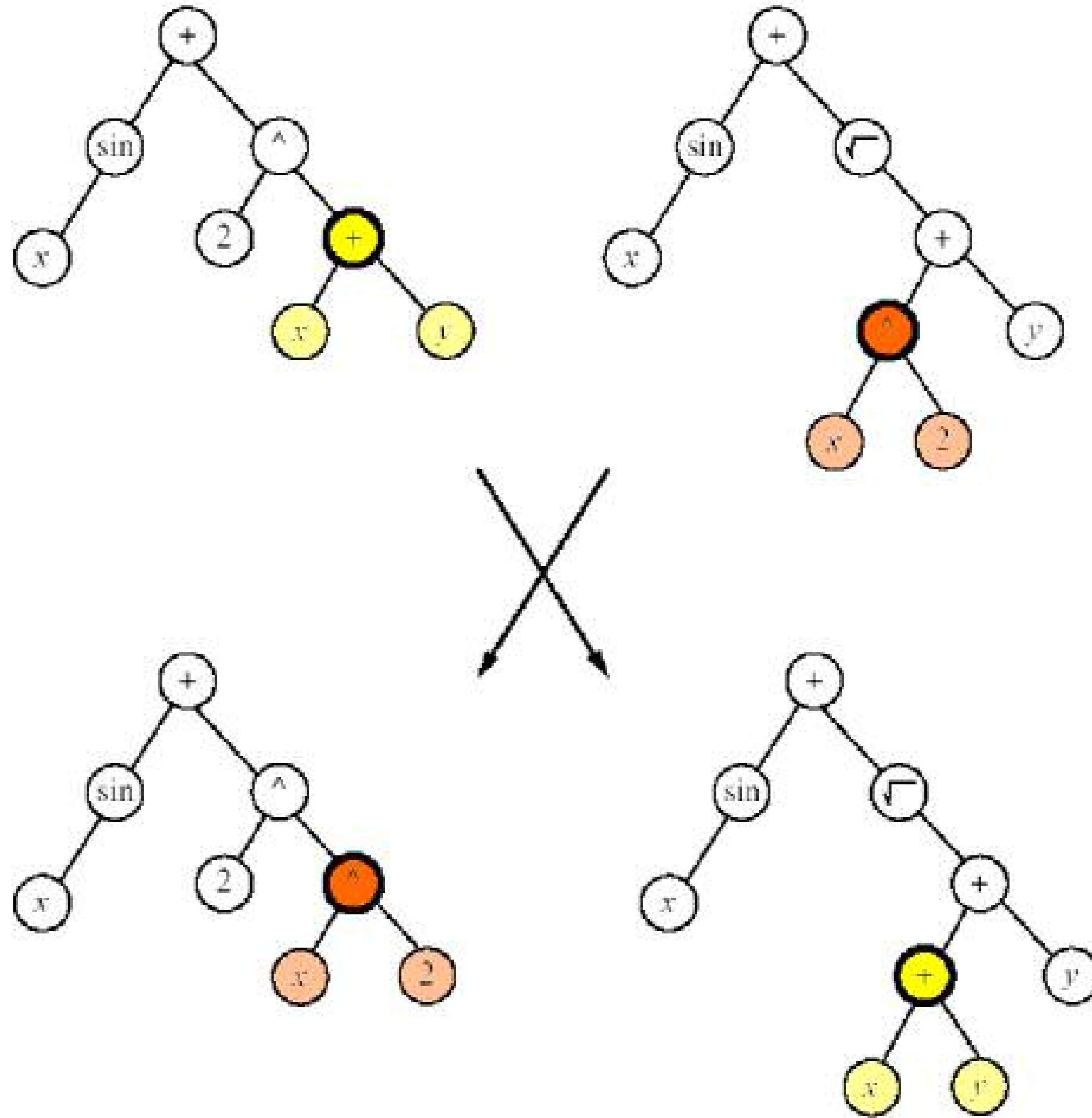
PG: un esempio

- Supponiamo che il nostro programma calcoli la funzione: $sen(x) + \sqrt{x^2 + y}$
- Il suo albero sintattico è il seguente:



- Simboli terminali: *x*, *y* e costanti (es. 2)
- Funzioni: sen, +, radice, quadrato

Crossover nella PG

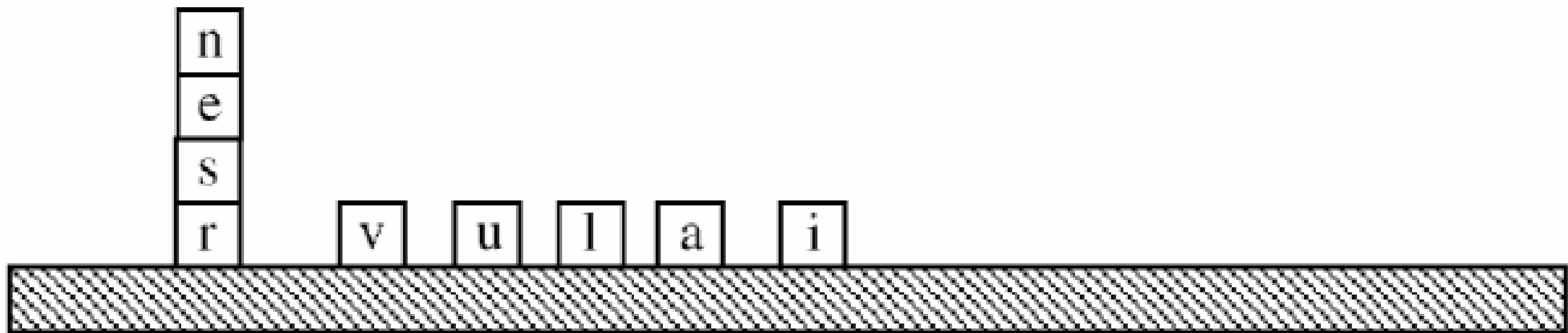


Passi preparatori per la PG

- Determinare l'insieme dei terminali ammessi
- Determinare l'insieme delle funzioni
- Determinare la misura di fitness:
  - Il fitness di un singolo programma (individuo) è determinato dall'accuratezza del programma eseguito su un insieme di addestramento
- Determinare i parametri per il run
- Determinare il criterio per terminare un run

Esempio:
Il problema dei blocchi (Koza, 1992)

- Vogliamo sviluppare un algoritmo che prenda in input qualsiasi configurazione iniziale di blocchi distribuiti a caso tra una pila e il tavolo e li inserisca correttamente nella pila nell'ordine corretto (per leggere: UNIVERSAL)
- Azioni permesse:
  - Il blocco in cima alla pila può essere spostato sul tavolo
  - Un blocco sul tavolo può essere spostato in cima alla pila



Il problema dei blocchi (Koza, 1992)

- La scelta della rappresentazione può influenzare la facilità di risolvere il problema
- Simboli terminali:
  - CS (“current stack”) = denota il blocco in cima alla pila o F se non c’è nulla
  - TB (“top correct block”) = nome del blocco in cima alla pila tale che tutti i blocchi sulla pila sono nell’ordine corretto
  - NN (“next necessary”) = nome del prossimo blocco richiesto sopra TB sulla pila per poter leggere “universal” o F se abbiamo finito

Il problema dei blocchi (Koza, 1992)

- Funzioni:
  - (MS x): ("move to stack"), se il blocco x è sul tavolo, sposta x in cima alla pila e restituisce il valore T. Altrimenti non fa niente e restituisce il valore F
  - (MT x): ("move to table"), se il blocco x è da qualche parte nello stack, sposta il blocco che è in cima allo stack sul tavolo e restituisce il valore T. Altrimenti restituisce F.
  - (EQ x y): ("equal"), restituisce T se x è uguale a y e restituisce F altrimenti
  - (NOT x): restituisce T se $x = F$, altrimenti restituisce F
  - (DU x y): ("do until") esegue l'espressione x ripetutamente finché l'espressione y restituisce il valore T

Il programma appreso

- Allenato su 166 configurazioni iniziali di blocchi
- Il fitness di qualsiasi programma è dato dal numero di esempi risolti dallo stesso
- Usando una popolazione di 300 programmi, si trova il seguente programma dopo 10 generazioni che risolvono i 166 problemi:

(EQ (DU (MT CS)(NOT CS)) (DU (MS NN)(NOT NN)))