Variazione genetica e selezione naturale

Gli Algoritmi Genetici (AG), proposti nel 1975 da J.H. Holland, sono un modello computazionale idealizzato dall'evoluzione naturale darwinista. Ogni individuo ha sue caratteristiche e proprietà specifiche, manifestate esternamente e "visibili", che ne costituiscono il *fenotipo*. È il fenotipo a dettare le possibilità e i limiti delle interazioni dell'individuo con l'ambiente in cui vive. Ma il fenotipo è determinato sostanzialmente dall'invisibile patrimonio genetico o *genotipo*, costituito dai geni, che sono le unità fondamentali dei cromosomi. Ad ogni gene corrisponde, in generale, un caratteristico fenotipo. Pertanto la sopravvivenza degli individui con caratteristiche più adatte, significa in realtà la sopravvivenza dei geni più adatti. I due principi fondamentali dell'evoluzione sono la *variazione genetica* e la *selezione naturale*. Affinché la popolazione possa evolvere, gli individui che la costituiscono devono anzitutto avere una ricca varietà di fenotipi e quindi di genotipi. Può allora scattare la selezione, che premia la sopravvivenza, la longevità e la riproduzione degli individui più adatti. I meccanismi generatori della varietà del genotipo sono sostanzialmente due: un processo combinatorio dei geni, grazie ai diversi apporti dei genitori, nell'ambito della riproduzione sessuale, e le mutazioni geniche casuali. Le mutazioni producono nuovi geni, alcuni dei quali si tramandano alle generazioni successive, mentre altri scompaiono e il cosiddetto *pool di geni*, nel quale "pesca" la selezione naturale, cambia continuamente. I cambiamenti che si verificano da una generazione all'altra sono molto piccoli ma quelli positivi si accumulano (selezione cumulativa) e, nell'arco di tempi lunghissimi, danno origine a cambiamenti enormi. Secondo la moderna versione degli *equilibri punteggiati*, l'evoluzione sarebbe fortemente influenzata da eventi eccezionali e soprattutto avverrebbe anche *per salti*. Ciò significa che a periodi di ristagno, che possono essere anche lunghissimi, seguono periodi di accelerazione evolutiva relativamente brevi.

Come già aveva notato Darwin, in alcuni casi bastano periodi limitati di tempo affinché si verifichino cambiamenti notevoli. Un esempio è costituito dalla selezione di animali domestici, guidata dall'uomo anziché dall'ambiente, che ha prodotto nel giro di poche migliaia di anni razze canine così diverse come il pastore tedesco e il bassotto. Un esempio di rapidità evolutiva ancora maggiore è quello verificatosi in Inghilterra in piena era industriale. Prima dell'avvento della macchina a vapore, esisteva una specie di farfalle di colore chiaro. Con lo sviluppo delle miniere di carbone, queste farfalle risaltavano sui tronchi d'albero, anneriti dal pulviscolo, e diventavano facile preda degli uccelli. Senonché esistevano poche ma decisive farfalle di colore alquanto scuro (variazione genetica, caso) che poterono sopravvivere e moltiplicarsi (selezione naturale, necessità), originando così in poche decine di anni una specie diversa di farfalle aventi colore decisamente nero.

Algoritmi genetici

Sono tecniche euristiche di calcolo *general purpose*, relativamente nuove, ispirate dalla meccanica della selezione naturale. Gli AG vengono applicati in un ampio spettro di problematiche: da problemi di natura prettamente controllistica per gasdotti, altiforni e la guida di missili terra-aria, a problemi di natura ottimizzatoria come quello del commesso viaggiatore e quelli ben più vasti di scheduling, dai problemi di ingegnerizzazione come la progettazione di turbine e parti aerodinamiche di velivoli, alla modellazione di mercati finanziari telematici. Molti problemi di scheduling, come il flow-shop, sono problemi di permutazione che possono essere gestiti tramite gli algoritmi genetici basati su incroci a permutazione. Spesso tali algoritmi vengono realizzati insieme ad euristiche più semplici per la risoluzione di problemi di scheduling più complessi. Tali euristiche servono a realizzare soluzioni ammissibili, mentre l'algoritmo genetico si concentra sulla ricerca di ottimalità. Altri approcci utilizzano gli algoritmi genetici per governare la preferenza di una euristica rispetto un'altra. Gli AG vengono usati anche per approcci multi obiettivo su problemi di scheduling o per ricercare soluzioni Pareto ottimali. Gli AG non hanno ancora avuto una elevata diffusione in problemi di ricerca operativa e di *decision making* e ancora vi è nell'ambito scientifico una certa riluttanza nel loro utilizzo. Ciò è dovuto in parte a ragioni storiche in parti a difficoltà implementative.

Per quanto attiene alle ragioni storiche, dopo la prima diffusione degli AG da parte dell'Università del Michigan, i ricercatori preferirono presentare i risultati in convegni specializzati invece che su convegni e riviste orientati alle applicazioni. Ciò perché, a differenza del SA che per le sue caratteristiche è assimilabile ad una variazione della *random search*, la natura totalmente differente degli AG rispetto le usuali tecniche di ottimizzazione non consente un confronto di performance immediato con altri approcci. I problemi di natura implementativa derivano dalla stretta correlazione fra metodica algoritmica e fenomeno naturale, di per se alquanto complessa sia in termini fenomenici che nella terminologia. Gli AG utilizzano la *sopravvivenza* dell'*individuo* in popolazioni di regole decisionali o di soluzioni di problemi o di valori di funzioni. Le *creature* (regole codificate come stringhe di informazioni logiche) che hanno *successo* nella popolazione hanno l'opportunità di *accoppiarsi* e produrre progenie, la cui struttura combina caratteristiche ereditate dalle stringhe dei loro genitori. L'evoluzione della popolazione di stringhe è casuale, ma il suo tasso di miglioramento è fortemente superiore a quello di una semplice *random walk*. Gli AG (come l'evoluzione biologica) impiegano l'esperienza passata per produrre una ricerca efficiente con performance elevate. A differenza degli algoritmi SA, che esplorano lo spazio delle soluzioni sequenzialmente, gli AG lavorano con popolazioni di soluzioni (*parallelismo intrinseco*) e tentano di guidare la ricerca attraverso miglioramenti, utilizzando il principio di sopravvivenza delle migliori soluzioni.

La qualità di ogni soluzione è misurata con la *fitness* (equivalente alla funzione obiettivo o alla funzione costo nei tradizionali metodi di ottimizzazione) che è la decodifica della stringa genetica. In tal senso la stringa viene chiamata *genotipo*, mentre la soluzione decodificata prende il nome di *fenotipo*. La ricerca procede con una serie di generazioni in cui ogni individuo contribuisce alla successiva generazione in proporzione alla sua fitness. Ciò viene realizzato selezionando gli individui in modo casuale. A tal fine viene utilizzata una funzione di probabilità pesata tramite i valori della fitness, o i valori del fenotipo scalati o, ancora, un semplice rango.

Rappresentazione genetica di un problema

Gli AG quindi risolvono un determinato problema ricorrendo a una popolazione di soluzioni che, inizialmente casuali e quindi con fitness bassa, vengono poi fatte evolvere per un certo numero di generazioni successive, sino all'apparizione di almeno una soluzione con fitness elevata. Per poter applicare l'algoritmo genetico, occorre anzitutto codificare numericamente le soluzioni e individuare una opportuna funzione di fitness.

Codifica numerica

Ogni individuo della popolazione è codificato da un *cromosoma*, una stringa di lunghezza costante formata da *geni*. Generalmente i geni sono binari, con valori 0 o 1. Se abbiamo N geni a disposizione si possono realizzare 2^N stringhe diverse. Il valore dei geni si chiama *allele*.

Scelta della funzione di fitness

La funzione di fitness è la misura numerica della "bontà" di una soluzione, generalmente normalizzata tra 0 e 1, e fortemente dipendente dal problema che occorre risolvere.

Alcuni esempi di codifica numerica e di fitness

Esempio 1. Supponiamo di dover determinare il massimo della funzione:

$$Y = X + |\sin(32\,X)|$$

con $0 \le X \le \pi$. Tale massimo non può essere determinato per via analitica (annullando la derivata della funzione), per la presenza del valore assoluto del seno. Esso può essere trovato facilmente dall'algoritmo genetico con stringhe binarie che rappresentano il valore reale di X.

Questo valore varia tra 0 e 3.1415... : accontentandoci della precisione di 1/100 dovremo quindi codificare 314 valori diversi, con stringhe di 9 bit ($2^9 = 512$ mentre $2^8 = 256$). La funzione di fitness coincide ovviamente con la stessa funzione *Y*.

Esempio 2. Supponiamo di avere un labirinto dato e un robot che debba percorrerlo dall'entrata all'uscita in un massimo di 100 passi. I movimenti possibili di ogni passo siano 4 (avanti A, indietro I, a sinistra S, a destra D), così che una generica sequenza potrebbe essere:

DSSAADSI ……D

Codificando ognuna delle 4 mosse con 2 bit, una stringa generica avrà $2 \times 100 = 200$ bit. La fitness di una stringa può essere misurata dalla distanza fra l'uscita del labirinto e il punto in cui si troverebbe il robot dopo avere eseguito le 100 mosse ammesse.

Esempio 3. Consideriamo il semplice problema di trovare la regola logica che sintetizza i casi accettabili della tabella seguente:

Caso	Prezzo	Qualità	Accettabilità
1	450	3	si
2	500	3	si
3	600	3	no
4	450	2	si
5	500	2	si
6	600	2	no
7	450	1	no
8	500	1	no
9	600	1	no

La regola cercata sia del tipo: **(Prezzo O_1 V_P) O_2 (Qualità O_1 V_Q)**

dove:

O_1 è un operatore con valori possibili: "≥" e "≤" (rispettivamente 0 e l);

O_2 è un operatore con valori possibili: "and" e "or" (rispettivamente 0 e 1);

V_P è il valore del prezzo: 450, 500, 600 (rispettivamente 01, 10, 11);

V_Q è il valore della qualità: 1, 2, 3 (rispettivamente 01, 10, 11).

Secondo tale codifica, la stringa di 7 bit: <0111101>

equivale alla regola: (Prezzo ≥ 600) or (Qualità ≤ 1).

Per quanto riguarda la fitness di una regola, essa è uguale al rapporto tra il numero di casi accettabili che essa soddisfa e il numero totale dei casi (la regola precedente ha fitness = 0). La soluzione del problema, tramite l'algoritmo genetico, è fornita dalla stringa:

<1100010> = (Prezzo ≤ 500) and (Qualità ≥ 2)

che ha appunto una fitness = 1.

Operazioni Genetiche

La popolazione che evolve ha un numero M di individui ($M << 2^N$), che viene mantenuto costante da una generazione all'altra. Ad ogni generazione, vengono eseguite opportune *operazioni genetiche* che producono nuovi individui e quindi varietà. Per mantenere costante M, occorre copiare (riprodurre) nella generazione successiva solo M individui ed essi vengono selezionati con criteri probabilistici, premiando tendenzialmente quelli dotati di maggiore fitness. Le operazioni genetiche più importanti sono il *cross-over*, la *mutazione* e l'*inversione*:

Cross-over (incrocio)

Questa operazione coinvolge 2 stringhe "genitrici"

$$< A_1, A_2, \ldots, A_n >$$
$$< B_1, B_2, \ldots, B_n >$$

e, dopo la scelta casuale di una posizione k ($1 \leq k \leq n$), o punto di cross-over, effettua lo scambio di geni che produce le stringhe "figlie":

$$< A_1, A_2, \ldots, A_k, B_{k+1}, B_{k+2}, \ldots, B_n >$$
$$< B_1, B_2, \ldots, B_k, A_{k+1}, A_{k+2}, \ldots, A_n >$$

Una variante di questa operazione sceglie casualmente due punti di crossover k, l ($k \leq l$) nelle due stringhe genitrici:

$$< A_1, A_2, \ldots, A_k, \ldots, A_l, \ldots, A_n >$$
$$< B_1, B_2, \ldots, B_k, \ldots, B_l, \ldots, B_n >$$

ed effettua lo scambio:

$$< A_1, A_2, \ldots, A_k, B_{k+1}, \ldots, B_{l-1}, A_l, \ldots, A_n >$$
$$< B_1, B_2, \ldots, B_k, A_{k+1}, \ldots, A_{l-1}, B_l, \ldots, B_n >$$

Un'altra variante considera ogni stringa come un anello chiuso diviso, da k punti di cross-over, in k segmenti che vengono scambiati tra le due stringhe in modo alternato. Un'ultima variante scambia, con una probabilità prefissata, ogni bit di una stringa con quello corrispondente dell'altra.

Mutazione

La mutazione riguarda un singolo bit di una stringa che viene cambiato nel valore opposto, con una probabilità prefissata. Si potrebbe così avere, scegliendo a caso il k-esimo bit:

$$< A_1, A_2, \ldots, A_k = 0, \ldots, A_n > \rightarrow < A_1, A_2, \ldots, A'_k = 1, \ldots, A_n >$$

Inversione

L'inversione riguarda una sola stringa; scegliendo a caso un punto k di inversione, si ha:

$$< A_1, A_2, \ldots, A_k, A_{k+1}, \ldots, A_n > \rightarrow < A_1, A_2, \ldots, A_k, A_n, A_{n-1}, \ldots, A_{k+1} >$$

Anche per l'inversione, come per il cross-over, esistono varianti a due 0 più punti di inversione.

L'incrocio assicura, assieme alla duplicazione, il mantenimento di buoni individui per migliorare le soluzioni, mentre la mutazione e l'inversione mantiene la diversità nella popolazione e permette di ampliare l'esplorazione. Tutti e tre gli operatori dipendono dal caso ossia dalla probabilità di incrocio, di riproduzione e di mutazione. Quest'ultima è peraltro molto bassa, dell'ordine di circa 0,001 - 0,0001.

Algoritmo genetico di base

Lo sviluppo di un algoritmo genetico nella risoluzione di un particolare problema coinvolge due tipi di decisione. La prima riguarda il modo in cui il problema deve essere modellato e include la definizione dello spazio delle soluzioni ammissibili, la forma della funzione di fitness e il modo in cui gli individui devono essere presentati come stringhe. La seconda concerne i parametri dell'algoritmo genetico stesso e include le proporzioni della popolazione da riprodurre, da incrociare e mutare, la procedura di selezione, la grandezza della popolazione, il numero di generazioni, e decisioni relative a variazioni rispetto l'algoritmo di base.

L'algoritmo genetico di base, di cui esistono molte varianti, è il seguente:

1. inizializzazione casuale di una popolazione di M cromosomi e calcolo delle fitness f_i;
2. scelta di due cromosomi, ciascuno con probabilità proporzionale a f_i, ed esecuzione dell'operazione di cross-over con probabilità P_{cross}, ottenendo due nuovi cromosomi (dei quali si calcola la fitness);
3. scelta casuale di una stringa e di un suo bit, esecuzione quindi dell'operazione di mutazione con probabilità $P_{mut} << P_{cross}$, ottenendo un nuovo cromosoma (di cui si calcola la fitness);
4. riproduzione di M cromosomi (eliminando quindi i 3 soprannumerari) con probabilità proporzionale a f_i; generalmente il cromosoma migliore viene comunque riprodotto nella generazione successiva, per non rischiare di perderne il prezioso patrimonio genetico;
5. ritorno al precedente punto 2, sino al soddisfacimento di un criterio prefissato, come l'avvenuta esecuzione di un determinato numero di cicli, l'ottenimento di almeno un cromosoma con fitness superiore a un valore prefissato o il conseguimento di un prefissato grado di omogeneità nella popolazione.

La scelta di un cromosoma con probabilità proporzionale a f_i (compreso tra 0 e 1) può realizzarsi mediante la generazione di un numero a caso ε con distribuzione uniforme tra 0 e 1 e la verifica che esso sia inferiore a f_i. Un metodo analogo è quello della roulette:

1. La circonferenza di un immaginario cerchio viene divisa in *M* segmenti con apertura angolare:

$$A_i = \frac{2\pi f_i}{\sum(f_i)}$$

quindi con: $$\sum_i (A_i) = 2\pi$$

2. per ogni f_i si genera un numero a caso *k*, con distribuzione uniforme tra 0 e 2π;
3. il cromosoma (*i*) viene scelto se *k* cade nel relativo settore A_i.

Generalmente la popolazione comprende da 30 a 200 individui, la probabilità di cross-over varia da 0.5 a 1 e la probabilità di mutazione da 0.001 a 0.05. Secondo un criterio empirico, suggerito dalla esperienza, se la popolazione è piccola, entrambe le probabilità dovrebbero essere più grandi che nel caso contrario, ad esempio:

Popolazione	P_{cross}	P_{mut}
l00	0.6	0.001
30	0.9	0.01

L'esecuzione dell'algoritmo viene interrotta di solito alla convergenza della popolazione, ossia quando la varianza della popolazione è nulla, o equivalentemente, quando tutti i valori di fitness nella popolazione sono identici.

Una spiegazione matematica dell'algoritmo genetico?

Per gli algoritmi genetici non esiste una teoria completa e rigorosa che spieghi tutto, suggerisca i parametri più opportuni da adottare ecc. Esiste tuttavia un **teorema di Holland** che assicura la convergenza dell'algoritmo genetico verso una soluzione ottimale. Esso è la controparte matematica dell'ipotesi euristica dei cosiddetti *building blocks*. Vediamo in breve di cosa si tratta.

Nell'ambito di una stringa ci possono essere dei segmenti che contribuiscono molto alla soluzione ottimale come, supponiamo, i blocchi (011) e (110) nei due *schemi* seguenti a 8 bit (il simbolo # indica qualunque valore):

0 1 1 # # # # #

1 1 0

(*NB*: ogni schema rappresenta un sotto-insieme di stringhe aventi tutte gli stessi building blocks).

Le operazioni genetiche favoriscono generalmente, ma non sempre, il "montaggio" dei building blocks in schemi di fitness crescente come, supponiamo:

0 1 1 # # 1 1 0

Il teorema di Holland dimostra appunto che, sotto certe condizioni, gli schemi con fitness superiore alla media tendono a crescere esponenzialmente nella popolazione. Quanto sopra implica che le "bontà" dei building blocks siano additive.

Esiste però talvolta un sottile fenomeno di interazioni non lineari tra i bit di una stringa (*epistaticità*) per il quale non è detto che abbinando building blocks di per sé "buoni" si ottenga una stringa "più buona". Non sempre quindi l'operazione genetica di cross-over produce buoni risultati e talvolta da due cromosomi relativamente "buoni" se ne produce uno decisamente "cattivo".

Nonostante la mancanza di una solida teoria globale e il permanere di alcune questioni controverse, gli algoritmi genetici si sono dimostrati molto utili per le applicazioni pratiche, dove rivaleggiano validamente o collaborano con le reti neuronali, le euristiche fondate su criteri di scelta, gli algoritmi evolutivi.

Variazioni dell'algoritmo genetico di base

Steady-state.

In tale modello, a differenza di quello *generazionale* (AG classico), solo alcuni individui vengono sostituiti ad ogni generazione. Di conseguenza il numero di sostituzioni diventa un parametro strategico e inoltre è necessario che i figli siano sufficientemente diversi dai genitori per ottenere una buona convergenza.

Si noti che, come regola grossolana, una iterazione del modello generazionale corrisponde a un numero di iterazioni dell'approccio steady-state pari a circa metà della dimensione della popolazione.

Per maggior chiarezza in tabella sono riportate le principali differenze fra modello generazionale e modello steady-state

Tipo di algoritmo genetico	**Funzione**	**Modalità**
generazionale	selezione sostituzione	roulette aggiunta
steady-state	selezione sostituzione	rank biased tramite rank

Metodo di selezione rank biased.

Ha un range [1.0, 2.0]. Tale tipo di selezione assegna ad ogni individuo un *rango* che dipende dalla propria fitness. In tal modo è possibile individuare nella popolazione un certo numero di sottopopolazioni in base al proprio rango (aree differenti dell'insieme di soluzioni). Tra queste sottopopolazioni si ha un modello di *migrazione* con flussi migratori limitati. Viene preferita l'interazione con il vicino rispetto ad un incrocio misto fra due individui *stranieri*.

Metodo di incrocio uniforme.

Ogni posizione è intercambiabile con una data probabilità. Il *tasso di incrocio* varia fra 0 e 1. 0 equivale a non esservi incrocio ma solo duplicazione e mutazione, 1 equivale al fatto che ad ogni generazione tutti gli individui vengono generati dall'incrocio senza clonazione (non esiste riproduzione).

Mutazione per swap.

Effettua lo scambio degli alleli di due *geni* della stessa stringa cromosomica. Il tasso di mutazione varia in un range [0.0, 1.0]. Di solito è 0.001.

Elitismo.

L'elitismo ha due funzioni: per un algoritmo generazionale, effettua due copie del miglior individuo della vecchia popolazione che inserisce nella nuova popolazione, assicurando la sopravvivenza del miglior codice genetico. L'altra azione ha effetto su entrambe i modelli. In questo caso l'elitismo seleziona i due migliori cromosomi dai genitori e figli. Quindi se un figlio non sarà performante come il genitore, non verrà inserito nella nuova popolazione.

Bibliografia

Cammarata S., *Sistemi a logica fuzzy*, ETASLIBRI, pp. 197-207, 199x.

Corcoran A. L., Wainwright R. L., *LibGA: A User-Friendly Workbench for Order-Based Genetic Algorithm Research,* Proceedings of the 1993 ACM/SIGAPP Symposium on Applied Computing (SAC 93), Indianapolis, Indiana, February 14-16, 1993

Dowsland K.A., *Genetic Algorithms - A Tool for OR ?*, Journal of the Operational Research Society, vol. 47, n. 4, pp. 550-561, 1996

Goldberg D.E., *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley Publishing Company, Inc., 1989