Algoritmos genéticos

Tratamiento Inteligente de la Información y Aplicaciones

Juan A. Botía Blaya
juanbot@um.es
Departamento de Ingeniería de la Información y las Comunicaciones
Universidad de Murcia

November 15, 2007

Introducción [4]

- Los GAs son algoritmos inspirados en los mecanismos de adaptación de los seres vivos
- Aplicación en problemas de búsqueda, optimización y AC
- Búsqueda *beam* $\longrightarrow$ población de individuos
- Individuo $\equiv$ posible solución a un problema
- Cada individuo presenta un nivel de adaptación al entorno
- Los mejor adaptados tienen más probabilidades de reproducirse en la siguiente generación

Diferencias con el resto de paradigmas

- Diferente al Backprop
  - saltos bruscos en el espacio
- Diferente al ID3
  - búsqueda aleatoria

Popularid de los GAs

La popularidad de los algoritmos genéticos se debe a las siguientes razones

- Está sobradamente probado que la evolución es un exitoso y robusto método para la adaptación en sistemas biológicos
- Se desenvuelven bien en espacios de hipótesis complejos en los que es difícil determinar el impacto de cada parte de la hipótesis en el rendimiento global
- Son facilmente paralelizables

Algoritmo

GA(Fitness, α, p, r, m)

- Siendo α el umbral de adaptación, p el tamaño de la población, r la proporción de hipótesis a remplazar en cada generación y m el ratio de mutación.
- Inicializar población: $P \leftarrow$ Generar p hipotesis aleatoriamente
- Evaluar: para cada $h \in P$, calcular $Fitness(h)$
- While $[\max_h Fitness(h)] < \alpha$ hacer
  1. **Seleccionar**: probabilítiscamente, seleccionar $(1 - r)p$ de P para añadir a P_s usando

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

  2. **Entrecruzar**: probabilísticamente seleccionar $r * p/2$ pares de hipotesis de P. Para cada par, $< h_1, h_2 >$, generar dos descendientes aplicando el operador de entrecruzamiento. Añadir toda nueva hipótesis a P_s.
  3. **Mutar**: Invertir un bit, seleccionado con probabilidad uniforme, en $m * p$ miembros de P_s
  4. **Actualizar**: $P \leftarrow P_s$
  5. **Evaluar**: para cada $h \in P$, calcular $Fitness(h)$
- Devolver la hipótesis $h \in P$ con mayor fitness

Puntos importantes

1. Codificación de los individuos
2. Condición de terminación
3. Se debe evaluar cada individuo
4. Para general una nueva población se han de usar operadores genéticos. Existen dos familias de estos operadores, los de entrecruzamiento y los de mutación.

Representación de individuos

- Codificación en ristra de bits de longitud fija.
- Ejemplo: regla IF-THEN [2]
  - Antecedente

$$(Tiempo = Nublado \vee Lluvioso) \wedge (Viento = Fuerte)$$

lo representamos como

Tiempo	Viento
011	10

  - Regla completa

$$Si\ Viento = Fuerte\ ENTONCES\ JugarTenis = Si$$

lo representamos como

Tiempo	Viento	JugarTenis
111	10	10

Representación de individuos (II)

- Para ejemplos anómalos, los controlamos
  1. Restringiendo la codificación
  2. Usando operadores genéticos adecuados
  3. Asignando una adaptación suficientemente baja
- Ejemplo: red neuronal MLP
  - Aprendizaje paramétrico
  - Pesos para los w_{ij}
  - Un gen para cada arco
- Genotipo: cjto. de parámetros de un cromosoma particular, útil para construir una solución.
- Fenotipo: solución construida a partir del genotipo.

Condición de terminación

Posibilidades

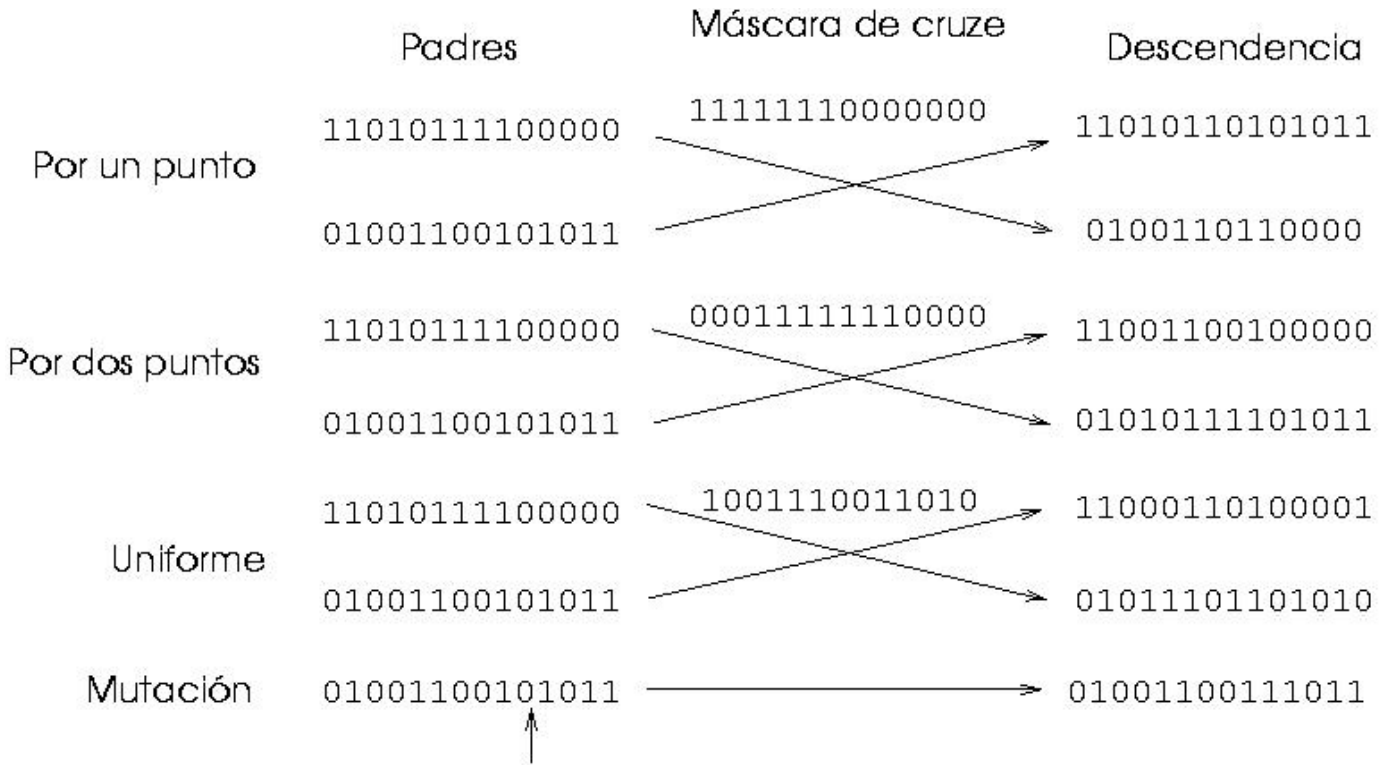
- Número de iteraciones fijo
- El mejor individuo x tiene un nivel de adaptación $f(x) > \alpha$
- Todos los individuos (o un porcentaje alto [1]) son el mismo

Función de adaptación

- Se puede tanto maximizar como minimizar
- Dependiente del problema
- Típicamente RMSE para aprendizaje
- Puede ser multi-objetivo

Operadores

- Operadores genéricos

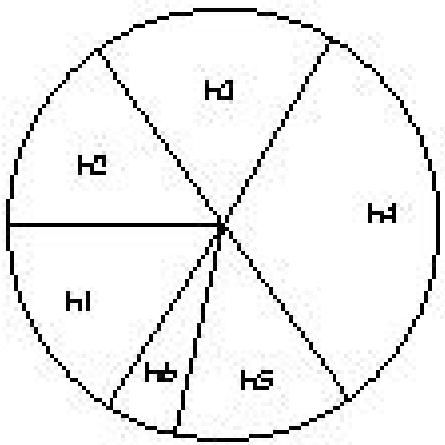


- Operadores específicos (e.g. especialización de reglas)

Selección de la hipótesis más adaptada

- Selección basada en la ruleta

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



es posible que se llegue a la situación en que todos los individuos sean el mismo más facilmente

Otros esquemas de selección

- Selección por torneo
  - Seleccionar h_1 y h_2 con igual probabilidad
  - Escoger la más adaptada con probabilidad p

- Selección por ranking
  - Ordenar todas las h_i según si fitness
  - Escoger con probabilidad proporcional al ranking

Aprendizaje de reglas de decisión [2]

- Aprendizaje conjuntos de reglas de decisión disyuntivas
- El algoritmo usado es exactamente el prototípico
- Antecedentes son conjunciones de tests sobre atributos a_1 y a_2
- Un bit para el consecuente
- Función de fitness, siendo h el conjunto de reglas

$$Fitness = (correct(h))^2$$

- Representación: la regla

$$Si\ a_1 = T \wedge a_2 = F\ Then\ c = T;\ Si\ a_2 = T\ Then\ c = F$$

se representa como

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

Aprendizaje de reglas de decisión (II)

Requerimentos para operadores genéticos

- Conjuntos de reglas de longitud variable
- Cadenas de bits representado hipótesis bien formadas
- Mutación en un bit escogido aleatoriamente

Comenzamos con las hipótesis

	a_1	a_2	c	a_1	a_2	c
h_1:	10	01	1	11	10	0
h_2:	01	11	0	10	01	0

- Sean d_1 la distancia desde el punto de cruce más a la izquierda al límite de regla izquierdo más próximo
- Sean d_2 la distancia desde el punto de cruce más a la derecha al límite de regla izquierdo más próximo
- Escoger puntos de cruce para h_1, e.g. despues de los bits 1 y 8 (por ejemplo)

Entrecruzamiento de cadenas de longitud variable (II)

- Restringimos los puntos de h_2 a aquellos que producen cadenas con una semántica bien definida: aquellos con igual d_1 y d_2 ($< 1,3 >$, $< 1,8 >$ y $< 6,8 >$)
- Con $< 1,3 >$ tenemos

				a_1	a_2	c			
				h_3:	11	10	0		
	a_1	a_2	c	a_1	a_2	c	a_1	a_2	c
h_4:	00	01	1	11	11	0	10	01	0

- Conseguimos conjuntos con numero de reglas diferente al de los padres, siendo bien formadas

Más detalles de GABIL

Añadimos nuevos operadores genéticos, que se aplicarán probabilísticamente

- AddAlternative (AA): generalizar restricción sobre a_i al cambiar un 0 por un 1
- DropCondition (DC): generalizar restricción sobre a_i al cambiar todos los 0's por un 1

También añadimos un bit que determine si aplicar los operadores o no

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

La estrategia evoluciona al mismo tiempo que el proceso de búsqueda

Resultados de GABIL

- El rendimiento de GABIL es comparabla al de algoritmos de inducción de reglas como AQ14, C4.5
- Precisión promedio en 12 problemas tipo
  - GABIL sin operadores AA ni DC: precisión del 92.1%
  - GABIL con operadores AA y DC: precisión del 95.2%
  - Los métodos simbólicos conseguían un rendimiento mínimo de 91.2% y máximo del 96.6%

El teorema del esquema

Responde a la necesidad de justificar teóricamente la convergencia del proceso de búsqueda

- Esquema: cadena conteniendo 0, 1 ó * (cualquiera de los dos)
- Ejemplo de esquema: 10**0*
- Instancias de ese esquema: 101100, 100000, ...
- Se trata de caracterizar a la población estudiando los posibles esquemas que pueden encontrarse en la población

El teorema del esquema (II)

- $m(s, t)$ número de instancias del esquema s en la población en el instante t
- $\overline{f}(t)$ fitness promedio de la población en el instante t
- $\hat{u}(s, t)$ fitness promedio de las intancias de s en el instante t
- La probabilidad de h en un paso de selección

$$Pr(h) = \frac{f(h)}{\sum_{i=0}^{n} f(h_i)} = \frac{f(h)}{n\overline{f}(t)}$$

- La probabilidad de seleccionar un miembro de s en un paso

$$Pr(h \in s) = \sum_{h \in s \wedge p_t} \frac{f(h)}{n\overline{f}(t)} = \frac{\hat{u}(s, t)}{n\overline{f}(t)} m(s, t)$$

- Número de instancias de s después de n selecciones

$$E[m(s, t+1)] = \frac{\hat{u}(s, t)}{\overline{f}(t)} m(s, t)$$

El teorema del esquema (III)

- Por lo tanto, es de esperar que esquemas más potentes sobrevivan en sucesivas generaciones
- Ahora, si consideramos también entrecruzamiento y mutación

$$E[m(s,t+1)] \geq \frac{\hat{u}(s,t)}{\overline{f}(t)} m(s,t) \left(1 - p_c \frac{d(s)}{l-1}\right) (1-p_m)^{o(s)}$$

  - p_c probabilidad de operador de entrecruzamiento *single point*
  - p_m probabilidad de mutación
  - l longitud de las cadenas de bits
  - $o(s)$ número de bits definidos (i.e. no *) en s
  - $d(s)$ distancia entre los bits más a la izquierda y derecha definidos en s

Interpretación

- El término intermedio es la probabilidad de que un individuo del esquema s siga representando al esquema s después de un entrecruzamiento
- El término de la derecha es la probabilidad de que un individudo del esquema s siga representando al esquema s después de la mutación
- Esquemas más adaptados crecen en influencia
- Mejores los esquemas con menos bits definidos y más próximos entre sí

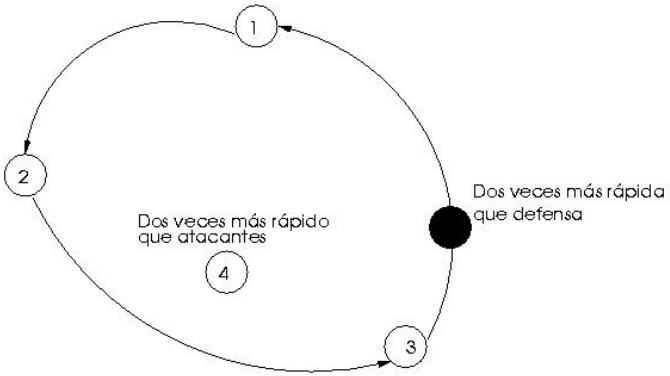
Aplicación de la programación genética al fútbol robótico [3]

- Aprendizaje *por capas* usando programación genética
- Tarea concreta: *keep-away soccer* (i.e. rondo)
- Problema: explosión combinatoria en el espacio de soluciones
- Idea: dividir el aprendizaje en subproblemas que se usan después para recomponer el problema completo
  - Tarea entre dos agentes: pasar la bola ($t1$)
  - $t1$ una vez aprendida se incorpora en tareas de más agentes: proteger la bola, avanzar hacia arriba con la bola, preparar para rematar, etc.
- Individuos: agentes especificados como programas

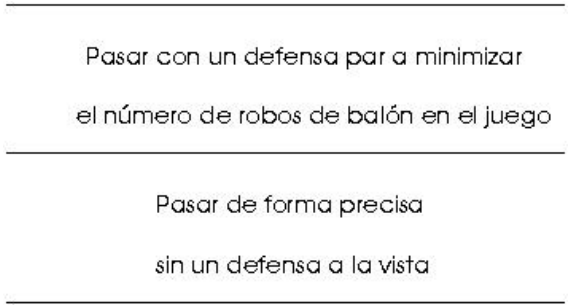
El problema del rondo en fútbol robótico

- Tarea: mantener la pelota fuera del contacto de un defensa, que intentará arrebatársela a varios jugadores atacantes
- Para que tres agentes o más se coordinen adecuadamente, cada uno debe ser capaz de, al tener la pelota,
  - seleccionar un compañero al que pasar
  - temporizar el pase adecuadamente
  - mantener un pasillo de pase abierto

Representación del juego



Comportamientos en capas



Aprendizaje por capas

Aprendizaje por capas	Programación genética
El aprendizaje inductivo simple no es suficiente como estrategia	Aplicaciones MAS complejas deben descomponerse en niveles si queremos aplicar PG
Se tiene una descomposición bottom-up del problema	La tarea de aprendizaje MAS se puede descomponer
El aprendizaje es independiente para cada nivel	PG puede aplicarse como estrategia de aprendizaje a cada nivel por separado
La salida de la capa inferior es entrada de la capa superior	La población en la última generación de una capa es usada como la población inicial de la siguiente capa

Diseño de experimentos de aprendizaje

- Dos experimentos
  - El mejor individuo de la primera capa va replicado a toda la población de la segunda (LLGP-Best)
  - Toda la población evolucionada de la primera capa va a toda la población de la segunda (LLGP-All)
- Capas de aprendizaje
  1. Primera capa
     - ★ Fitness: maximizar el número de pases precisos
     - ★ Individuos: equipos de tres copias del mismo agente jugando en el mismo campo que se realiza el rondo
  2. Segunda capa
     - ★ Fitness: minimizar el número de robos de balón
     - ★ Individuos: equipos de tres agentes iguales y un cuarto oponente igual

En cada paso de simulación, si un agente tiene la bola se evalua el árbol sintáctico de patada y produce un vector (θ, d) para chutar

Si no, se evalua el árbol sintáctico de movimiento

Ambos árboles se componen de terminales y funciones

Elementos terminales

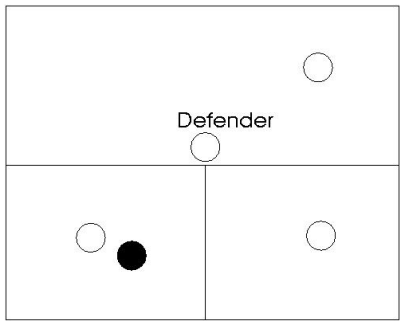
Terminal	Descripción
Defender	Vector a un oponente
Mate1	Vector al primer compañero
Mate2	Vector al segundo compañero
Ball	Vector a la bola

Conjunto de funciones

Función	Descripción
Rotate90(1)	Rotar el vector actual 90^0 en dirección *counter-clockwise*
Random(1)	Nuevo vector aleatorio con magnitud entre 0 y el valor actual
Negate(1)	Negar dirección del vector
Div2(1)	Dividir magnitud del vector por 2
Mult2(1)	Multiplicar magnitud del vector por 2
VAdd(2)	Sumar dos vectores
VSub(2)	Restar dos vectores
IFLTE(4)	*if* $\|\|v_1\\| < \|\|v_2\\|$ *then* v_3 *else* v_4

Más detalles

- El defensor está programado a mano
- Cada simulación de un individuo (i.e. equipo) usa 200 pasos de simulación
- Situación inicial



- Óptimo local: todos los agentes atacantes se agrupaban en torno a la bola para evitar que el defensor la robara $\longrightarrow$ se evitó haciendo que el defensor pudiera *atravesar* a los atacantes


K. DeJong.
The Analysis and behaviour of a Class of Genetic Adaptive Systems.
PhD thesis, University of Michigan, 1975.


K.A. DeJong, W.M. Spears, and D.F. Gordon.
Using genetic algorithms for concept learning.
Machine Learning, 13:161–188, 1993.


W. H. Hsu and S. M. Gustafson.
Genetic programming and multi-agent layered learning by reinforcements.
In *In Genetic and Evolutionary Computation Conference*, New York, July 2002.


Tom M. Mitchell.
Machine Learning.
McGraw-Hill, 1997.