Algoritmos Bioinspirados: Evolución Diferencial

Alberto García y Diego Martínez

5 de abril de 2020

1. Introducción

En este trabajo hemos implementado el algoritmo diferencial desarrollado en el paper de Tian, Gao y Dai[1]. Una de las principales características de este algoritmo es la capacidad de autogestión y adaptabilidad a la hora de elegir entre diversidad y convergencia.

A continuación describimos los mecanismos de mutación y recombinación, pasando por alto detalles más concretas del algoritmo como el cálculo de F_1 o $\vec{d}_{r2,G}$.

1.1. Mutación

Los individuos mutados se generan mediante la siguiente fórmula:

$$\vec{v}_{i,G} = \begin{cases} \vec{x}_{\text{rand},G} + F_1(\vec{x}_{g,G} - \vec{x}_{\text{rand},G}) + F_2(\vec{x}_{r1,G} - \vec{d}_{r2,G}) & \text{si rand} < \xi_1 \\ \vec{x}_{\text{cur},G} + F_1(\vec{x}_{g,G} - \vec{x}_{\text{cur},G}) + F_2(\vec{x}_{r1,G} - \vec{d}_{r2,G}) & \text{caso contrario} \end{cases} \tag{1}$$

Donde las variables tienen el siguiente significado:

- ξ_1: Constante definida por el usuario, ≤ 1.
- rand: Variable aleatoria con probabilidad uniforme entre 0 y 1.
- G: Generación a la que pertenecen los individuos.
- $\vec{v}_{i,G}$: Individuo mutado.
- $\vec{x}_{\text{cur},G}$: Individuo a mutar.
- $F_1,\ F_2$: Constantes determinadas por el fitness.
- $\vec{x}_{\text{rand},G}$: Individuo seleccionado al azar.
- $\vec{x}_{g,G}$: *Guiding individual*. Individuo seleccionado al azar entre los individuos con mejor *fitness*.
- $\vec{d}_{r2,G}$ Vector aleatorio dentro del espacio de búsqueda.

La predisposición hacia la convergencia o la diversidad viene dada por el valor que demos a ξ_1.

El primer término favorece la convergencia: "sustituye" la posición del individuo original por uno aleatorio, forzando que los puntos se mantengan donde está la mayoría. El segundo término favorece la exploración: "mantiene" la posición original y luego le suma dos perturbaciones: una que lo lleva hacia el *guiding individual* y otra aleatoria.

Nótese que no se están redefiniendo las posiciones de la población original, denotada por $\vec{x}$, sino definiendo una nueva población mutada $\vec{v}$.

1.2. Crossover

Mediante el *crossover* generamos individuos que comparen características entre la población inicial y la población mutada. Lo primero que hacemos es definir la probabilidad CR de que un individuo de la siguiente generación herede características del individuo mutado.

$$\text{CR} = 1 - \frac{R_g}{\text{NP}} \tag{2}$$

Con

- NP: Número total de individuos.
- R_g: *Ranking* del *Guiding individual*. Si el *guiding individual* es el mejor individuo $R_g = 1$, si es el segundo mejor $R_g = 2$, etc.

Si nos fijamos en la ec.(1) vemos que los individuos mutados tienen tendencia a acercarse al *guiding individual*. El *crossover* explota esto otorgando una CR alta cuando el *fitness* del *guiding individual* es mejor (R_g pequeño).

Una vez calculada la probabilidad de mutación CR generamos un *trial* a partir de cada par individuo original - individuo mutado. Hacemos que el *trial* $u_{i,G}$ herede componentes de la población mutada en función del CR:

$$u_{i,G}^j = \begin{cases} v_{i,G}^j & \text{si rand} \leq \text{CR ó randn}(i) = j \\ x_{i,G}^j & \text{otro caso} \end{cases} \tag{3}$$

Donde el superíndice j denota la j-ésima componente.

- randn(i): Entero aleatorio entre 1 y D, siendo D el número de componentes de x, v, u.

La condición randn(i) = j asegura que al menos una componente (la componente número rand(i)) sea mutada. Nótese que para problemas de una única dimensión esta condición siempre se cumple y los *trials* son idénticos a la población mutada.

1.3. Selección

En esta parte del algoritmo se elige al individuo de la generación $G + 1$, eligiendo para ello entre el individuo de la generación G y el trial generado a partir de él.

Para efectuar esta selección se define previamente un fitness ponderado:

$$f_w(x_{i,G}) = \alpha \frac{f(x_{i,G}) - f_{\min}}{f_{\max} - f_{\min}} + (1 - \alpha) \frac{\text{Dis}_{\max} - \text{Dis}(x_{i,G}, x_{best,G})}{\text{Dis}_{\max} + \text{Dis}(x_{i,G}, x_{best,G})} \tag{4}$$

Con

- f_w: Función *fitness* ponderada.
- $f, f_{\min}, f_{\max}$: Función *fitness* del problema a resolver; el mínimo y el máximo valor de esta función para la población de la generación G.
- α: Variable aleatoria entre 0.8 y 1.
- Dis, $\text{Dis}_{\max}$: Función que devuelve la distancia euclídea entre dos puntos; la máxima de estas distancias para la generación actual.
- $x_{\text{best},G}$: Individuo con mejor *fitness* en la generación G.

La función fitness ponderada introduce una penalización por alejarse del individuo con mejor *fitness*.

El método de selección es esencialmente comparar ambos los *fitness* de el individuo original y el trial. Si el trial supera al original el individuo es sustituido por el trial.

$$\vec{x}_{i,G+1} = \begin{cases} \vec{u}_{i,G} & \text{si } f(\vec{u}_{i,G}) < f(\vec{x}_{i,G}) \\ \vec{u}_{i,G} & \text{si } f(\vec{u}_{i,G}) \leq f(\vec{x}_{i,G}) \text{ y } x_{i,G} \neq x_{\text{best},G} \\ \vec{x}_{i,G} & \text{otro caso} \end{cases} \tag{5}$$

Hacemos una excepción para el individuo con el mejor *fitness*, $x_{\text{best},G}$, que siempre pasa a la siguiente generación.

2. Estructura del programa

Para escribir el programa hemos modificado bastante el programa inicial dado en clase. El programa se sigue llamando desde `lanzador.R`, que inicializa el problema mediante las funciones del directorio `funciones` y el script `inicia.R` (aunque este ha sido renombrado como `inicializador.R`).

A partir de aquí los scripts se han directamente sustituido o eliminado. Una vez inicializado el problema, `lanzador.R` llama a `evolutivo.R`. Este es el script central del programa, y se encargará de llevar a cabo todos los pasos descritos en el paper[1]. Para ello hará uso de los siguientes scripts:

1. `mutacion.R`: este script recibe como argumento una población de individuos y genera una población mutada, sin modificar la población original.

2. `crossover.R`: este script toma la población original y la mutada mediante `mutacion.R` y las combina generando un individuo *trial*, en base a la variable *CR*, que determina la probabilidad de un individuo de mutar. El *trial* generado es un individuo que puede tener componentes tanto del individuo original como del individuo mutado.

3. `seleccion.R`: este script selecciona los individuos eligiendo entre los individuos originales o los *trials*. Para ello tiene en cuenta el *fitness* de los individuos.

Referencias

[1] M. Tian, X. Gao, and C. Dai, "Differential evolution with improved individual-based parameter setting and selection strategy," *Applied Soft Computing*, vol. 56, pp. 286–297, 2017.