Ilustrando el algoritmo genético

En realidad, nos vamos a referir a una variante de algoritmo evolutivo que se denomina “DE/best/1/bin”, pero en el título queremos destacar el origen de este tipo de algoritmos que buscan solucionar un problema copiando a la naturaleza.

Hay que empezar diciendo que no se trata de una demostración matemática, lo que se pretende es ilustrar lo que hay en el fondo de un algoritmo que en una primera impresión puede parecer mágico, especialmente por sus excelentes resultados.

Dado un problema en el que intervienen un número determinado de variables se trata de encontrar el valor de todas y cada una de ellas que satisface una o varias condiciones, un ejemplo sería resolver un sistema de ecuaciones lineales o no lineales, o encontrar el mínimo de una función de esas variables más o menos compleja. Buscamos un vector, de la dimensión del número de variables (N), cuyos elementos serán los valores de estas que solucionan el problema.

El esquema de trabajo de este algoritmo es el siguiente:

Se empieza creando una población de P individuos (candidatos a solución) al azar, es decir, de P vectores cuyos N elementos (valores de las N variables) se generan al azar, dentro de unos rangos que se prefijan de antemano, para cada elemento. Esta población es la primera generación y es el punto de partida para ir creando sucesivas generaciones.

El algoritmo necesita una forma de asignar un mérito (aptitud) a cualquier individuo, obviamente a partir de los valores del vector que lo representa. Una vez obtenido el mérito de cada individuo podemos seleccionar al que tiene el mejor mérito y lo llamaremos individuo de referencia.

A partir de una generación dada, se crea una nueva generación compuesta por nuevos individuos mediante el siguiente proceso de “Mutación”, “Recombinación” y “Selección”.

1. Mutación: Se crea una población de individuos mutantes, a partir del individuo de referencia, aplicando la fórmula 1. El uso del 'Mejor individuo' es lo que determina la denominación '**best**' en 'DE/**best**/1/bin'. La inclusión del '**1**' en la denominación 'DE/best/**1**/bin' hace referencia a que sólo hay una resta. Como puede verse, el individuo mutante en la posición 'n' de la población es totalmente independiente del individuo original en esa misma posición, la relación con el individuo original se produce en la recombinación. En este punto, la nueva población está compuesta por mutantes del “Mejor individuo” que actuará como primer progenitor de todos los hijos que surgirán en el paso siguiente: la recombinación (una práctica habitual en la naturaleza).

2. Recombinación o cruce: A continuación, se procede a la recombinación (crossover) de los individuos mutantes con los de la población original (que podemos considerar que actúan como segundo progenitor). Para cada individuo ‘n’ de la población, se construye un vector de prueba (hijo) rellenándolo secuencialmente con valores del vector 'Mutante' (primer progenitor) o del vector del individuo original (segundo progenitor). La decisión de usar el valor del 'Mutante', o del individuo original, se toma mediante una distribución binomial (de aquí el término '**bin**' en 'DE/best/1/**bin**'), para ello, se genera un número aleatorio en [0, 1), si este número es menor que un determinado parámetro, que se denomina constante de recombinación, el parámetro se toma de 'Mutante', de lo contrario, se toma desde el individuo original. Se exige que siempre se incluya, como mínimo, un valor del vector 'Mutante' (seleccionado aleatoriamente), para evitar que la nueva solución a probar sea una copia exacta de la original.

3. Selección: Una vez construido el candidato a solución, se evalúa su mérito. Si el candidato a solución (vector de prueba) es mejor que el individuo original, lo reemplaza. Si es también mejor que el individuo de referencia (el mejor hasta ese momento) lo sustituye, pasa a ser el individuo de referencia.

$$Mutante = Mejor\ individuo + mutación * (Individuo_{azar\ 1} - Individuo_{azar\ 2})$$

Fórmula 1. Generación de un individuo mutante mediante la estrategia “DE/best/1/bin”

Una vez que hemos definido el algoritmo vamos a ilustrar porque esta es una forma sorprendentemente útil de llegar a una solución.

Vamos a partir de un problema muy simple, incluso trivial. Solo tendremos dos variables, para poder representar a los individuos como un punto en un plano, y queremos encontrar los valores del vector (**x**,**y**) que satisfacen una condición obvia para un humano, debe cumplirse que:

x = 6

y = 4

Pero no olvidemos que se la vamos a plantear a un algoritmo que es una máquina que no piensa (por mucho que en la actualidad se nos quiera convencer de que las hay que si lo hacen).

El primer algoritmo que vamos a probar es también el más sencillo que podemos imaginar, el algoritmo de “Monte Carlo” puro y duro. Vamos a generar puntos al azar y comprobar si cumplen la condición de nuestro problema. Para generar puntos al azar lo primero que necesitamos es establecer cuál es el rango en el

que pueden variar la **x** y la **y**. Este rango lo vamos a fijar en el intervalo [0,10] para ambas variables.

El algoritmo empieza a generar números al azar, uno tras otro, con la esperanza de que uno cumpla que **x** = 6 e **y** = 4.

Como no queremos alargar mucho los cálculos vamos a poner un límite al número de puntos a generar que será de 2464.

En la figura 1 se muestra cómo va cambiando (o no) el mejor punto obtenido según vamos generando los puntos, solo mostramos un gráfico cada 32 intentos, si el punto es verde es que no ha mejorado respecto del gráfico anterior y rojo si lo ha hecho.

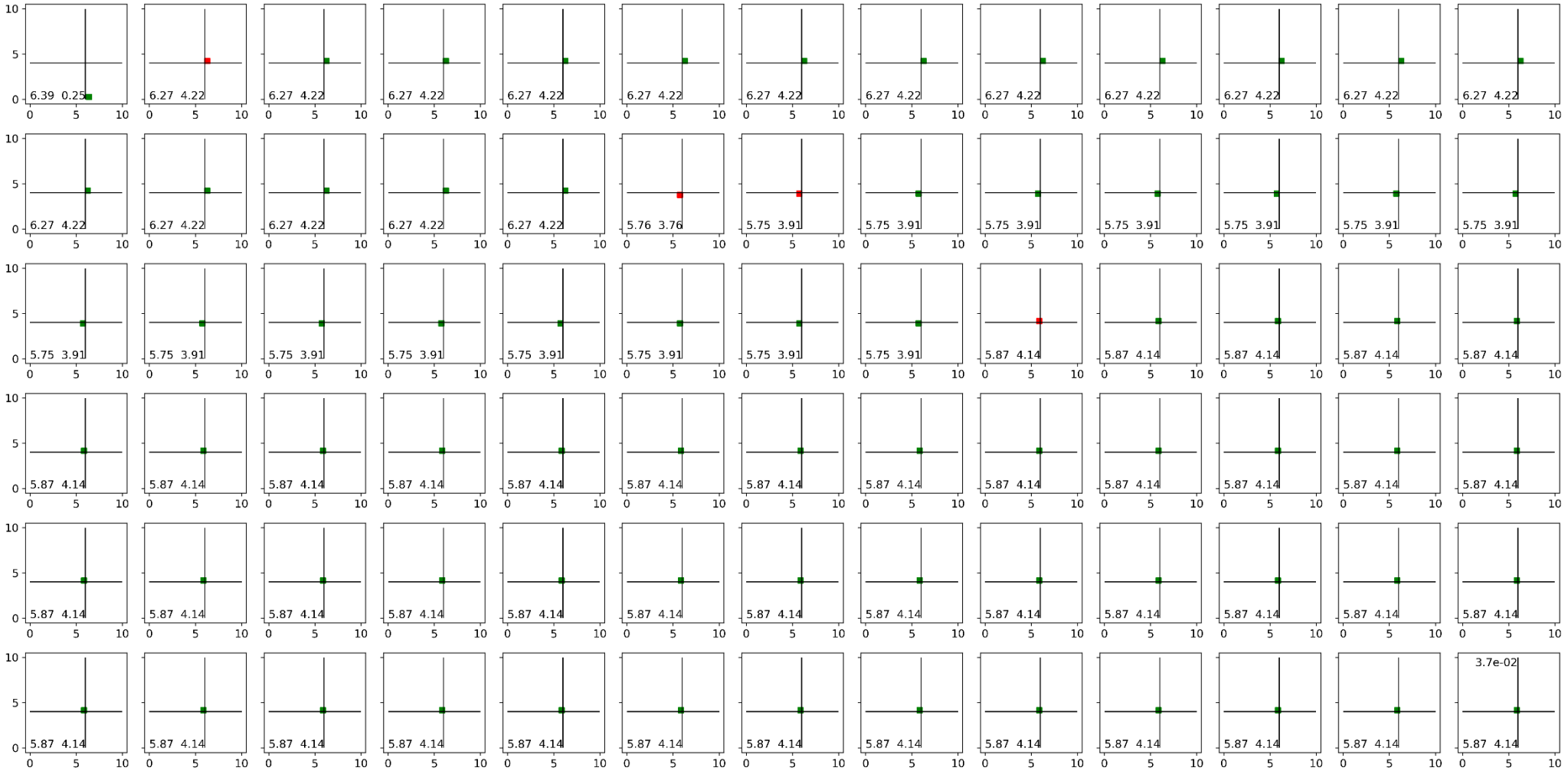

Figura 1. Puntos (uno de cada 32) generados al azar en el rango [0,10] tanto para la **x** como para la **y**.

En la figura 1 vemos que lo más que nos hemos aproximado a la solución (6, 4) es:

x = 5,87

y = 4,14

Que para ser al tuntún no está del todo mal, pero este primer algoritmo es demasiado burdo y hay una mejora evidente y muy fácil de implementar que, además, nos ofrece un nuevo punto de vista para que este ejercicio no nos parezca una absoluta tontería.

Una vez que generamos el primer punto, el siguiente lo generamos en las proximidades de este y si el punto generado se encuentra más cerca de la solución, nos quedamos con él, en caso contrario lo descartamos y probamos a generar otro, repitiendo sucesivamente el procedimiento. Esto nos permite interpretar nuestro ejercicio como la búsqueda de un camino desde un punto inicial al azar, hasta el punto solución, le damos un propósito a nuestra máquina que como vamos a ver (figura 2) muestra atisbos de inteligencia.

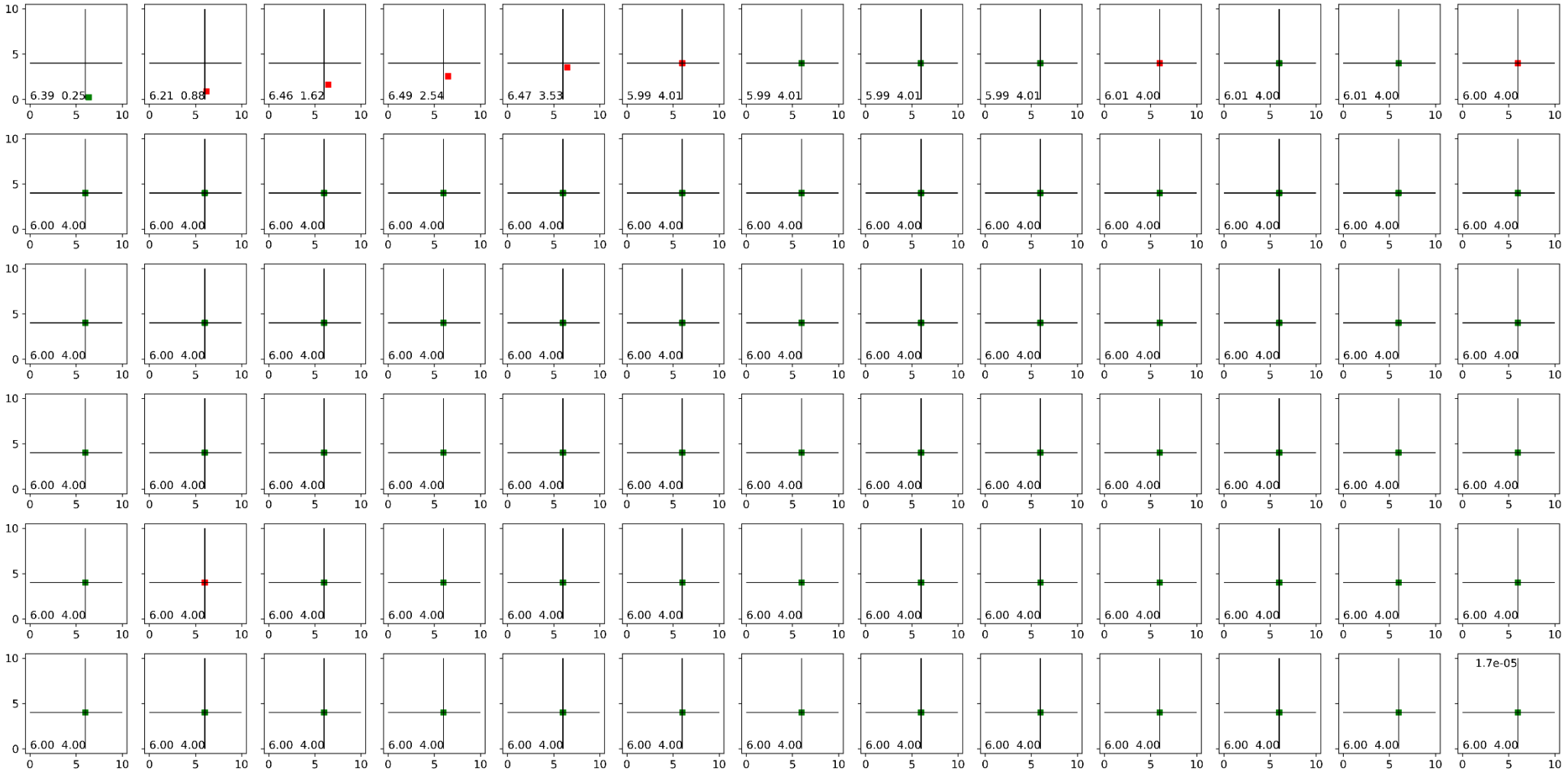

Figura 2. Puntos (uno de cada 32) generados al azar de forma sucesiva (cada uno en las proximidades del anterior).

La figura 2 nos muestra que en último gráfico de la primera fila el algoritmo ha llegado al punto de destino.

Aunque es fácil imaginar lo que significa generar un punto en las proximidades de otro vamos a formularlo (fórmula 2) para que no queden dudas y para ver que no nos hemos olvidado del algoritmo genético.

$$Punto = Punto\ anterior + factor * número\ al\ azar$$

Fórmula 2. Monte Carlo sucesivo.

Si en la fórmula 2, el $número\ al\ azar$ es un número perteneciente al rango de variación de la variable y hacemos que $factor$ sea inferior a 1 el nuevo punto estará cerca del anterior, más cuanto menor sea $factor$.

Las fórmulas 1 y 2 presentan una cierta semejanza, producen un nuevo individuo a partir del mejor hasta ese momento. $factor$ se correspondería con $mutación$ y la diferencia es que en el algoritmo genético el "desplazamiento" es el resultado de restar los vectores que corresponden a dos individuos elegidos al azar y aquí es un vector obtenido directamente al azar. Si consideramos que en el algoritmo genético todos los individuos se van aproximando a la solución (solo los cambiamos si su mérito aumenta) es razonable pensar que cada vez se parecerán más entre si y por tanto la resta de cualquier par de ellos irá haciéndose más pequeña según avanzan las generaciones. Esto nos da una pista para nuestro Monte Carlo sucesivo, a la que también podríamos haber llegado sin mucho esfuerzo: según nos vamos aproximando al punto de destino conviene ir reduciendo el $factor$ para que los desplazamientos sean más pequeños y no nos saltemos el punto de destino. Esto, dicho sea de paso, es lo que haría un humano mínimamente sensato.

Con esta nueva mejora (Monte Carlo sucesivo de factor decreciente) el camino de nuestra máquina es el de la figura 3.

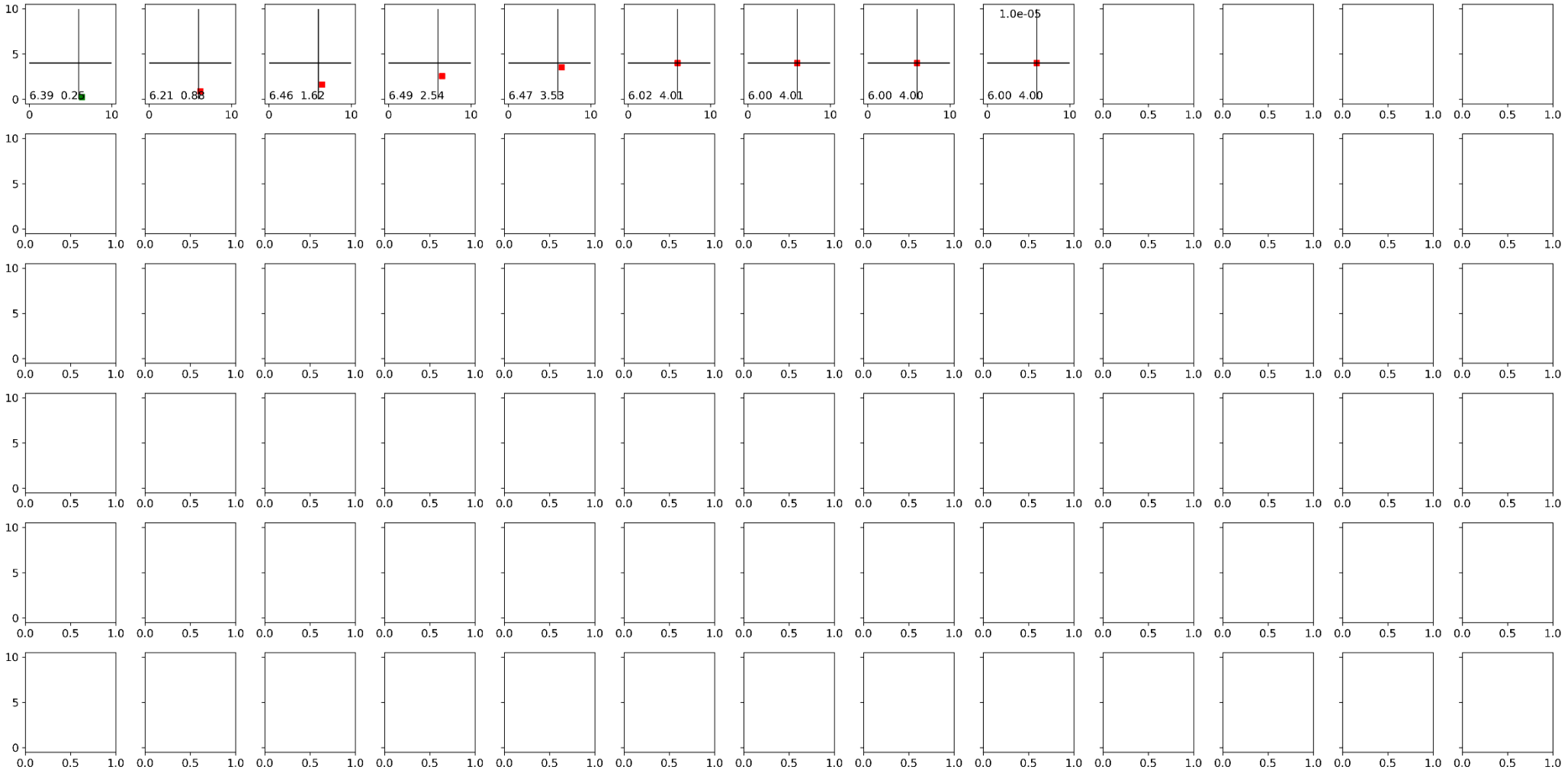

Figura 3. Monte Carlo sucesivo de paso variable.

El Monte Carlo sucesivo de paso variable no podía mejorar el de su predecesor, que ya era exacto, pero ha podido alcanzarlo en menos tiempo.

Como vamos a ver y podíamos imaginar, el algoritmo genético también encuentra la solución. Para aplicarlo hemos establecido una población de 16 individuos y los resultados se resumen en la figura 4.

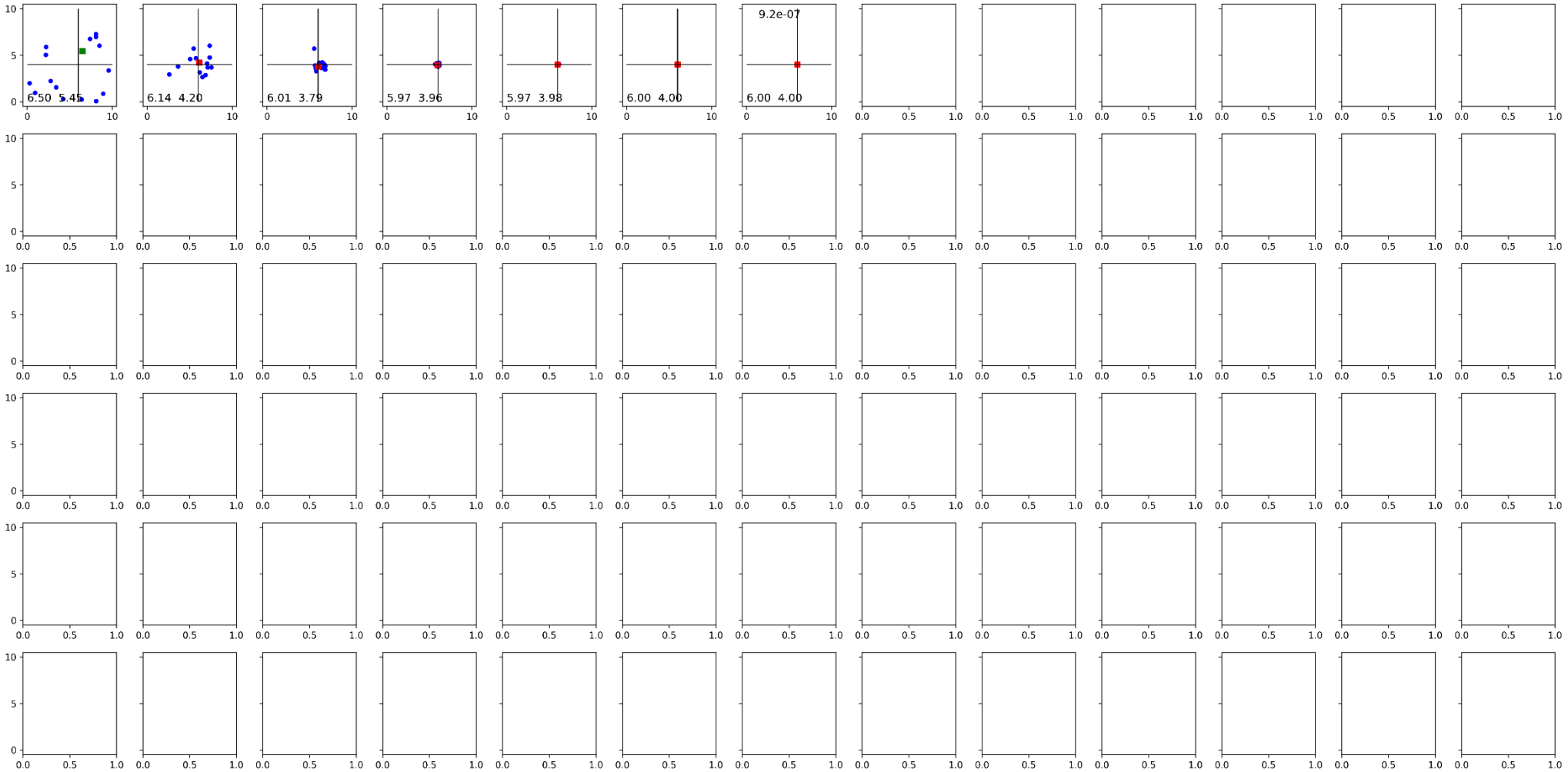

Figura 4. Algoritmo genético

La figura 4 muestra como los individuos de la población (puntos azules), inicialmente generados al azar, se van acercando a la solución a cada generación y con ello disminuyendo la distancia entre ellos, lo que conlleva la consiguiente reducción de los “desplazamientos” (mutaciones) que se calcularán para obtener la generación siguiente.

En la tabla 1 se resumen los resultados obtenidos. Para que el número de generaciones sea equivalente, por cada generación del algoritmo genético se conceden 16 intentos a Monte Carlo porque el primero evoluciona 16 individuos y el segundo 1. Dado que por merito estamos considerando la distancia (geométrica) a la solución, el merito es mayor cuanto menor es su valor.

algoritmo	generaciones	merito	6,000	4,000
genetico	12	0	5,999	3,999
mc simple	2464	0,03668	5,869	4,140
mc sucesivo	2464	0,00002	6,004	4,001
mc modulado	241	0,00001	6,002	4,002

Tabla 1. Resultados de los distintos algoritmos utilizados.

Llegados a este punto podemos declarar que el algoritmo genético, a pesar de que se ha presentado con el romántico ropaje de la evolución de las especies, no es más que una versión, no muy sofisticada, del método de Monte Carlo. No obstante, aporta una ventaja que lo convierte en una herramienta verdaderamente útil.

Para apreciar esta ventaja vamos a realizar este ejercicio con vectores de 6 variables, y no de 2 como hasta ahora, y siendo el punto desino el siguiente:

(6, 35, 16, 90, 200, 40)

Los rangos de variación de las 6 variables son los de la tabla 2:

	rango	
variable	inferior	superior
1	0	10
2	30	50
3	5	100
4	10	200
5	0	500
6	0	1000

Tabla 2. Rangos de variación de las variables.

La tabla 3 resume los resultados de un primer intento:

algoritmo	generaciones	merito	6	35	16	90	200	40
genetico	154	0,49	6,68	34,99	16,04	90,03	200,13	40,03
mc simple	2464	1325,11	9,18	34,57	17,46	122,04	183,30	37,36
mc sucesivo	2464	8,44	8,09	33,15	15,28	90,25	199,89	40,21
mc modulado	2464	8,44	8,09	33,15	15,28	90,25	199,89	40,21

Tabla 3. Resultados del ejercicio con 6 variables y población de 16 individuos en el algoritmo genético.

Dado que ni siquiera el algoritmo genético alcanza la solución exacta, vamos a ampliar la población inicial de 16 individuos a 28 y el número de intentos de los Monte Carlo en la misma proporción. Los resultados pasan a ser los de la tabla 4.

algoritmo	generaciones	merito	6	35	16	90	200	40
genetico	38	0,00	6,00	35,00	16,00	90,00	200,00	40,00
mc simple	4312	1325,11	9,18	34,57	17,46	122,04	183,30	37,36
mc sucesivo	4312	7,96	8,01	33,14	15,81	90,14	200,34	39,43
mc modulado	4312	7,96	8,01	33,14	15,81	90,14	200,34	39,43

Tabla 4. Resultados del ejercicio con 6 variables y población de 28 individuos en el algoritmo genético.

El algoritmo genético encuentra rápidamente la solución, pero Monte Carlo no lo consigue y además reducir el valor del $factor$ (mc modulado) no aporta nada respecto a no hacerlo (mc sucesivo).

Esto nos revela que la gran aportación del algoritmo genético es la modulación del $factor$.

Para empezar, en Monte Carlo es un parámetro al que debemos darle un valor a priori, al menos inicialmente y en el algoritmo genético no hay que hacerlo, en segundo lugar y más importante en Monte Carlo el $factor$ es el mismo para todas las variables mientras que en el genético no. Diferenciar el $factor$ por variables en Monte Carlo es impensable para un número mediano, o grande, de variables. A esto hay que sumar que, aunque el valor del $factor$ se diferencie por variables también hay que diferenciar su evolución. El algoritmo genético no lo da todo hecho a través de la resta de dos individuos elegidos al azar.

En la figura 5 se muestra como ha ido variando, a lo largo de las generaciones, el $factor$ de cada una de las 6 variables, obtenido de las restas de vectores entre los pares de individuos seleccionados al azar.

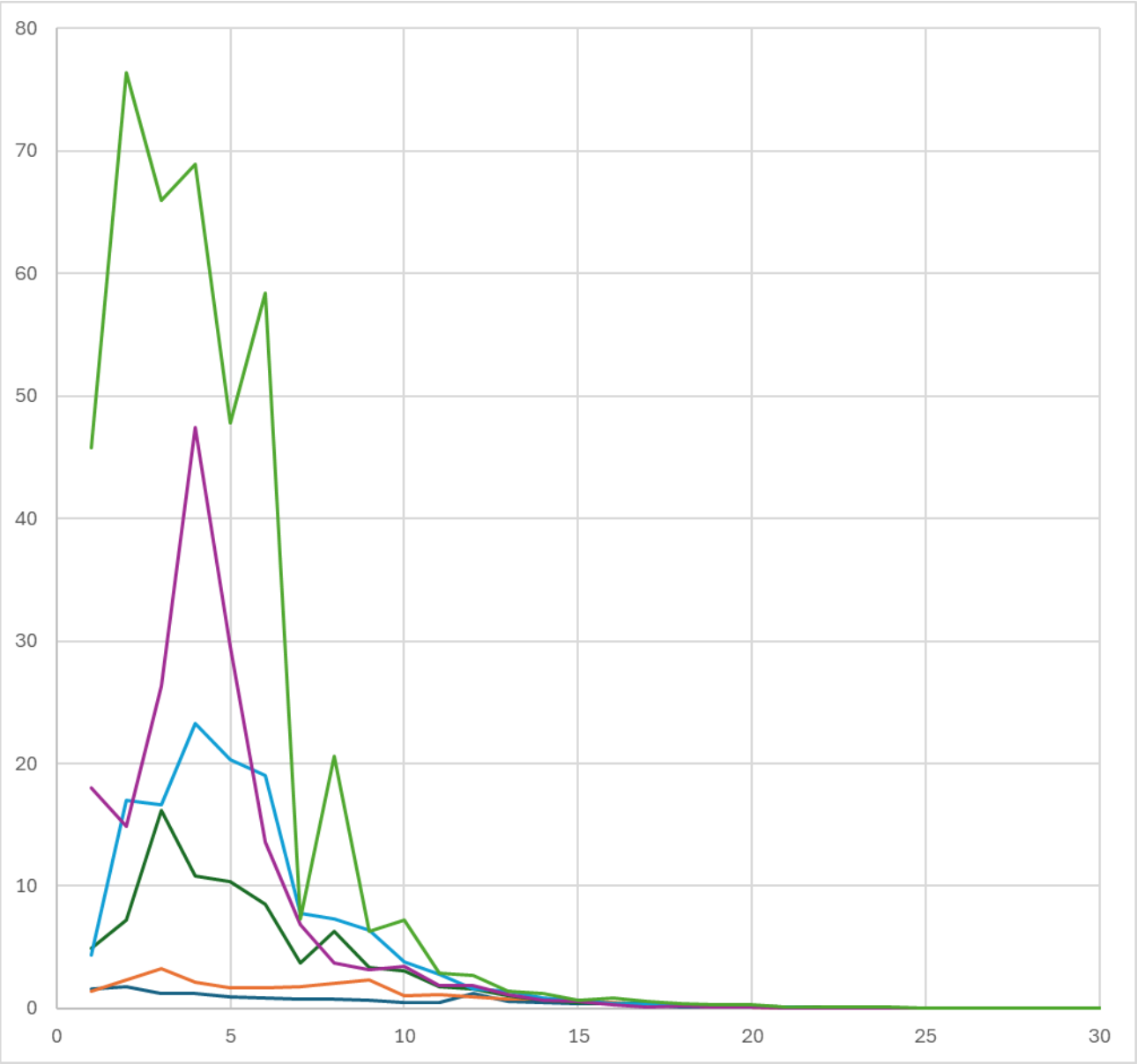

Figura 5. Variación del $factor$ equivalente en el algoritmo genético.

Para destacar el papel de la evolución diferenciada del $factor$ por variables hemos aplicado el Monte Calo de $factor$ variable (mc modulado) utilizando los factores representados en la figura 5. Los resultados los vemos en la tabla 5.

algoritmo	generaciones	merito	6	35	16	90	200	40
genetico	38	0,00	6,00	35,00	16,00	90,00	200,00	40,00
mc simple	4312	1325,11	9,18	34,57	17,46	122,04	183,30	37,36
mc sucesivo	4312	7,96	8,01	33,14	15,81	90,14	200,34	39,43
mc modulado	4312	0,53	5,77	35,69	16,00	89,98	200,01	39,98

Tabla 5. Resultados del ejercicio con 6 variables y población de 28 individuos, usando en el "mc modulado" los $factores$ medios resultantes del algoritmo genético.

El "mc modulado" mejora significativamente y si no alcanza la solución exacta es porque los $factores$ con los que le estamos alimentando son los valores medios de cada generación del algoritmo genético (la media de 16 restas).

No es raro encontrar en la bibliografía al algoritmo genético como parte de la inteligencia artificial. En los últimos tiempos nos estamos asombrando de innovaciones menos importantes que otras que se han producido anteriormente haciendo mucho menos ruido, pero que han cambiado la vida de las personas más de lo que, en mi opinión, cambiará la que alguien ha denominado, con mucho acierto, inteligencia fingida.