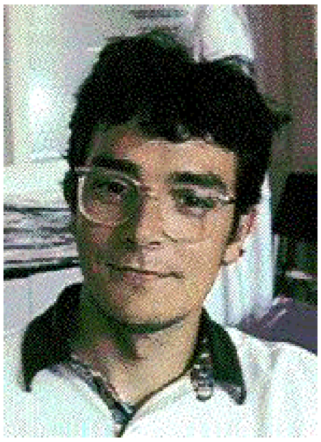
Algoritmos genéticos y SUGAL



José María Cañas Plaza

email:jmplaza@iai.csic.es

29 de Agosto de 1997.

I. INTRODUCCIÓN.

La familia de **algoritmos genéticos** presenta un modo de abordar problemas inspirado en los mecanismos que descubrió Darwin en la evolución de las especies biológicas. Así hablando de algoritmos genéticos enseguida surgen palabras como selección natural, población, adaptación, mutación, genes, etc tan comunes en biología. Típicamente se aplican en problemas de optimización y de minimización de funciones.

Básicamente estos algoritmos plantean el problema de la optimización como un proceso *iterativo* de búsqueda de la mejor solución dentro del espacio de posibles respuestas al problema. Comienza con un conjunto aleatorio de soluciones tentativas, que constituye la *población inicial*. Combinando las mejores respuestas de ese conjunto se crea un nuevo conjunto que reemplaza al anterior, y que es la generación siguiente. Así repetidamente. Con cada nueva iteración la población se va refinando más, consiguiendo mejores respuestas al problema, convergiendo finalmente la mayoría de los individuos en un entorno de la solución óptima.

En el segundo apartado haremos una primera introducción genérica sobre los algoritmos genéticos (a partir de ahora AG), explicando su funcionamiento básico y algunas de las variables más notables. En el tercer capílo presentaremos el paquete de software SUGAL, que es una herramienta de ayuda en la experimentación con AG y explicaremos los parámetros principales que ofrece SUGAL para modular adecuadamente el comportamiento del AG. En el cuarto apartado abordaremos algunas cuestiones prácticas sobre AG como la convergencia prematura, los mínimos locales, etc, y como éstas aparecen en diversos ejemplos con SUGAL.

II. ALGORITMOS GENÉTICOS.

Algoritmo génetico básico.

Para aplicar un AG a un problema se necesita identificar el espacio de posibles soluciones, de respuestas al problema, y una **codificación** de los puntos de ese espacio en un determinado formato. Por ejemplo supongamos que tenemos un sistema controlado por 3 parámetros (a, b y c) y queremos saber el valor conjunto de esos parámetros que hace que el sistema funcione del mejor modo posible.

En este caso el espacio de soluciones será el tridimensional con todos los posibles valores del trío (a-b-c), i.e. R3. Cada punto de este espacio es una posible respuesta al problema y su codificación podría ser la cadena de bits que representa a los tres números reales en el ordenador. A esta codificación se le llama cadena cromosómica,**cromosoma** o **genotipo**, y a sus diferentes partes, **genes**. En nuestro caso tendríamos tres genes.

El AG se puede ver como un algoritmo de búsqueda dentro del espacio de las posibles soluciones al problema planteado. Comienza con una **población inicial**, que es un conjunto aleatorio de respuestas tentativas al problema, codificadas. En nuestro caso sería un grupo de tríos con valores concretos de a-b-c. Cada trío es un individuo de esta generación, que es la primera de todo el proceso evolutivo.

En el siguiente paso se calcula la **función de salud** (función de fuerza, "fitness") para cada individuo de la población, que indica el grado en que cada solución tentativa es adecuada como solución al problema. En nuestro ejemplo diremos que un trío tiene mucha salud, es bueno, en la medida en que hace que el sistema se comporte adecuadamente. En biología sería el grado de adaptación del individuo a su entorno.

A continuación se seleccionan los individuos con mayor salud para que engendren la siguiente generación. Este selecto grupo constituyen los padres de la nueva población. Aquellos puntos con muy baja adaptación, esto es, los que son claramente malos como solución al problema desaparecen al no tener descendencia. Sólo las soluciones más óptimas se reproducen y sus genes pasan de una generación a otra.

Combinando las soluciones tentativas del grupo de padres se generan nuevas soluciones tentativas, los hijos. La creación de estos nuevos puntos se lleva a cabo mediante los operadores genéticos, que principalmente son dos:

- **Cruce** Se elige un punto dentro de los cromosomas del padre 1 y del padre 2, y por ahí se cortan sus cromosomas. La cadena cromosómica del hijo se compone de un trozo de la del padre 1 y de la parte complementaria en el padre 2. El hijo es una nueva combinación de los genes que ya existían en los padres, y en cierta medida mantiene cierta similitud con sus progenitores, pero es un punto enteramente nuevo en el espacio de posibles soluciones. En nuestro ejemplo de los padres 23-34-58 y 57-21-44, podrían salir los hijos 23-24-44 y 57-21-58 si el punto de corte está entre el segundo y tercer gen.

- **Mutación** Provoca un cambio aleatorio en alguno de los genes del hijo, introduciendo así nuevos genes en la población, a través de este individuo. Estos cambios serán favorables si aumentan la salud del individuo que los lleva, es decir, si le hacen ser una solución mejor. Si los cambios son perjudiciales, esos genes no se propagarán de una generación a otra porque los individuos que los portan se extinguirán; al ser peores soluciones nunca serán elegidos como padres y morirán. Por ejemplo el individuo 57-21-58 podría mutar a 57-29-58.

Para terminar este ciclo la nueva generación *reemplaza* a la anterior. Puesto que se seleccionan como padres con mayor probabilidad los puntos más óptimos y se desechan los de peor comportamiento, la nueva generación tenderá a tener mejores genes y por ello tendrá soluciones más óptimas al problema planteado. *Iterando* este proceso repetidamente se obtienen sucesivas poblaciones, i.e. conjuntos de soluciones, cada vez mejores. Se dice entonces que las poblaciones se desplazan, en el espacio de posibles respuestas, hacia la solución óptima.

Finalmente la evolución *converge* situándose las poblaciones en entornos muy pequeños de ese punto óptimo. Por lo tanto el AG converge en el punto óptimo, encontrando así la mejor solución al problema que se quería resolver.

AG como técnica de optimización.

El problema de la optimización se puede plantear como un problema de búsqueda del máximo de una función de bondad, o búsqueda del mínimo de una función de coste, en el espacio de posibles soluciones. El problema de la minimización está muy estudiado, y en ciertos dominios eficientemente resuelto. Existe un conjunto de técnicas que resuelven esta búsqueda en determinados dominios. Por ejemplo si podemos plantear segundas derivadas en el dominio podremos aplicar la **técnica de descenso del gradiente**. Otros algoritmos como el recocido simulado (**"simulated annealing"**) también están a nuestra disposición, y por supuesto siempre tenemos la **búsqueda exhaustiva** como referencia, o determinadas **búsquedas ayudadas por heurísticos**.

Los AG emergen también como una técnica de búsqueda con sus ventajas y sus inconvenientes. Para encontrar el máximo de una función f con AG basta encontrar una codificación aplicable para todos los puntos del dominio y asignar como función de salud la propia f. Como hemos visto el AG desemboca en el punto donde la función de salud es mayor, que en este caso es precisamente el máximo de f. De modo idéntico se resuelve la minimización con AG, por ejemplo asignando como "fitness" $-f$ (el máximo de $-f$ es el mínimo de f).

A diferencia de otras técnicas, como el descenso de gradiente, los AG prueban con un conjunto de puntos en cada iteración, no con uno solo. Es decir tienen un búsqueda en paralelo lo cual le permite cierta robustez frente a mínimos locales. Otra de las ventajas de los AG es su rango de aplicación. Puesto que no exigen cálculo de derivadas, únicamente evaluar la función, se puede aplicar en dominios no numéricos, definiendo los operadores genéticos adecuados.

Como principal desventaja de los AG se les achaca que son lentos y necesitan evaluar la función muchas veces. Si ese cálculo es muy costoso, los AG no son muy recomendables porque consumen mucho procesador y serán muy lentos. También se les critica que a veces es muy difícil conseguir esa función de salud que les sirva de guía en su búsqueda.

III. EL PAQUETE SUGAL.

SUGAL es una librería de código C para experimentar con AG. Ha sido diseñada en la universidad de Sunderland por D.Andrew Hunter y es de libre distribución, con lo que se puede obtener por internet y utilizar sin pagar nada.(http://www.trajan-software.demon.co.uk/sugal.htm)

Esta herramienta permite implementar y probar con muchas variantes de AG, ofreciendo al usuario muchos parámetros de configuración que modulan el comportamiento del AG. En esta sección veremos el algoritmo que sigue SUGAL, sus principales parámetros y cómo se utiliza.

Para usar SUGAL:

1. Primero hay que especificar sus parámetros de funcionamiento en un **fichero de configuración**. Con ellos se determina por ejemplo cuantos individuos queremos en cada población, cuantas generaciones nacerán, que tipo de datos encapsulan los individuos, etc.

2. En segundo lugar hay que proporcionar al programa la **función de salud**, de la cual el AG tratará de encontrar el máximo. El usuario debe codificar una función en lenguaje C, que tome como argumento un individuo y devuelva el valor de su salud.

3. Finalmente se llama a la **función de inicio** de SUGAL. SUGAL leerá el fichero de configuración, creará la primera población y comenzará la evolución siguiendo las pautas del AG. Una vez finalizada la evolución SUGAL vuelca sus resultados en otro fichero.

En el programa `ejemplo1.c` se puede ver una muestra de cómo el usuario utiliza SUGAL. La función `evaluate` calcula el valor de salud (`fitness`) para cada individuo (`chrom`). La función `SuRun` carga el fichero de configuración `ejemplo1_cfg` y pasa el control al AG de SUGAL.

En cuanto al fichero de configuración `ejemplo1_cfg` los parámetros aparecen agrupados alfabéticamente tras el epígrafe `# Sugal v2.1 Configuration`. Entre los más básicos destacaremos el parámetro **population** que determina cuantos individuos componen cada población, y **generations** que marca el número de generaciones que SUGAL creará. Una vez alcanzado el punto final de la evolución SUGAL escribirá sus resultados en el fichero que indica el parámetro **output_file**. Así con el `ejemplo1_cfg` las poblaciones constan de 30 individuos y SUGAL simula la evolución hasta 100 generaciones, volcando sus resultados en el fichero `ejemplo1_out`.

Codificación de los individuos.

Una cuestión importante en la configuración de SUGAL es la especificación de cómo son los individuos, cuántos y de qué tipo son sus genes. Esto depende totalmente del problema al que vayamos a aplicar el AG. En el ejemplo del sistema con tres parámetros de control cada individuo es un trío de números reales. Conviene no olvidar que cada individuo del AG es una solución tentativa al problema planteado. La codificación debe ser lo suficientemente rica como para contemplar todo el espacio de posibles soluciones, o al menos un rango muy amplio de él puesto que en ese rango y no más allá buscará el AG.

El tipo de los genes determina la naturaleza de los operadores genéticos aplicables y su significado. Por ejemplo si un gen es un número real existen operadores de mutación que le suman una cantidad aleatoria. Para genes consistentes en un único bit, el operador de mutación clásico consiste en invertir su valor con una cierta probabilidad. Los operadores genéticos se aplican sobre genes, no sobre bits directamente aunque al final todos los individuos se representen como cadenas de ceros y unos (después de todo trabajamos con un ordenador).

SUGAL plantea el individuo o genotipo como un conjunto de genes, ofreciendo un amplio abanico de tipos para cada gen. Aunque permite el uso de individuos heterogénicos, es decir con genes de distintos tipos, ofrece más funcionalidad con individuos homogénicos, esto es, los que tienen todos sus genes del mismo tipo. En estos últimos nos centraremos en este trabajo.

Dentro de SUGAL los genes de un individuo pueden ser bits, enteros, enteros sin signo, números reales o símbolos, según el parámetro de configuración **datatype** valga *[bit, integer, unsigned_integer, real, symbol ...]* respectivamente. Cada tipo lleva asociado un rango y unos operadores genéticos aplicables. Con el tipo binario cada individuo es una colección de bits y los operadores genéticos se aplican directamente sobre los bits, puesto que los bits son en este caso los genes. Con un tipo numérico los individuos son ristras de números y los operadores se aplican sobre ellos. En el caso de los símbolos es el propio usuario quien define el alfabeto en el que puede variar el tipo.

El número de genes que componen un individuo se determina con el parámetro **length**. Por ejemplo para un problema donde los individuos fueran quintetos de enteros sin signo en el fichero de configuración aparecería:

```
datatype unsigned_integer
lenght   5
```

En el `ejemplo1_cfg` se ha codificado cada individuo como 16 genes binarios. Por lo tanto el espacio en el cual buscará SUGAL va desde el 0000h hasta el FFFFh.

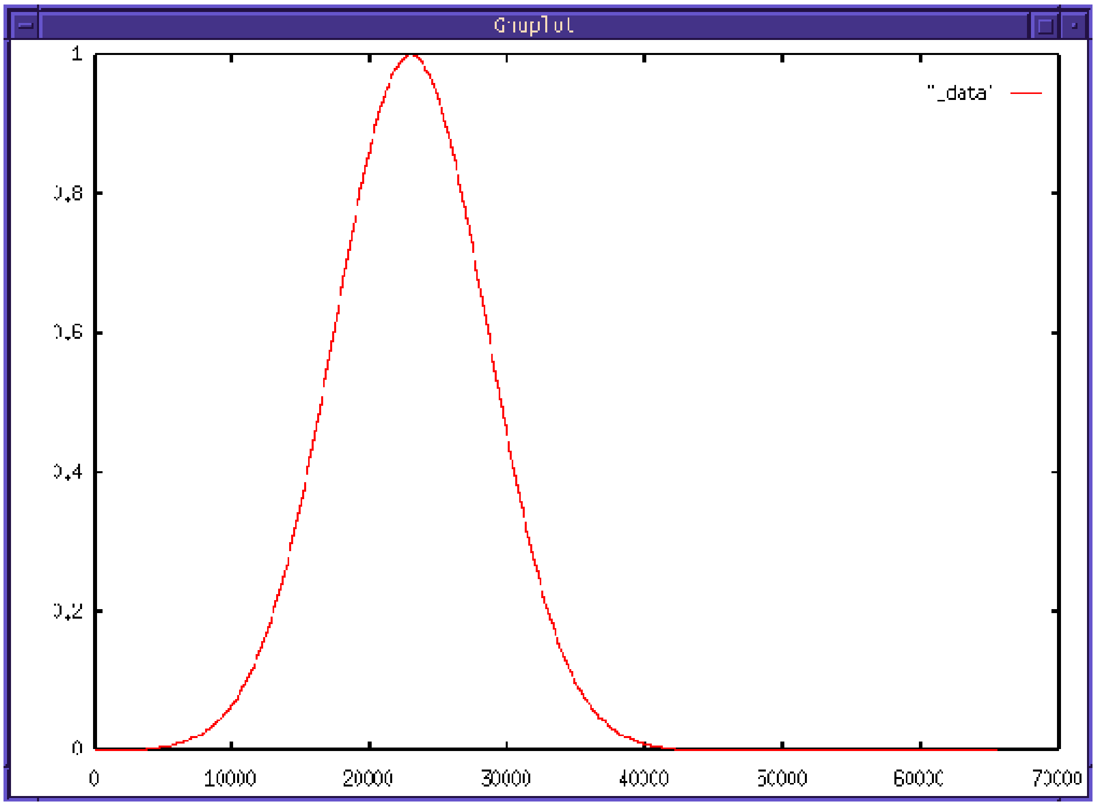
Función de salud.

Esta función tiene como entrada un individuo codificado y como salida su valor de salud correspondiente. SUGAL necesita evaluar la función en cada punto que prueba y requiere que el usuario programe esa función que quiere maximizar. Para poder codificar sin problemas esta función SUGAL ofrece una interfaz de acceso a los distintos genes de cada individuo.

Esta función debe llamarse `evaluate` y su interfaz E/S está perfectamente especificado: `int evaluate(SuChromosome *chrom, double *fitness)`, donde `chrom` es un puntero a un individuo y `fitness` un puntero al resultado que se entrega como double. En el programa `ejemplo1.c` la función `evaluate` primero recompone un valor numérico desde los bits y le aplica la función gaussiana de media 23000 y desviación 10000.

Como los AG buscan el máximo de la función de salud se pueden emplear para encontrar máximos de funciones. Para ello las poblaciones del AG se distribuyen en el dominio de la función a maximizar y ésta misma se emplea como medida de la salud de cada individuo. Así el programa `ejemplo1.c` llama a SUGAL tratando de maximizar precisamente esa función gaussiana.

A diferencia de otros métodos de minimización-maximización, los AG no necesitan del cálculo de gradientes, ni derivadas en modo alguno, sólo requieren poder evaluar la función en cada punto. Esto es una gran ventaja porque abre la aplicabilidad de AG incluso a espacios donde no se puede hablar de derivada, por ejemplo espacios no numéricos.



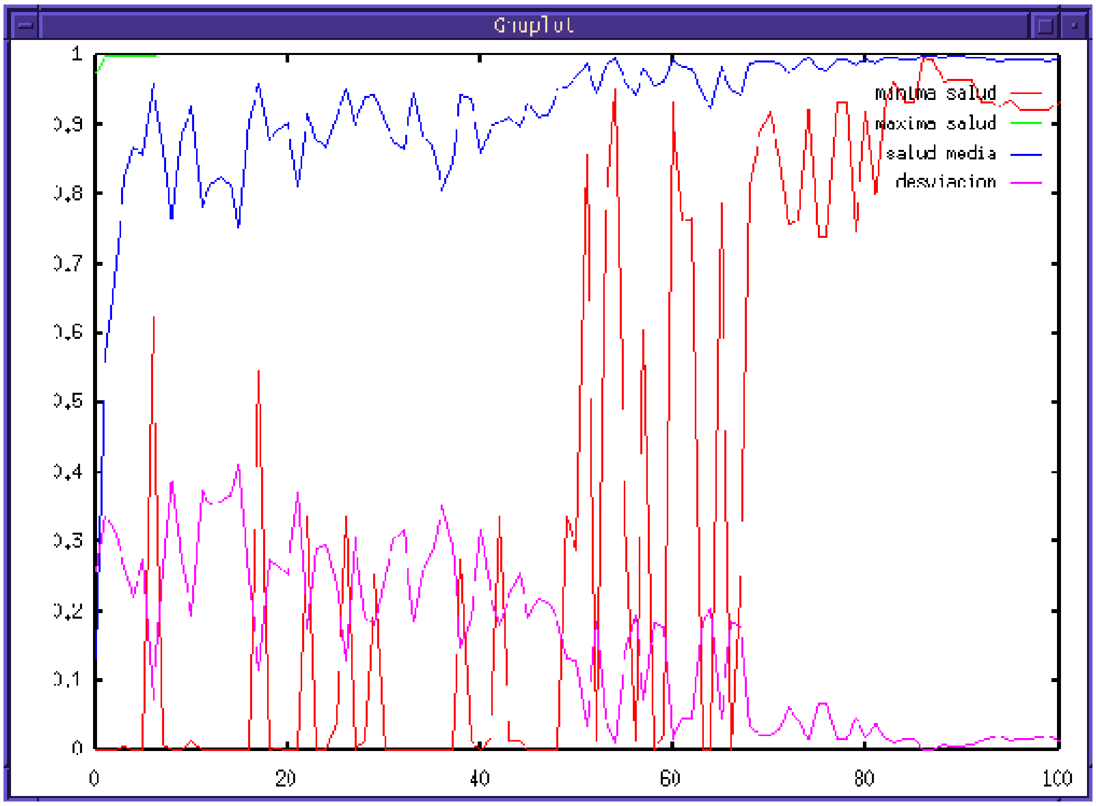
En el programa `ejemplo1.c` cada número del 0 al 2e16 (=65 536) es un individuo, su codificación para SUGAL es un conjunto de 16 genes binarios, cada bit es un gen. La función a maximizar es la de la figura 1, una campana de gauss centrada en 23000 y con desviación típica 10000. Cada vez que SUGAL necesite saber la salud de un individuo llamará a la función `evaluate()`, que le entrega el valor de la campana en ese punto.

Algoritmo básico de SUGAL.

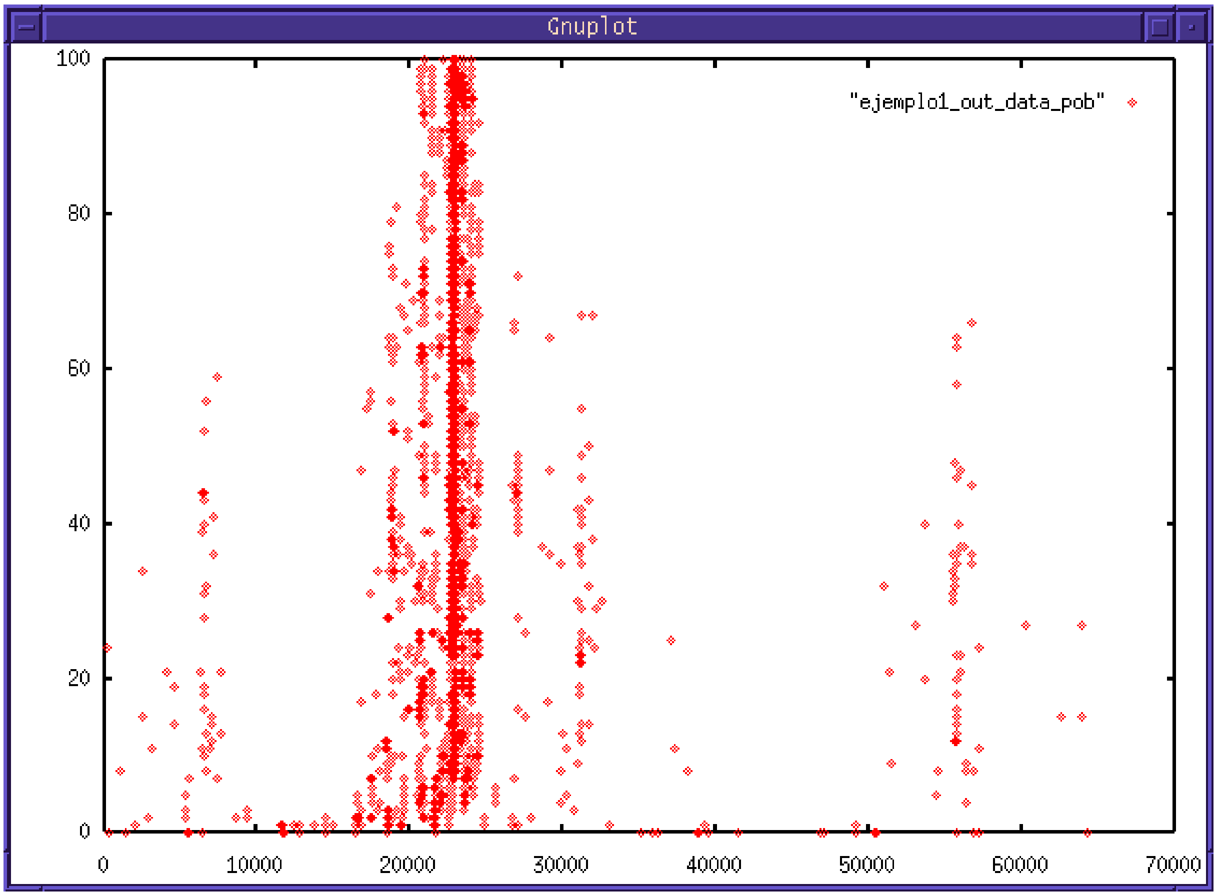
Esquemáticamente el proceso que sigue SUGAL para llevar a cabo la evolución de poblaciones es el siguiente:

```
 1. Inicializar la primera poblacion.
 2. Evaluar la salud de los individuos y normalizarla.
 3. Volcar estadisticas de primera poblacion a fichero

 4. Para cada generacion sucesiva:
             Seleccionar los padres.
             Aplicar los operadores geneticos para generar los hijos.
             Reemplazar la poblacion antigua con los hijos.
             Evaluar salud de los individuos y normalizarla.
             Volcar estadisticas de la poblacion al fichero de salida.
             Ver si se dan las condiciones de final de evolucion.
```



Ejecutando el `ejemplo1` genera el fichero de salida `ejemplo1_out`. La evolución de la salud de la población con el tiempo se puede ver en la figura 2. En el eje de abcisas se representa el número de generación, en las ordenadas en azul la salud media y en verde la salud máxima. En este caso se observa como en una generación temprana algún individuo de la población se aproxima mucho al máximo, por ello la linea verde sube muy pronto a su máximo. También se observa una tendencia creciente en la media, que se desplaza más erráticamente hacia saludes crecientes a medida que más individuos de la población se acercan al máximo.



La evolución de la población para el `ejemplo1` se puede observar en la figura 3 donde en el eje vertical tenemos el número de generaciones, y en el eje horizontal tenemos el dominio de la función con un punto rojo por cada individuo de la población, localizado dentro de ese dominio. Se puede ver que al comienzo toda la población está repartida uniformemente por todo el dominio pero a medida que pasa el tiempo, esto es, a medida que avanzan las generaciones, los puntos tienden a hacercarse al máximo de la función que está en las proximidades del 23000. Al final de la evolución casi todos los individuos se concentran entorno a ese punto, la población converge al máximo.

Inicialización de la población.

SUGAL ofrece dos tipos de inicialización la uniforme y la gaussiana, que se eligen con el parámetro **init**: *[uniform, gaussian]*.

- Para tipos numéricos la inicialización uniforme distribuye los individuos (cada uno de sus genes) equiprobablemente dentro del rango entre **init-minimum** e **init-maximum**. Para tipo binario esta inicialización asigna el 1 a cada bit con la misma probabilidad que el 0.

- Para tipos numéricos la inicialización gaussiana distribuye los individuos de la primera población según una normal de media **init-mean** y desviación **init-sd**. Para el tipo binario esta inicialización pone cada bit a 1 con una probabilidad **init-mean**. Así con **init**=*gaussian* y **init-mean**=*0.8* se puede sesgar la población inicial hacia el valor máximo, sus bits tienden a valer 1 inicialmente.

En el `ejemplo1` se han iniciado los bits uniformemente y esto se observa en la figura 3 como una densidad de puntos homogenea a lo largo del dominio para la primera generación.

Selección de padres.

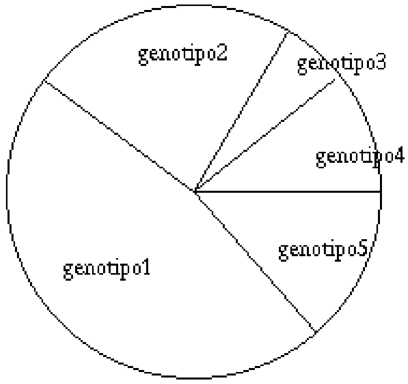
Buena parte de la tendencia a la mejora de las sucesivas poblaciones se explica mediante la selección de los individuos mejor adapatados para procrear la siguiente generación. De este modo los genes mejores perviven de una población a la sucesora, mientras que los genes malos, que hacen a los individuos peores soluciones, desaparecen al no reproducirse los individuos que los portan. En la naturaleza esta etapa englobaría los combates prenupciales que se dan en muchas especies animales y sobre todo la criba que los predadores o las condiciones ambientales hacen de los ejemplares débiles o mal adaptados, de manera que los descartan para la reproducción.

En los AG es muy frecuente que esta selección se lleve a cabo con el **método de la ruleta**: cada individuo tiene una probabilidad de ser seleccionado como progenitor proporcional a su salud. De este modo los individuos mejor adaptados, las mejores soluciones tienen más posibilidades de ser elegidas.

En SUGAL el método de selección se elige dando el valor adecuado al parámetro **selection**. Además de la ruleta SUGAL ofrece otras tres técnicas:

- *roulette*: El método de la ruleta. Según muestra la figura 4, la probabilidad que tiene un genotipo o individuo de ser elegido es su *salud normalizada*, que es el porcentaje de salud que tiene su genotipo respecto del total de la población. Cuenta con el inconveniente de tener que calcular esta fracción para cada individuo de la población.

- *uniform*: Los padres se eligen aleatoriamente entre la generación actual. Todos los individuos de la población, independientemente de su salud, tienen equiprobabilidad de ser elegidos. Esta técnica no garantiza que la población tienda hacia mejores soluciones puesto que no discrimina los individuos con genes malos, pero se ha incluido con objeto de realizar búsquedas puramente aleatorias y poder comparar eficiencias.



METODO DE LA RULETA

- *tournament*: En la **selección por torneo** se escogen aleatoriamente grupos de individuos, y de cada grupo sólo se selecciona como padre el mejor, el que gana ese "torneo". El tamaño de los grupos entre los que se establece el torneo se selecciona con el parámetro `selection_tournament_size`.

- *integral_roulette*: Parecida a la ruleta, pero menos "aleatoria". Se garantiza que un individuo tendrá al menos tantos hijos como la parte entera de su *salud normalizada*. Por ejemplo si se crean 100 hijos nuevos en cada generación y un individuo tiene el 2'5 % de la salud global de su población por lo menos tendrá 2 hijos. Con el 0'5 % restante de salud participará en una ruleta con las partes decimales de los otros ejemplares compitiendo para generar los hijos que restan por crear en esa generación.

¿Cuántos padres se seleccionan?

En SUGAL se seleccionan tantos padres como hijos se quieran generar. El número de hijos que se conciben en cada generación , y por lo tanto también de padres que se seleccionan, se especifica en SUGAL con el parámetro `replacement_rate`. Esta variable se interpreta conjuntamente con `replacement_rate_type`. Si `replacement_rate_type` vale:

- *absolute*, entonces `replacement_rate` se interpreta como el número exacto de hijos que nacen en cada generación.

- *proportion*, entonces el numero de hijos en cada generación es el porcentaje que marque `replacement_rate` sobre la población total (OJO `replacement_rate` varia de 0 a 1).

- *absolute_kept*, entonces el numero de hijos en cada generación coincide con la población total menos `replacement_rate`, que en este caso especifica el numero de individuos que no son padres en cada generación.

En el `ejemplo1` se ha elegido el tradicional método de la ruleta, reemplazando en cada generación al 80 % de la población.

Operadores genéticos: cruce y mutación.

El grupo de padres seleccionados engendra gracias a los operadores genéticos los nuevos hijos. En SUGAL estos operadores son dos: el cruce y la mutación, que se aplican secuencialmente. Los padres se toman de dos en dos para generar mediante cruce dos hijos, y posteriormente se aplica el operador mutación a cada uno de estos hijos. Si el numero de hijos necesarios es impar se selecciona un padre más y del ultimo cruce se descarta uno de los hijos. Si no se puede coger otro padre, no se utiliza cruce para engendrar a este último hijo.

Cruce

SUGAL ofrece varios tipos de cruzamiento, eligiendo el valor adecuado para el parámetro `crossover`:

- *onepoint*: es el cruce clásico. Se escoge al azar un punto de corte en el cromosoma del padre y de la madre, y se intercambian los finales de las cadenas cromosómicas. SUGAL cuida que el punto de corte caiga siempre separando genes, nunca dentro de un gen.
- *twopoint*: se eligen dos puntos aleatorios de corte y para generar los hijos se intercambian los genes entre ellos.
- *npoint*: se eligen varios puntos de corte (`crossover_points`) y se van intercambiando trozos de cadena cromosómica alternativamente.
- *uniform*: los hijos cogen cada gen con igual probabilidad del padre que de la madre.

Este operador permite explorar nuevas combinaciones de los genes ya existentes en la población. Está intimamente relacionado con la reproducción sexual de muchos seres vivos, a diferencia de la reproducción asexual en la cual solamente interviene la mutación de una generación a la siguiente.

SUGAL permite aplicar probabilísticamente el operador cruzamiento, es decir para generar cada pareja de hijos se crucen sus padres con cierta probabilidad o no se crucen, directamente se copien. Esta probabilidad se determina con el parámetro `crossover_rate`, y por defecto es 1, esto es, se aplica siempre. En el `ejemplo1`, según se puede observar en su fichero de configuración `ejemplo1_cfg` hemos elegido un cruce sencillo, de un sólo punto, que se aplica en todo nacimiento de nuevos hijos.

Mutación

Una vez cruzados los padres, se aplica una mutación a los genes de cada hijo generado. Este operador introduce una variación aleatoria en alguno de los genes. SUGAL ofrece distintos tipos de mutación, dependiendo del parámetro `mutation`:

- *invert*: Se aplica a genes binarios, cambia el bit.
- *reinit*: Válido para todo tipo de genes, lo reinicializa según esté programada la inicialización: uniforme, gaussiana ....
- *step*: Válido solo para tipos numéricos. Le suma o resta un paso al gen. La longitud del paso se especifica en el parámetro `mutation_size`.
- *uniform*: Le suma un paso aleatorio al gen. El paso aleatorio se obtiene de una distribución

uniforme entre [-mutation_size, +mutation_size]. Válido sólo para genes numéricos.

- *gaussian*: El paso que le suma al gen, se obtiene de una distribución gaussiana de media cero y desviación mutation_size. De este modo normalmente se incrementa el gen en valores pequeños, pero ocasionalmente hay saltos grandes.

El operador mutación es probabilístico, con una probabilidad mutation_rate se aplica efectivamente la mutación y en el resto de los casos deja intacto a gen, o al individuo. Modulando este parámetro se hacen más o menos frecuentes las mutaciones. mutation_rate se puede interpretar como probabilidad de alterar un gen, o probabilidad de alterar un individuo, dependiendo de que mutation_rate_type tome el valor *per_gene* o *per_chromosome*.

Este operador es una fuente de nuevos genes en la población, en la búsqueda de buenos genes que es el AG, y normalmente está muy relacionado con la convergencia de la población.

Reemplazo.

En un enfoque simple se generan tantos hijos como individuos haya en la población, y la generación de hijos sustituye plenamente a la población anterior. Sin embargo SUGAL permite muchas más variantes. Una vez que se han generado los hijos, estos todavía no pertenecen a la población. Además pueden generarse menos hijos que la población total, como vimos con el parámetro replacement_rate. Para SUGAL éstos hijos generados forman un grupo de *candidatos* a entrar en la población. Por otro lado SUGAL elige un conjunto de individuos de la población que son *reemplazables*, es decir a estos y no a otros sustituirán los nuevos hijos, si esa sustitución procede. Ambos conjuntos tienen la misma cardinalidad. Siguiendo con un enfoque simple cada hijo podrá entrar en la población sólo sustituyendo a otro individuo con peor salud que él. Con lo cual este reemplazo constituye otro pilar de la mejora sucesiva de las poblaciones, la verdadera selección natural. No es que las soluciones malas no entren en el juego de la reproducción, es que se *mueren* literalmente, al ser eliminadas de la población.

En este aspecto la naturaleza y SUGAL difieren. Mientras que en la naturaleza las poblaciones pueden crecer, y es el medio el que va eliminando individuos, en SUGAL las poblaciones tienen un tamaño fijo, de modo que para que un nuevo individuo entre a formar parte de ella debe reemplazar a otro.

SUGAL aborda el reemplazo con dos parámetros: replacement determina como se eligen los individuos *reemplazables*, y replacement_condition determina cuando un candidato sustituye a un *reemplazable*. En cuanto a la elección de individuos reemplazables, replacement puede valer:

- *uniform*: el conjunto de reemplazables se rellena al azar de individuos de la población, donde cada uno de ellos tiene equiprobabilidad de ser elegido.

- *ranked*: los individuos en la población se ordenan de mayor a menor salud. Si se necesita un conjunto de *n* reemplazables, lo forman los *n* individuos de peor salud de la población.

- *tournament*: Se establecen *n* torneos, similares a los que se pueden hacer para seleccionar padres, y el individuo con peor salud de cada torneo se escoge como reemplazable. Los torneos eligen replacement_tournament_size individuos al azar, y de ellos el de peor salud "gana" el torneo.

- *crowding*: para cada candidato se establece un torneo entre la población, y como reemplazable por ese candidato se elige el individuo más parecido a él del torneo.

- *parental*: cada hijo se asigna a uno de los padres, y es precisamente su padre el individuo reemplazable para ese candidato. De este modo los hijos solo reemplazan a los padres, si procede.

En cuanto al criterio que dicta si un candidato sustituye a un reemplazable, SUGAL presenta las siguientes ofertas, como valores del parámetro `replacement_condition`:

- *unconditional*: el hijo siempre sustituye al reemplazable, independientemente de la salud de ambos.

- *if_improved*: el candidato sólo sustituye al reemplazable si su salud es mejor. Esta técnica introduce un claro sesgo en la evolución hacia soluciones cada vez mejores, no permitiendo que soluciones peores se incorporen a la población.

- *annealed*: parecido al caso anterior, pero más aleatorio. El individuo reemplazable siempre se cambia por el hijo más saludable que él, pero se permite que también se sustituya, con cierta probabilidad, por un hijo de peor salud. Esa probabilidad de aceptar un hijo peor disminuye con el tiempo según el parámetro `annealing_decay`

En el `ejemplo1` se seleccionan como individuos reemplazables los de menor salud de toda la población (`replacement` = *ranked*, y se sustituyen por los hijos siguiendo una política de recocido ("annealed").

Condiciones de parada.

En el `ejemplo1` la evolución se detiene cuando se llega a 100 generaciones, según se especifica la variable `generations`. Sin embargo SUGAL proporciona otras condiciones de parada que pueden terminar el proceso evolutivo. Se elige una u otra seleccionando el valor adecuado del parámetro `stop`:

- *fitness_target*: la evolución acaba cuando algún individuo de la última generación iguala o supera la salud marcada como umbral en el parámetro `fitness_target`.

- *fitness_convergence*: la evolución se detiene cuando todos los individuos de la última generación tienen aproximadamente la misma salud. Concretamente cuando la desviación estandard cae por debajo del umbral `fitness_convergence_level`. Básicamente capta la idea de que cuando todos las soluciones sean aproximadamente iguales, converjan en la misma zona del espacio de búsqueda, y por lo tanto su salud sea muy similar, entonces en esa zona está el máximo de salud, y el AG se detiene.

- *diversity_convergence*: es similar al anterior, la evolución se acaba cuando todos los puntos de la población están muy próximos en el espacio de soluciones. Sin embargo la proximidad de individuos no se mide indirectamente a través de saludes similares, sino que se hace a través de una función de diversidad o distancia. Cuando la diversidad media de la población cae por debajo del umbral `diversity_convergence_level` el AG se detiene. Por ejemplo para individuos binarios la diversidad se media es la distancia Hamming media entre puntos de la población.

- *generations*: la evolución termina cuando se llega a un determinado número de generaciones, que se especifica en el parámetro `generations`

La condición de parada activa se evalúa una vez creada cada nueva generación. Si no se satisface se procede a crear la siguiente generación dando otra vuelta de tuerca al proceso evolutivo, a la búsqueda genética. Si la condición se presenta entonces el AG se detiene, SUGAL vuelca resultados a su fichero de salida si procede y devuelve el control al programa usuario, que regresa de la llamada a la función de inicio `SuRun`. El nombre del fichero en el cual se vuelcan resultados se especifica en el parametro `output_file`, y se habilita poniendo `file_output` a *on*

Reportajes.

El interfaz del usuario con SUGAL son ficheros. En un fichero se especifican a SUGAL sus parámetros de configuración y el programa vuelca en otro fichero sus resultados. Existe una versión del paquete con interfaces gráficas sobre la librería Motif, pero no lo comentaremos en este trabajo.

SUGAL puede registrar las estadísticas de la salud de la población a medida que avanza el proceso evolutivo: la evolución de la salud media, la salud máxima y la mínima de los individuos en cada población, así como la diversidad media de cada generación. Poniendo a *on* el parámetro `report_pool_stats` e igualmente el valor de los parámetros `report_fitness_mean`, `report_fitness_max`, `report_fitness_min` y `report_fitness_sd`, según interese, SUGAL volcará estas características en el fichero de salida. Por defecto SUGAL saca estas estadísticas para cada generación; si se quiere que lo haga cada 5 o 10 generaciones el parámetro `report_interval` con "periodo" 5 o 10, o cualquiera que deseemos.

Además de las estadísticas de salud de cada población SUGAL permite volcar en el fichero de salida todas y cada una de las poblaciones, esto es, los individuos que componen cada generación. Esto se consigue poniendo a *on* el parámetro `report_every_pool`. Si sólo se quiere analizar la última población basta activar el parámetro `report_final_pool`.

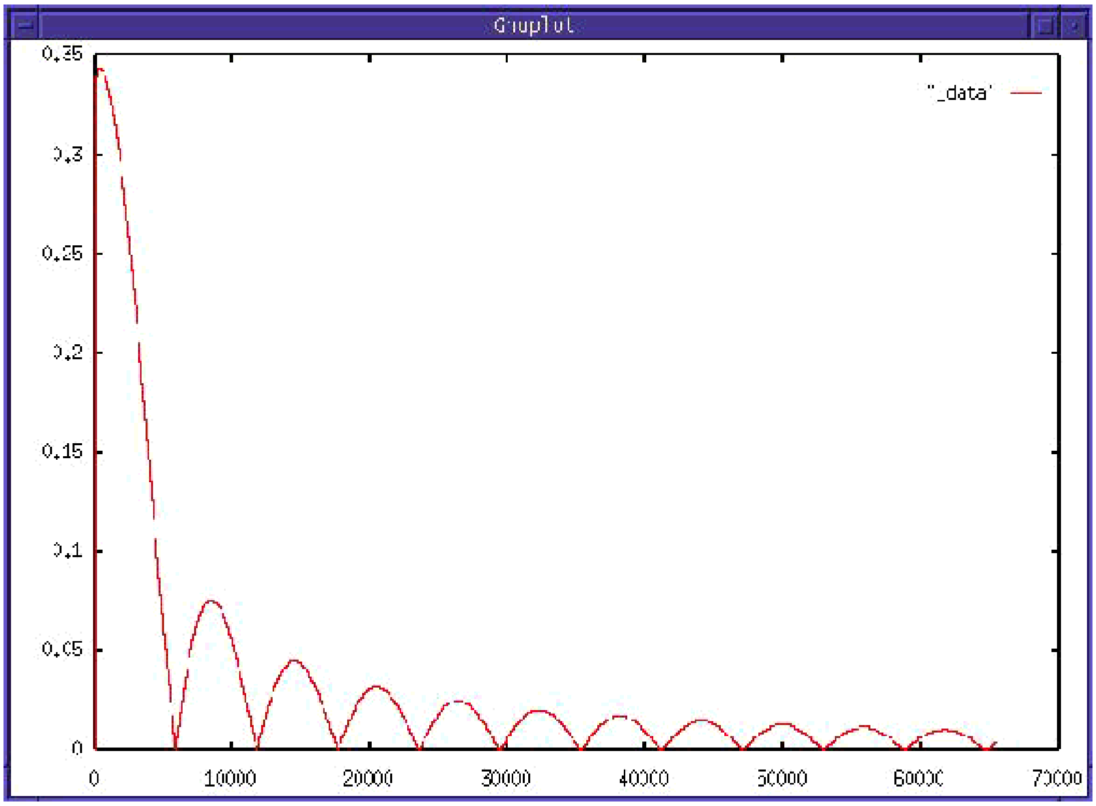
SUGAL también permite volcar los parámetros de configuración con los que fue lanzado, activando el parámetro `report_settings`. En el `ejemplo1` se ha activado el volcado de los reportajes con la salud media, mínima, máxima y desviación de la salud para cada generación y el volcado de los individuos de todas las generaciones. Sólo esta información aparece en el fichero de salida `ejemplo1_out` una vez que se ha ejecutado el `ejemplo1`.

IV. PRÁCTICA BÁSICA CON SUGAL

En el anterior capítulo hemos introducido el modo en que se utiliza la herramienta SUGAL y sus parámetros más básicos de configuración, presentando un primer ejemplo muy sencillo de utilización, `ejemplo1.c`. En este apartado presentaremos más ejemplos concretos con los cuales ilustraremos los efectos de algunos parámetros significativos en el comportamiento del AG. También probaremos a lanzar el AG de SUGAL con otras funciones de salud distintas de la gaussiana y comentaremos el problema de la convergencia prematura. Para ello hemos preparado tres programas de apoyo que permiten ejecutar automáticamente el AG con SUGAL y muestran cómo evolucionan los individuos y su salud dentro de cada población. Estos programas de apoyo son tres: funciondex, poblaciones y salud. Todos corren sobre plataformas un*x y necesitan de un interprete `perl` y del programa de gráficos `gnuplot`.

Visualización de distintas funciones y llamadas automáticas a SUGAL.

El programa **`funciondex`** permite llamar automáticamente a SUGAL con distintas funciones de salud y visualizar estas funciones gracias al programa `gnuplot` que debe existir en la máquina. Se ha preparado un repertorio de cinco funciones: `gaussiana`, `camello`, `sinc`, `constante` y `michalewicz`, para ilustrar distintos entornos dentro de los cuales el AG debe encontrar el máximo. Para visualizar la función `camello` por ejemplo basta con invocar "`funciondex -v -fcamello`", donde el parámetro `-v` indica visualización y `-f` indica el nombre de la función a mostrar. Igualmente se pueden visualizar las otras funciones. Además se puede modular la precisión con la cual se visualizan las funciones gracias al parámetro `-bBITS` donde `BITS` es el numero de bits con el cual se quiere visualizar la función, siendo 8 por defecto. Así por ejemplo para ver la función `sinc` con precisión de 16 bits hay que invocar "`funciondex -v -fsinc -b16`".



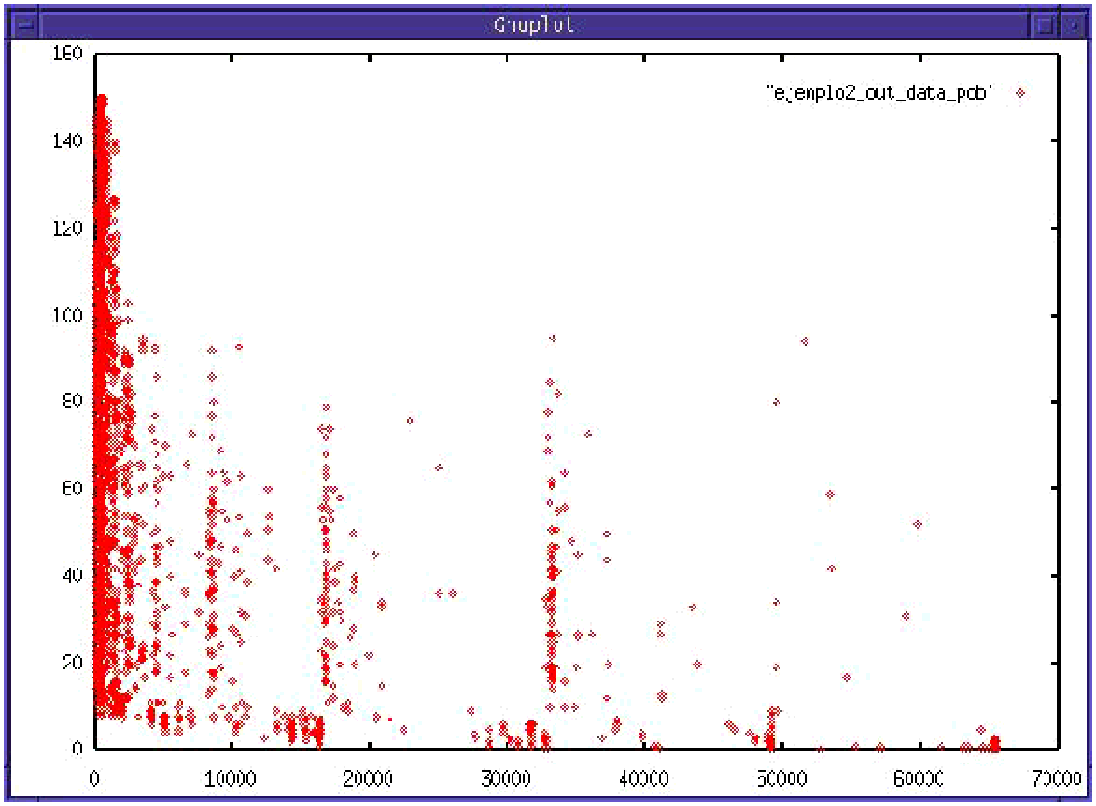
Además de mostrar en pantalla distintas funciones el programa **funciondex** permite llamar a SUGAL con esas mismas funciones. Basta invocar "`funciondex -s -fsinc ejemplo2_cfg`", donde el parámetro `-s` indica que debe llamar a SUGAL en lugar de visualizar, `-fsinc` señala que se utiliza la función `sinc` como función de salud del AG, y `ejemplo2_cfg` espefica el fichero del que se toma la configuración de SUGAL. Si no se le indica nada `funciondex` coge por defecto el fichero de configuración `funciondex_cfg`. LLamando a "`funciondex -help`" aparece un pequeño manual de utilización del programa.

Para que **funciondex** pueda llamar correctamente a SUGAL necesita operar con individuos binarios. Para calcular la salud de cada individuo `funciondex` convierte el individuo binario en un valor numérico al que aplica la función de salud escogida, que siempre es unidimensional. Como invocador automático de SUGAL este programa necesita que el usuario le indique el fichero de configuración y el nombre de la función que se SUGAL tomará como función de salud.

Las funciones del repertorio están parametrizadas en el código fuente, si se quieren modificar suss parámetros se puede retocar el programa fuente, recompilar y ejecutar para llamar a SUGAL con esas funciones modificadas. Por ejemplo se puede situar la gaussiana en otro punto, cerrar un poco su campana, mover uno de los dos picos de la función camello, variar la frecuencia de la función sinc, etc. Prueba a visualizar las cinco funciones preparadas y si quieres, a modificar alguno de sus parámetros principales.

Evolución de los individuos dentro del dominio.

En el `ejemplo2_cfg` hemos preparado un caso para ejecutar con la función `sinc`. Una vez llamado "`funciondex -s -fsinc ejemplo2_cfg`", SUGAL genera el fichero de salida `ejemplo2_out`. Hemos creado el programa **poblaciones** que permite ver la evolución de los individuos dentro del dominio a lo largo del paso de generaciones. Se invoca con el fichero de salida de SUGAL como único argumento. Así llamando a "`poblaciones ejemplo2_out`" podemos ver cómo se han ido moviendo los individuos de la población a medida que transcurrían las generaciones. En la figura 5 se puede observar la función a maximizar, valor absoluto de `sinc`, y en la figura 6 la evolución seguida por el AG.



En el eje vertical tenemos el número de generación y en el horizontal el dominio, con un punto por cada individuo de la población. Inicialmente los individuos de la población aparecen repartidos preferentemente en los valores altos, pues la inicialización es sesgada hacia los bits 1. Rápidamente se deplazan hacia las cercanías del máximo que está en el origen, en cuyos alrededores se concentra la mayoría de la población excepto algunos individuos díscolos. Para terminar, cerca de la generación 100 toda la población se superpone en un enorno muy reducido del origne, convergiendo en ese punto, el máximo de la función `sinc`.

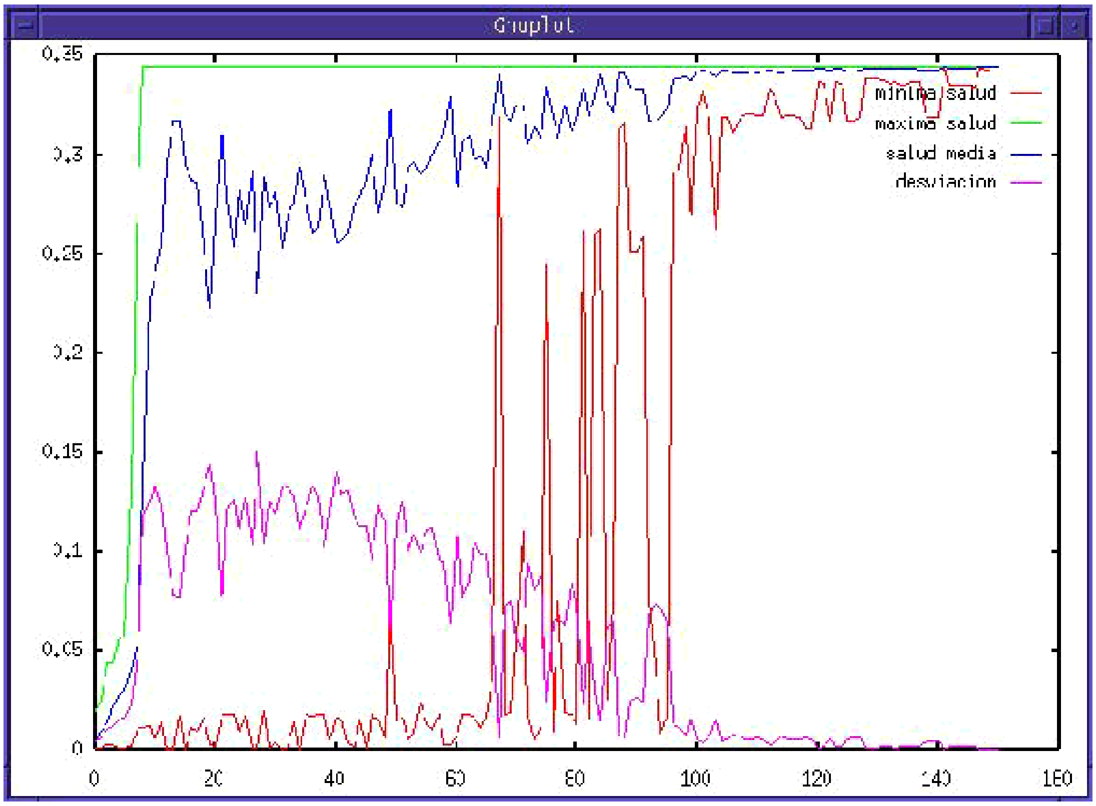
Modificando el fichero `ejemplo2_cfg` prueba a inicializar la población de modo uniforme y a sesgar la población inicial hacia los valores más pequeños del dominio, esto es, dando más peso al 0 en la inicialización. Llama a SUGAL en cada caso con `funciondex` y visualiza la evolución con `poblaciones`. Verifica que efectivamente la población inicial se distribuye equiprobablemente o concentrada en los valores pequeños. Verifica también que independientemente de la población de arranque el AG llega al máximo de la función.

Para que el programa **`poblaciones`** funcione correctamente los parámetros del fichero de configuración de SUGAL `report_pool_stats` y `report_every_pool` deben estar a *on*.

Evolución de la salud.

Hemos preparado el programa **`salud`** para visualizar la evolución de la salud de los individuos a lo largo de las poblaciones. Permite observar la salud máxima de cada generación, la mínima, la media y la desviación típica de la salud dentro de la población. Se invoca con el fichero de salida de SUGAL como único argumento. Así llamando a "`salud ejemplo2_out`" podemos ver cómo han ido evolucionando las estadísticas de salud a medida que el AG avanzaba. En el eje de abcisas tenemos el número de generación, el paso del tiempo. En el eje vertical tenemos el nivel de salud, en verde la salud máxima que se puede encontrar entre todos los individuos de una generación, en rojo la salud mínima, en azul la salud media ponderada entre toda la población y en violeta la desviación media en la distribución de salud dentro de cada población.

Para que el programa **`poblaciones`** funcione correctamente los parámetros del fichero de configuración de SUGAL `report_pool_stats` y `report_every_pool` deben estar a *on*.

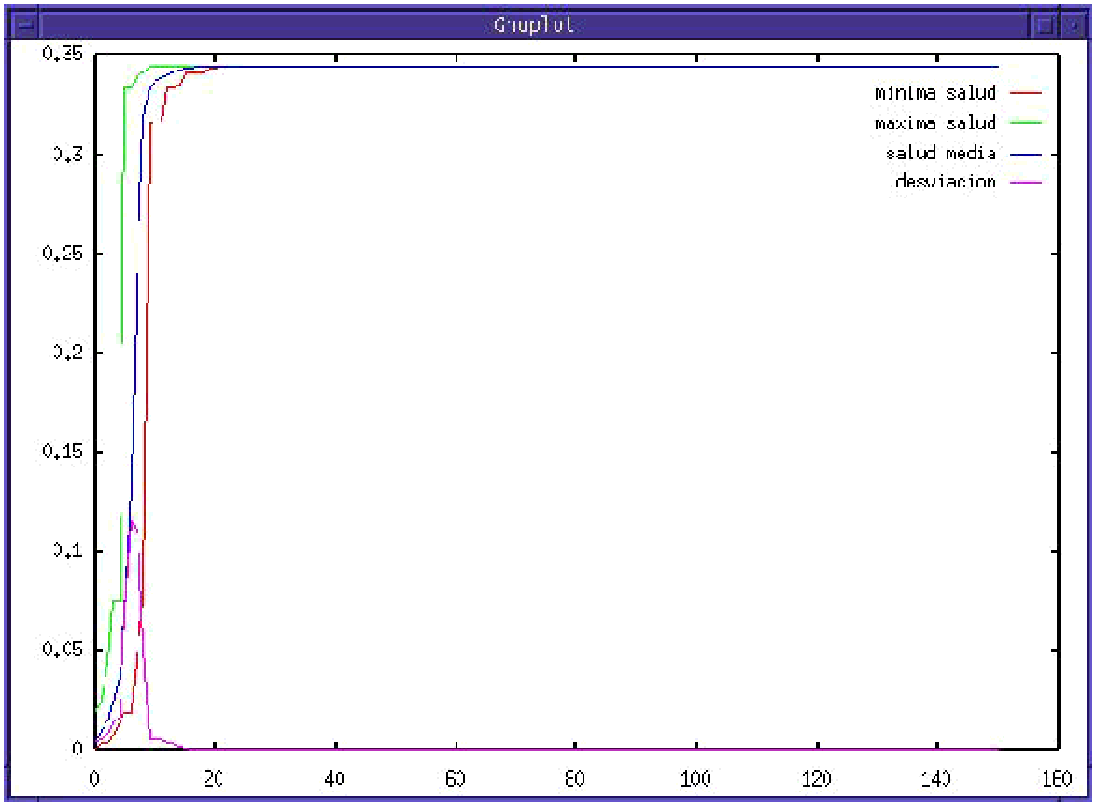


En el caso de la figura 7, obtenida con el `ejemplo2_cfg` y la función `sinc`, se puede observar como la desviación típica es pequeña al comienzo, al estar la inicialización de los individuos sesgada hacia el bit 1, casi todos los individuos son parecidos. La desviación en la función de salud se puede interpretar como una medida de la diversidad dentro de cada población. Después de una subida brusca de la salud y de la diversidad hacia la sexta generación, se aprecia una zona con una tendencia creciente y sostenida de la salud media, aunque que con vaivenes. En esta etapa mantiene inicialmente una diversidad notable para ir disminuyendo paulatinamente. Finalmente la salud media se acerca mucho a la salud máxima, la desviación se anula practicamente y la salud mínima coincide con la máxima. Entonces toda la población se ha estabilizado y está constituida por el mismo genotipo, el AG ha finalizado.

En este `ejemplo2_cfg` se ha elegido un reemplazo recocido ("*annealed*") de modo que al principio las sustituciones son casi incondicionales y a medida que avanza el tiempo tiende a reemplazarse sólo si el hijo es de mayor salud. En la zona intermedia de la evolución, cuando todavía la población no se ha estabilizado y las sustituciones son casi incondicionales nadie garantiza que los nuevos individuos sean mejores que sus progenitores. Por ello pueden darse estos altibajos en la evolución de la salud media. Incluso pueden darse en la salud máxima si el genotipo que tenía la salud mayor es reemplazado por un hijo y entre estos descendientes no hay ninguno mejor que él.

Prueba a visualizar evolución del la salud con SUGAL configurado según `ejemplo3_cfg` y tratando de maximizar igualmente la función `sinc`. Con respecto al `ejemplo2_cfg` se ha alterado el reemplazo, ahora un hijo sólo reemplaza a otro individuo si tiene mejor salud que él (`replacement_condition` = *if_improved*). De esta manera la salud media, la maxima y la mínima siempre serán funciones monotonamente crecientes, pues sólo se incorporan nuevos individuos si mejoran la salud de alguno ya existente. Esto se puede observar en la figura 8. Además se puede ver como enseguida toda la población llega al valor máximo de salud, converge muy rápidamente, la desviación típica se anula a partir de la generación decimoquinta. Esto no siempre es deseable según veremos posteriormente.

Prueba también a visualizar con un reemplazo incondicional. Aunque se localiza el máximo, la convergencia de la población no es tan clara.

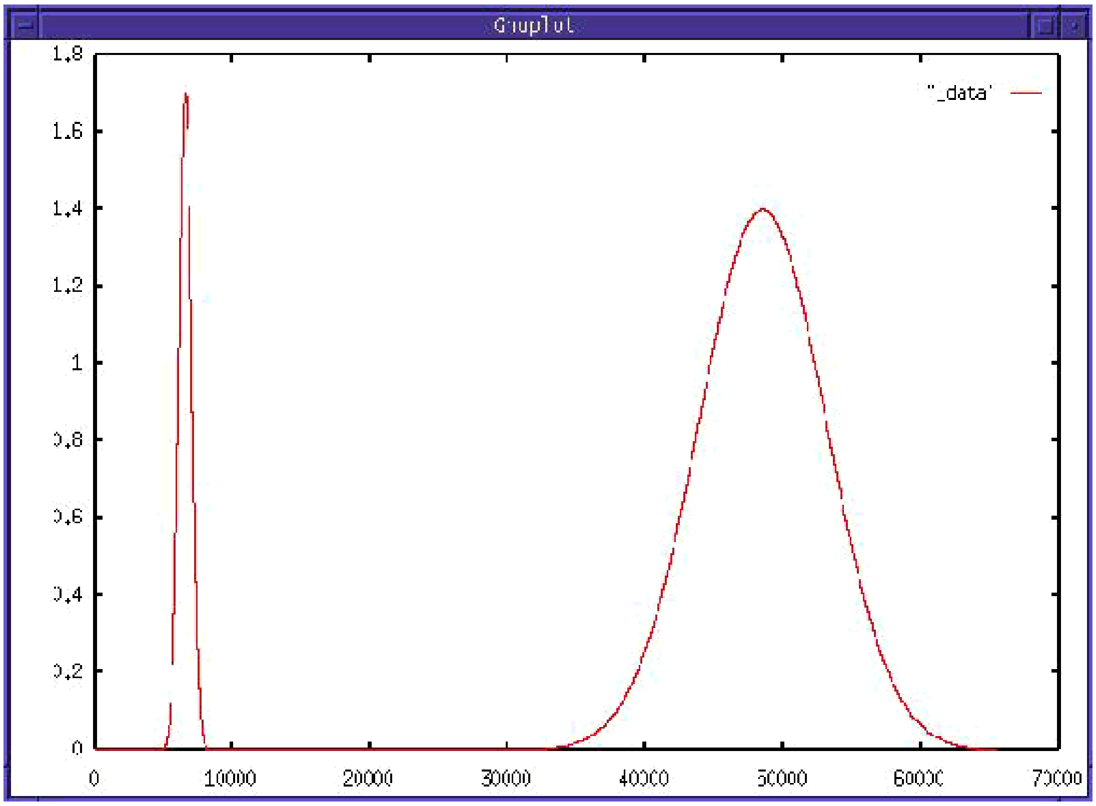


Prueba a variar el fichero de configuración `ejemplo2_cfg` para introducir **elitismo** en el AG. Poniendo el parámetro `elitism` a *on* SUGAL asegura que el individuo de mejor salud de una población pervive y pasa a la generación siguiente. Por ello se garantiza que la salud máxima nunca decrece, aunque sí lo puedan hacer la salud media y la mínima.

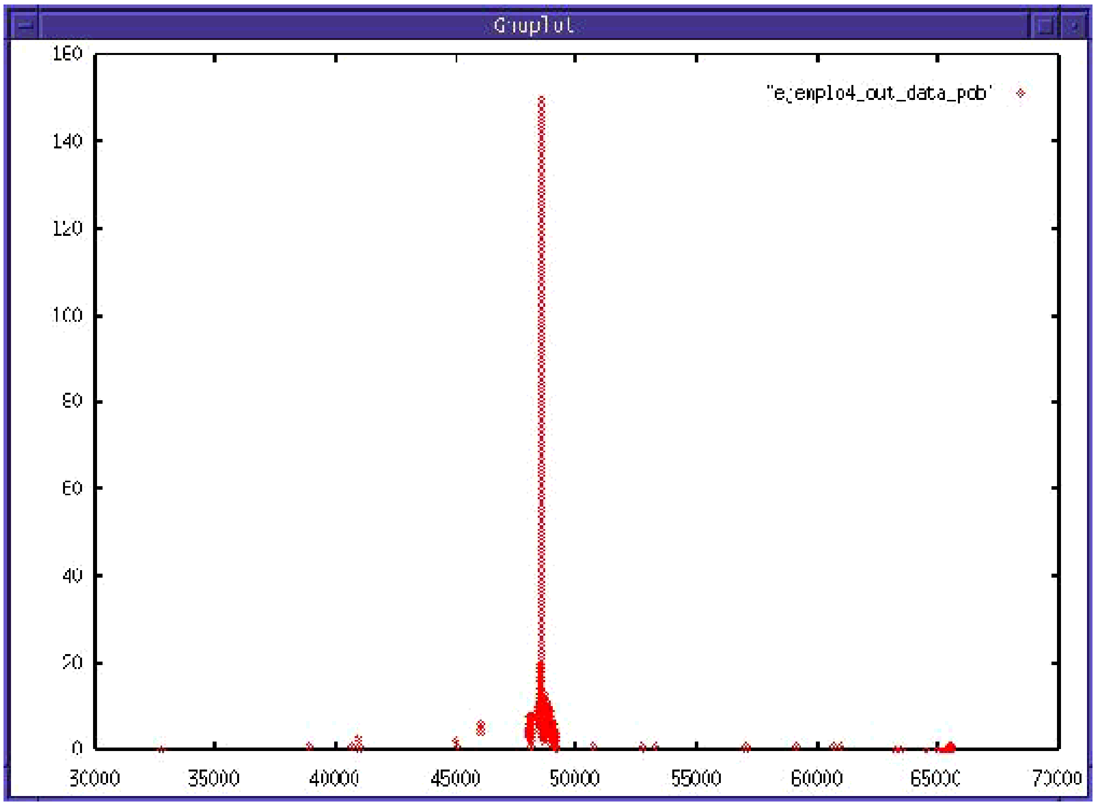
Prueba a variar los parámetros de reemplazo y de selección de *reemplazables* para ver su impacto en la evolución de la salud. En general reemplazando los individuos peor adaptados, e incluso efectuando la sustitución sólo si el candidato es mejor se logra acelerar la convergencia del algoritmo, a la vez que se pierde poder de exploración del AG. El AG busca por menos zonas del dominio, pues los individuos de baja salud difícilmente entran a formar parte de la población. Esos individuos se pierden como respuesta tentativa del AG, que no es del todo malo pues son individuos pésimos, pero también se pierden como zona del dominio alrededor del cual buscar, como fuente de algún buen gen.

Convergencia prematura y máximos locales.

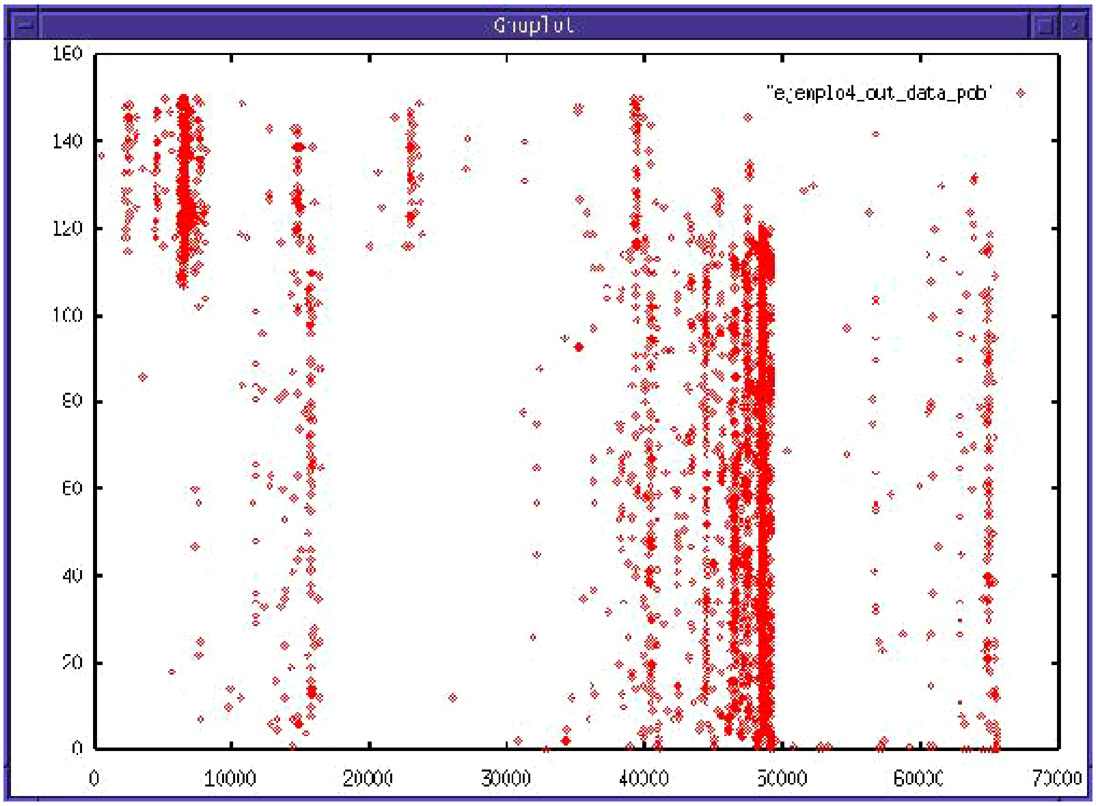
Uno de los problemas principales de acelerar demasiado la convergencia del AG, es que no haya explorado suficientemente el dominio y se sitúe en un máximo local, es decir aquél en cuyo entorno cercano la función es menor, pero que no es el máximo de la función en el dominio. Es lo que se conoce como **convergencia prematura** del AG. Para ilustrar este caso hemos desarrollado el `ejemplo4_cfg` tratando de maximizar la función `camello` que se puede ver en la figura 9.



Visualiza la función camello con "`funciondex -v -fcamello -b16`" y observa que tiene dos gaussianas, una centrada en 50000 y otra más alta cerca de 10000. Lanza SUGAL con "`funciondex -s -fcamello ejemplo4_cfg`" y observa la evolución de la población. Según muestra la figura 10 a partir de la generación 10 el grueso de la población converge en el máximo de la gaussiana más baja, sin encontrar por lo tanto el máximo de la función camello. Hay convergencia prematura.



Contra la convergencia prematura hay varias técnicas como aumentar la mutación, como admitir nuevos puntos en la población aunque sean ligeramente peores que los actuales, y como distribuir aleatoria y uniformemente la población inicial en el dominio. De hecho cambiando la condición de reemplazo de *if_improved* a *unconditional* en este `ejemplo4_cfg` se consigue que el AG localice el verdadero máximo de la función camello, y converja a él. Pruebalo. Observa el cambio en la evolución y en el punto final de convergencia del AG en la figura 11.



V. BIBLIOGRAFIA.

1. Hunter, A.,*SUGAL User Manual, v2.1*, (http://www.trajan-software.demon.co.uk/sugal.htm).
2. Hunter, A.,*SUGAL Programming Manual, v2.1*, (http://www.trajan-software.demon.co.uk/sugal.htm).
3. Velasco, J.R. y Magdalena, L., *Curso de postgrado "Algoritmos Genéticos: teoría y aplicaciones",curso 1996-97, ETSI Telecomunicaciones, Madrid.*